University of California, Merced

# Multi-Modal Planning for Humanlike Motion Synthesis using Motion Capture

A dissertation submitted in partial satisfaction of the requirements
for the degree Doctor of Philosophy

in

Electrical Engineering & Computer Science

by

**Mentar Mahmudi**

2013

Abstract of the Dissertation

# Multi-Modal Planning for Humanlike Motion Synthesis using Motion Capture

by

**Mentar Mahmudi**
Doctor of Philosophy in Electrical Engineering & Computer Science
University of California, Merced, 2013
Professor Marcelo Kallmann, Chair

Planning the motions of a virtual character with high quality and control is a difficult challenge. Striking a balance between these two competing properties makes the problem particularly complex. While data-driven approaches produce high quality results due to the inherent realism of human motion capture data, planning algorithms are able to solve general continuous problems with a high degree of control. This dissertation addresses this overall problem with new techniques that combine the two approaches.

Three main contributions are proposed. First, a simple and efficient motion capture segmentation mechanism is proposed based on geometric features that introduces semantic information for organizing a motion capture database into a motion graph. The obtained feature-based motion graph has less nodes and increased connectivity, which leads to improved searches in speed and coverage when compared to the standard approach. In addition, feature-based motion graphs enable a novel inverse branch kinematic deformation technique to be executed efficiently, allowing solution branches to be deformed towards precise goals without degrading the quality of the results.

Second, in order to address speed of computation, precomputed motion maps are introduced for the interactive search and synthesis of locomotion sequences from unstructured feature-based motion graphs. Unstructured graphs can be successfully handled by relying on multiple maps and a search mechanism with backtracking information, which eliminates the need of manually creating fully connected move graphs. Precomputed motion maps can simultaneously search and execute motions in environments with many obstacles at interactive rates.

Finally, a multi-modal data-driven framework is proposed for task-oriented human-like motion planning, which combines data-driven methods with parameterized motion skills in order to achieve human motions that are realistic and that have a high degree of controllability. The multi-modal planner relies on feature-based motion graphs for achieving a high-quality locomotion skill and integrates generic, task-specific data-based or algorithmic motion primitive skills for precise upper-body manipulation and action planning. The approach includes a multi-modal search method where primitive motion skills compete for contributing to the final solution.

As a result, the overall proposed framework provides a high degree of control and, at the same time, retains the realism and human-likeness of motion capture data. Several examples are presented for synthesizing complex motions such as walking through doors, relocating books on shelves, etc.

The dissertation of Mentar Mahmudi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Professor David Noelle

Professor Stefano Carpin

Professor Marcelo Kallmann, Committee Chair

University of California, Merced

2013

*To my parents,*
*Nevin and Mahmudnedim*

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to first thank my advisor Professor Marcelo Kallmann, for his never-ending support and guidance over many years. I would have never been able to finish this undertaking, if it weren't for his continuous motivation and encouragements. His supervision has greatly expanded my intellectual diapason and has made me a more competent man.

I would like to thank Professor Stefano Carpin, for teaching me how to program, for his continuing support, and for inviting me to join Jacobs's USAR team. I thank Professor David Noelle, for the teaching experience I have gained as TA for his AI courses. They have been most help for my academic development. I would like to thank my math mentor Mr. Agim Bukla who taught me everything about mathematics and prepared me for math olympiads. Also many thanks are due to my elementary teacher Mrs. Drita Ferati, who instilled in me the much needed discipline. She helped me appreciate the fruits of hard work and encouraged me to pursue mathematics. I'm also thankful to my boarding school for teaching me how to never quit.

During my years at Merced, I have been fortunate to meet many great friends. In particular, I want to distinguish Oktar Ozgen for his great personality and his boisterous camaraderie. I could have not asked for a better friend and colleague. I would also like to thank Siyu Wu, for his Oriental influence and his immaculate determination and positivism. In addition, I would like to thank my friends and colleagues: Görkem Erinç, Carlo Camporesi, Benjamin Balaguer, Yazhou Huang, Robert Backman, and Nicola Basilico, for all the many activities we have enjoyed together. Without you, Merced would have been just a small peaceful town with a vast and beautiful countryside.

I would like to thank my family for their boundless love and incalculable support. My father, Mahmudnedim, for passing me the desire to appreciate intellectualism and for making sure that I join the company of exemplary people. My mother, Nevin, for her immense love, kind and gracious personality, for telling me to always do the right thing no matter the cost and for showing me the important things in life. My sister, Elmedina, for setting up high standards in our family and for making me do this doctorate. Thank you for your cheerful outlook and your never-ending support. My brother, Abdyl Vahid, who is anatomically, psychologically, socially and genetically the closest person that I will ever meet in my life. Thank you for looking after our parents while I was away and thank you for being a brave young man who did everything that I couldn't do myself.

I thank my late grandfather, Mesut Dukagjini, who was the most important person in my life and who has influenced me immensely. He was a person whom I admired the most, and whose life wisdom and philosophy I wish to carry on.

# Vita

| | |
|---|---|
| 2006-2013 | Ph.D. (Electrical Engineering and Computer Science), University of California (UC Merced), Merced, CA. |
| 2006–2013 | Teaching Assistant, 9 semesters: Introduction to Artificial Intelligence, Introduction to Graphics, Introduction to Computer Architecture and Design, Discrete Mathematics, Introduction to Java Programming, School of Engineering, UC Merced. |
| 2006–2012 | Research Assistant, 5 semesters (under NSF and CITRIS support), Summer Graduate Fellowship, School of Engineering, UC Merced. |
| Nov 2011 | Best Paper Award, Conference on Motion in Games (MIG), Edinburgh, UK. |
| Jun 2006 | 2nd Prize RoboCup Virtual Rescue League (IUB-Team), Bremen, Germany. |
| 2003–2006 | B.Sc. (Electrical Engineering and Computer Science), Jacobs University, Bremen, Germany. |
| Summer 2005 | Intern, Urban Search and Rescue Development, University Pittsburgh, Pittsburgh, PA. |
| Summer 2004 | Intern, Max Planck Institute for Microbiology, Bremen, Germany. |

# Publications and Presentations Related to this Work

## In Preparation

M. Mahmudi and M. Kallmann, "Multi-Modal Data-Driven Motion Planning and Synthesis", *(in preparation)*.

## Publications

M. Mahmudi and M. Kallmann, "Analyzing Locomotion Synthesis with Feature-Based Motion Graphs", *IEEE Transactions in Visualization and Computer Graphics (TVCG)*, May, 2013.

M. Mahmudi and M. Kallmann, "Precomputed Motion Maps for Unstructured Motion Capture", *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (SCA)*, Lausanne, Switzerland, 2012.

M. Mahmudi and M. Kallmann, "Feature-Based Locomotion with Inverse Branch Kinematics", *International Conference on Motion In Games (MIG)*, Edinburgh, UK, 2011, **Best Paper Award**.

Y. Huang, M. Mahmudi and M. Kallmann, "Planning Humanlike Actions in Blending Spaces", *International Conference on Intelligent Robots and System (IROS)*, San Francisco, 2011.

M. Mahmudi and M. Kallmann, "Fast Path Following using Motion Graphs", *Symposium on Interactive 3D Graphics and Games (I3D)*, Poster Paper, Redwood City, 2008.

## Presentations

M. Kallmann, Y. Huang and M. Mahmudi, "Humanlike Motion Planning", *IROS Workshop on Progress and Open Problems in Motion Planning*, Poster Presentation, San Francisco, 2011.

# Other Publications

C. Goulart, M. Mahmudi, S. D. Jacobs, K. Crona, M. Kallmann, B. G. Hall, D. Greene, M. Barlow, "Designing antibiotic cycling strategies by determining and understanding local adaptive landscapes", *PLOS ONE, Vol. 2, Issue 8*, February, 2013.

K. Crona, D. Patterson, K. Stack, D. Greene, C. Goulart, M. Mahmudi, S.D. Jacobs, M. Kallmann, M. Barlow, "Antibiotic resistance landscapes: a quantification of theory-data incompatibility for fitness landscapes", *ArXiv preprint*, March 2013.

M. Mahmudi, S. Markov, Y. Nevatia, R. Rathnam, T. Stoyanov and S. Carpin, "VirtualIUB - development of a team of autonomous agents for the Virtual Robots Competition", *SRMED*, Bremen, Germany, 2006.

# CHAPTER 1

# Introduction

Motion planning for an articulated virtual human character is a difficult problem. The difficulty of the problem arises from two main aspects: first, the articulated character has many degrees of freedom and, as such, it is considered a high dimensional problem. These class of problems are considered problematic, because they are computationally difficult to solve. Second, human motion is intrinsically difficult to reproduce realistically. This is primarily due to different ways humans move. Each human has its own unique walking style and rhythm, and its own set of gestures and gimmicks. All of these are naturally influenced by one's cultural, societal and personal surrounding. Furthermore, human motion may be restricted by social norms. Constrains of these nature are typically very difficult to encode algorithmically and thus leave a programmer or animator with the alternative of key-framing the motion manually—a very laborious and expensive undertaking—or using motion capture.

With the advent of motion capture technologies, a new venue was made possible for animating virtual characters. A subject can be directly observed and all his or her motions recorded. This significantly improves the time spent on the key-framing process, because the animator does not need to create the motion but only edit it. However, this process has its own shortcomings. The captured motion data is not easily amendable to different situations and, unless the animator is very skilled at reproducing the subject's motions, the subject has to be recaptured in order to obtain these new motions. Therefore, to avoid any possible recapturing, the director scripts all the desired motions and tries to capture them from the subjects in as few takes as possible.

Thanks to the motion capture technology, we are able to produce highly realistic representations of human motion. Some examples of such representations may be easily seen in animated movies: exemplified by the productions of Pixar, Dreamworks, Walt Disney; film productions, as one could see from the numerous realistic visual effects, industrial simulators for various purposes such as testing, training, demonstration and validation, and, last by not least, computer games where the realism of the human characters is of high importance to the gaming experience of a computer gamer.

On the other hand, generation of humanoid motion (human-like or not) has also been extensively studied in the fields of robotics. The primary goal in robotics is, in many cases, to find a trajectory for the robot such that it does not collide with obstacles and fully satisfies the constraints of a given problem. Usually the generated results may be human-like if the robot is a humanoid, but would hardly look realistic or natural. However, since the problem is already difficult to solve, realism is always never the main concern in robotics. Also, since robots are not as versatile and dexterous as humans, they are not capable of achieving realistic motions, even if the generated motions are optimal.

One difference between the fields of computer animation and robotics is that the robotics problem usually involves planning, where as the computer animation approach often does not. For example, while designing a computer game, the designer takes the motion capture database and manually builds transitions within the database such that these transitions are as smooth as possible. This structure is traditionally called a *move tree* [MJC01]. When the user plays the game, all the possible range of motions are bounded by the motions within the move tree. Not only are the set of motions fixed, but also the various transitions. For example, if the character is flying, it may not be able to start dancing, even if that might be the wish of the user. If the game is about a first person shooter, we may not command the virtual character to suddenly start skiing or dancing. The main reason for this is that computer games, as interactive as they are, do not involve any elaborate planning.

In robotics, however, the robot is capable of achieving any posture which respects equilibrium constraints, and therefore any motion, within the allowable range of joint limits. Theoretically, we may have a robot that hunts, dances, cooks, hunts some more, and then skis. There is no reason why a virtual character should not achieve all of these motions. An animator could, after all, key-frame all of the desired motions and make possible for a wide range of motions. However, we would like to solve this problem in a generic fashion, such that if the user asks for *any* motion, the system would be able to solve it. Traditionally, computer games do not allow for such variability, however, because the expectations from the user are well defined. It is not difficult, to imagine a situation where the motions that need to be generated are not known in advance and may be needed to be planned as the user controls the virtual human character.

Being able to solve motion planning at this level is an active field of research both in computer animation and in robotics. Many people share the dream of intelligent machines where robots are capable of cleaning our dishes and folding our laundry. Many others find attractive the idea of being immersed into a virtual physics classroom where a virtual Gödel is the class instructor. In such a virtual classroom, one would be able to ask the virtual Gödel a question about the Incompleteness Theorem and he would be able to reach for the chalk and write the formulas on the board, or even open a book and show which chapters talk about the Incompleteness Theorem. The questions asked by the students may not be known to the virtual Gödel, so it would have to plan its motions as the questions are being asked. Depending on the questions, the level of understanding, subsequent questions and other inputs from the students, such an ultimate virtual

Gödel would have to determine the trajectory of the chalk in order to explain the subjects that answer students' questions. This level of independence and controllability is yet not possible in computer games or real robots, and the general problem is an active field of continuing research.

Current methods have tried to solve different aspects of this problem by further advancing the field towards these goals. As mentioned earlier, the motion capture technology has significantly helped in achieving realism. On the other hand, motion planning algorithms have been developed to find solution trajectories among obstacles. A complete algorithm that analytically solves the problem of motion planning was found by John Canny [Can88]. However, the implementation of these algorithms is difficult. Moreover, the computational complexity of Canny's algorithm is exponential on the number of degrees of freedom (DOF) of the robot. A configuration space of 10-15 DOFs is already considered high dimensional. A human character can easily have more than 100 DOFs and therefore Canny's approach is not suitable for high dimensional planning such as that of virtual characters. In order to mitigate these difficulties, later algorithms introduced sampling-based approaches which greatly reduced the computational time it took to solve these problems.

Unlike Canny's approach, sampling-based algorithms do not try to analytically decompose the free section of the configuration space. Instead, they sample the free configuration space and check whether the sampled configuration is collision-free or not. By continuously sampling, these methods build a roadmap of direct line paths such that when the robot moves along this line, no collision occurs with the environment or with the robot itself (self-collisions). A solution between the start and goal configuration is then reduced to a graph search. Although sampling-based methods are randomized, they are still typically resolution-complete, meaning the probability of deciding on a problem asymptotically reaches one. Moreover, the convergence rate could roughly guide the user in terms of how many samples would be needed to solve an instance of a given problem. However, establishing the convergence rate is often difficult.

While sampling based approaches made higher dimensional problems become more tractable, they have their downsides. First of all, they are still considered slow for high dimensionality associated with human characters. Second, these algorithms do not have any added information about the nature of human motion. It is easy to generate configurations that do not violate the joint limits of a character, however, in order to encode the nature of walking, one would need a very elaborate list of constraints. For example, a human uses stepping motions in order to walk and does not slide with the feet. Such a constraint is difficult to define within the scope of the sampling-based algorithms. Lastly, it is challenging to personify gestures in various styles and rhythms. These aspects are most pressing problems with motion planning algorithms for synthesizing realistic motions.

In computer animation, these problems were handled by extensively using motion capture. While motion capture easily handles the above-mentioned problems faced by sampling-based algorithms, the recorded motions can not be easily modified to different scenarios, but can only be replayed. A great body of research work has concentrated on solving this problem. An early solution was to use move trees. As mentioned earlier, move trees were built manually by an animator and could easily control the character transitions from one behaviour to another by carefully choosing transition points between these behaviours. At this stage no planning was involved. With the introduction of motion graphs [KGP02, LCR02, AF02] at SIGGRAPH 2002 an *automatic* method to construct move trees was made possible. By using motion graphs it is possible to automatically generate transitions within the motion capture database and as such it is possible to deploy a large set of motions. The large number of motions that can be encapsulated within a motion graph, coupled with the fact that it can be constructed automatically, made motion graphs a popular technique for character animation. The algorithm requires a mechanism to extract motions out of the motion graph and this is done by satisfying the constraints of the query using a search or an optimization procedure. With methods based on motion graphs, an animator is able to automatically generate an indefinite amount of new high fidelity motions by combining motions that are directly recorded from human subjects.

## 1.1 Our Approach

Although many past methods combined motion capture and planning algorithms in an effort towards synthesizing high quality motions, preparing the data, so that they may be processed by planning algorithms has remained a time consuming task. Motion graphs helped in reducing this pre-processing step by automatically building a graph of connected motions. However, even with motion graphs, three problems remained challenging:

- The motion capture data is usually unstructured and as such, the planner can not exploit any extra information about the semantic meaning of the motion.
- Extracting and synthesizing motions from data-driven approaches involves extensive planning and does not scale well with the size of the motion capture database. Faster and smarter searches by better understanding the motion capture database are desirable.
- Independent control per degree of freedom is needed for solving problems with precise goal reaching and complex motion generation among obstacles. In complex situations, the planning can not fully rely on motion capture to generate desired solutions. Hybrid methods where motion capture examples are complemented by algorithmic motion primitive skills are needed.

This dissertation presents methods which solve these three main challenges. The feature-based motion graphs presented here provide a way to automatically introduce semantic information from the motion capture database and thus speed up and improve the planning part of the challenge. Feature-based motion graphs may be seen as a new 'language' upon which a new 'calculus' is used to solve the other two problems. The novel precomputed motion maps guided by a triangulation-based path planning mechanism made it possible, for the first time, to synthesize motions in real-time from an unstructured motion graph. And finally, a new multi-modal planning framework is shown to control and solve motions respecting configuration level constraints by using a hybrid approach, where different motion primitive skills, implementing different planners, can compete to solve a complex multi-modal problem such as that of the human motion.

Next, we present the definitions frequently used here in order to introduce the contributions of the dissertation.

## 1.2 Definitions

A virtual character is defined by a hierarchical acyclic set of joints which describe the character's skeleton. Each joint has only one parent joint, an offset and its joint limits. A geometry may be attached to a joint to give form to the virtual human character, as shown in Figure 1.1. The attached geometries define the space in the environment occupied by the character. A collision occurs if there is an intersection between the character's geometry and the obstacles in the environment or if there is an intersection between the character's own geometries.

A vector $f = (p, q_1, q_2, ..., q_n)$ represents a posture or a configuration of the character, where $p \in \mathbb{R}^3$ is the position of the root of the skeleton and $q_i \in \mathbb{SO}(3)$ is $i$th joint's local rotation in respect to the parent joint. All rotations are represented as quaternions.

In order to calculate the global joint positions of the skeleton one needs a skeleton and a posture. The skeleton remains fixed at all times and does not change. In other words, bone (or offset) lengths of the skeleton remain fixed during the entire motion. The posture is drawn from a configuration space $\mathbb{C}$ and, by applying it to a skeleton, fully defines the geometry of the skeleton and its relation in regards to the world $\mathbf{W} \in \mathbb{R}^3$ (see Figure 1.2). By applying a motion frame to a skeleton, one can create a posture of the virtual character. A few examples are shown on Figure 1.3. A posture is considered a *valid* posture if it respects the joint limits of the skeleton and does not collide with itself or the environment.

A function $M(t)$, called a motion function, is a time-parametrized function which returns a sequence of valid frames. By applying frame $f_i = M(t_i)$ at time $t_i$ to the skeleton, one can generate a valid motion for the virtual character, as

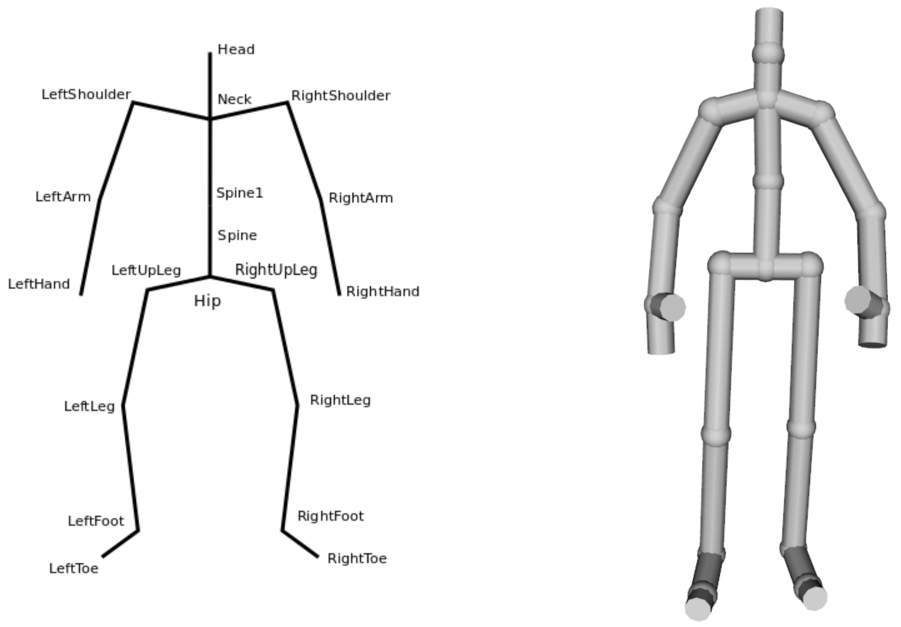

Figure 1.1: The skeleton on the left and its geometry on the right.

can be seen in Figure 1.3.

## 1.3 Quality of Motions

It is important to explain what we mean by realistic and human-like. A motion is considered *human-like* if it is similar in nature to those that one would usually observe in humans. For instance, if a character walks down the corridor, we would expect it to follow a smooth path, without turning in place, making a step to the left and then back to the right, and then continue walking. Such motions might be what the user requested, however, if no such explicit request is made, motions such as the one explained above are not considered *human-like*. Another example of non-human-like motion would be a scenario where the character opens and closes the door numerous times before it decides to finally open the door and walk through it. These examples are not desirable and if not accounted for may be generated by planners who are not aware of the nature of the problem that needs to be solved. Multi-modal planning handles these problems by carefully defining the modes the character must go through while solving the problem.

By *realistic*, we mean how realistic does the motion itself look without any regards to the nature of the motion. A realistic motion would be smooth and there would be no feet sliding or any 'robotic'-like movements. Motion capture data is considered state of the art in terms of realism. Any solution that simply replays a previously captured motion is considered fully realistic. Because motion graphs

Figure 1.2: A human character inside a virtual environment where everyday tasks are executed.

create artificial transitions between parts of the motion capture database which are most similar, the realism begins to slowly deteriorate. This small degradation is controlled through the threshold error associated with the created transitions during the motion graph construction phase. Since these transitions occur at low error points and are determined by the animator, they are usually considered to be of high fidelity and therefore the results obtained by these methods are also considered realistic.

## 1.4 Problem Statement

Given a world **W**, a skeleton and a high level task, such as move a book on the shelf from position $p_0$ to a new position $p_1$, efficiently generate valid motion function $M(t)$ that accomplishes the task and looks *realistic and human-like.*

In order to solve the problem stated here, and the challenges mentioned earlier in this chapter, we present a multi-modal data-driven motion planner and synthesizer that utilizes featured motion graphs, precomputed motion maps and a motion primitive skill planner that generates high quality motions efficiently with precise target reaching capabilities by handling complex motion generation problem among obstacles.

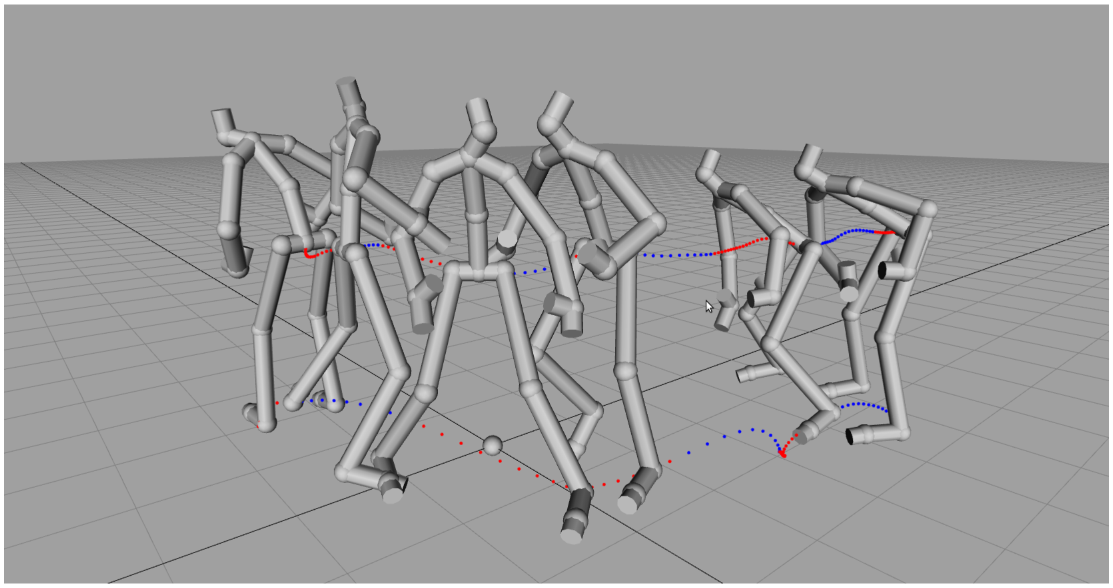

Figure 1.3: A sequence of basketball postures applied to a skeleton.

## 1.5 Dissertation Overview

This dissertation addresses the given problem by proposing three main approaches: feature-based motion graphs, precomputed motion maps and multi-modal data-driven planner using motion primitive skills. The following sections discuss these approaches in detail.

### 1.5.1 Feature-Based Motion Graphs

Locomotion is the most essential part of human motion. It provides a means of natural transportation for humans and a basis for the human to perform other motions in different locations. Therefore, locomotion is the most important motion that we deal with in this dissertation. In regards to the multi-modal planner, the locomotion of the virtual character is generated by the locomotion skill which is a data-driven motion primitive skill.

The locomotion skill is based on the same techniques presented in the Motion Graphs by Kovar, Lee and Arikan and their respective colleagues [KGP02, LCR02, AF02]. However, the locomotion skill extends these techniques and addresses some of its disadvantages. As mentioned earlier, data-driven planners, including methods based on motion graphs, suffer from the unstructured nature of the motion they deploy and as a consequence they do not have any semantical meaning associated with them. Therefore it is difficult to differentiate these motions and know whether they contain walking, standing, punching or other motions. We present a method in Chapter 3 that uses the idea of motion segmentation to segment the motion at certain geometric features. This segmentation has many advantages as it helps in solving some major problems with motion graphs: reducing the size of the motion graphs, improving the graph construction phase by two orders of magnitude, automatically correcting feet sliding without any degradation of the quality of the results. Geometrical features were designed to extract logically related motions scattered within some motion capture dataset [MRC05]. These features have proven to be quite powerful, because they incorporate spatial relationships between joints and try to find logically similar motion segments instead of numerically similar segments.

With traditional motion graphs usually the transitions are found first and then the motion is segmented. With feature-based motion graphs the construction process is done in reverse. Instead of segmenting the motions where the local minima occur in the error image, we use geometric features to first segment the motion into semantically similar clips and then sample the error image—instead of fully computing it—for suitable transitions. Not only do we avoid a major bottleneck, but by a using feature-based motion graph, we know that our planner contains clips which have a certain structure. This becomes essential for creating motion maps and also when using a multi-modal planning framework. For example, because we know the general nature of the segmented walking clips, we can potentially allow many motion primitives to be activated at the same mode. With traditional motion graphs this would not be possible as the length and the nature of motion clips would not be known in advance before beginning the planning step.

In addition to these advantages, feature-based motion graphs also provides two novel techniques: *triangulation-based* search procedure and *inverse branch kinematics*. Extracting motions from a motion graph usually involves a discrete search where the graph is unrolled into the environment. The graph itself does not include any positional information. That is, we may have a motion graph with only two nodes: a left step and right step, and with these two nodes alone we may generate straight walking motions indefinitely. Unrolling takes the motion graph, augments it with positional information and creates a search tree, where each branch represents a possible solution whose motion takes the character from the starting position to the end of the motion as defined by the last node (a leaf) of the said search tree branch. The unrolling, which controls how the search tree progresses, is directly dependent on the topology of the motions within the motion graph. For example, if A*-like search method is used, the search tree will try to progress towards the goal; however, if the motion graph only contains left turning motions, the search tree will be following a round of left turns until the goal is reached, if at all.

By using the triangulation-based search procedure [Kal10], we can improve the search mechanism, because we do not blindly unroll towards the goal without any consideration for the environment. Instead, we first triangulate the free-empty space, find an 2D viable channel between the start and goal and then search within this channel. This has two major implications: first, the channel is a global planner and thus guides the unrolling process and avoids local minima where search might get stuckl and second, because the channel has clearance, the search need not run the collision checking module. This speeds up the search significantly, as the collision checking is the most computationally consuming step.

Triangulating a 3D environment is already a complicated tasks. In our method we project all the vertices of the objects that consist the environment to the floor and compute their convex hull. Once this is achieved, we insert these convex hulls as constrained polygons in a 2D Delaunay triangulation. Some obstacles are labeled with special information in order to maintain the desired triangulation. Objects which are higher then the some maximum height (usually the character's high) do not get projected and therefore do not introduce additional constraints in the triangulation. This is the case with the wall were the door is attached. Vertices which are right above the door do not get projected into the floor such that the triangulation allows the character to pass through the door, otherwise, the planner would not be able to distinguish between the door and a plain wall. For more elaborate environments, more dedicated methods may be used for this purpose. Navigational graphs for multi-layered and uneven terrains exists [PLT05]. A triangulated extensions with clearance guarantees of this work may be used for our locomotion planner for more complicated environments involving multiple layers. Moreover, clearance guarantees need not always remain constant. Some narrow passages may have variable clearance and this may be important to the planner during the search process. Usually, the small the channel is the faster are the searchers. However, the added information may help in using specific types of motions or improve the search by avoiding running out of leaves in narrow passages. One such 2D path planning methods is developed by Gerarts [Ger10]. Our planner may also use this type of triangulation for the purpose of guiding our feature-based motion graph.

The other disadvantage of traditional motion graphs is that each node progresses in a discrete fashion—a clip at a time. Since the transitions are fixed and not subject to change, we cannot bend the search tree branch to reach the goal precisely, which is an important aspect of multi-modal planning. Hence, we propose an inverse branch kinematics mechanism which deforms the search branch tree and thus precisely reaches the goal.

In the definition given in Equation 3.2, the rotation, which defines the transformation for the automatically generated transitions, remains fixed, even if there could be room to further rotate the transition and still remain below the threshold. We suggest a method to exploit this fact, by first finding the optimal transformation, and then further rotating the transition (clockwise and counter-clockwise) and reevaluating the transition up until the error threshold is reached. In this manner, for each transition we have a range of acceptable limits, which later could be used by the search mechanism to bend the search tree within the allowable range. The branch can be seen as a kinematic chain with joint limits such that, when it is within a close distance to the goal, it could stop the unrolling process and run an optimization step (cyclic coordinate descent) until the goal is precisely reached. If the optimization does not bring the search branch closer to the goal, we stop the deformation and go back to the searching procedure. This makes it possible for the search process to build a continuous search tree which either reaches the goal precisely or speeds up the search by bending a search branch which might have missed the goal.

The above-mentioned novel methods greatly improve the motion generation process for data-driven methods. At the same time, these methods allow for a better data-structure for real-time motion generation from unstructured motion capture (motion maps) and independent joint-level control by using a multi-modal planning framework.

### 1.5.2 Precomputed Motion Maps

The triangulation-based search procedure helps to significantly speed up the searching and planning. However the problem still suffers from local minima within the channel. If the goal is behind a wall and the channel had to go around the wall, the node expansion would still occur on the nodes, which are close to the goal, thereby slowing down the expansion within the channel. What is needed is a fast path following mechanism where the search would prioritize nodes which are close to the path and not close to the goal. This is difficult, however, because we would have to switch to uniform cost search to achieve this. A better alternative is to use pre-computation. This technique was first used to speed up the manually built behaviour finite state machine of human motions [LK05, LK06]. There were two problems, however. First, for the pre-computation to cover good parts of the environment, the finite state machine of behaviours had to be crafted by hand, because there is no guarantee that a motion graph built automatically from unstructured data will populate the environment evenly and provide a good precomputed search tree. The second problem is that only one precomputed tree is built and used for planning, the downside of which is we have to ensure that leaves of the search tree connect with the root node. This could not be achieved unless the graph is built manually.

Precomputed motion maps solve all these problems and, furthermore, speed up the search, such that the planner can generate motions in real-time within obstacles and, most importantly, from an automatically-built motion graph from unstructured motion capture data.

The pre-computation procedure involves expanding a node of a graph up to a certain depth and storing the partial branches of the search tree into a hashmap called motion map, such that they can be later queried by the planner. The pre-computation essentially performs the same procedures involved within a graph search, however, instead of expanding single nodes, it expands entire branches. This makes it possible to search much faster at the expense of optimality and memory usage. The optimality is only slightly impacted, because our planner uses a global 2D path planner, which then later guides the graph search and its associated precomputed motion maps.

The other important contribution is to build motion maps from unstructured data, because our feature-based motion graphs introduce structure. Because of the even topology of feature-based motion graphs, the motion maps are exploring the environment much more evenly and, as a result, they can be used with any automatically built motion graphs. Moreover, unlike previous works, we are building motion maps for all the nodes of the motion graph and, as such, do not need to carefully prepare the motion capture—further making motion maps the only suitable method for automatically built motion graphs.

The planner begins by first finding a 2D path in the triangulated environment. Then an initial motion map is superimposed on the environment at the beginning of the 2D path. The procedure starts by sampling the 2D path and querying partial solutions from the motion map for suitable partial candidates. All valid candidates are evaluated using the arc-area metric as in the uniform-cost case, and the best candidate is chosen to define the partial solution. Then the process is repeated with the motion maps of the last solution. Occasionally, no candidate is available to follow the path closely; in that case the search procedure rolls back to the previous iteration and chooses the second best candidate and repeats the search down this new candidate. This continues until the goal is reached or all the alternatives are exhausted.

Another advantage of our method is that it can play and search at the same time. Because the duration of the motion is much smaller in comparison to the time it takes to search for a motion, we begin playing the partial solutions while the search deliberates and finds the residual solutions. This makes it possible for a real-time interactive planner in an environment with many examples. Thanks to feature-based motion graphs and motion maps, we were able to solve the problem of real time motion synthesis from an automatically-built motion graph for the first time, a problem which was considered difficult to solve [GSK03].

### 1.5.3 Multi-Modal Skill Planner

Although feature-based motion graphs and precomputed maps allow for efficient reuse of data-driven examples, they do not easily allow for more controllability of the human character. As mentioned earlier, we could transform the motion by translating it on the floor and rotating it about the Y axis, however we could not lift the arm of human character, if that was not already in the motion. As such, these previous two methods are good at allowing the use of similar motions, but do not scale well for complex tasks. More complex tasks require planning at independent joint-level control and, therefore, can not be easily handled by these methods.

We could solve this problem by capturing more motions, however, as the motion capture database increases so does the motion graph. Since the search is exponential on the number of the motion graph nodes, it is not efficient to capture many variations of complex tasks and hope that some of them might be picked by our planner. Moreover, even if we capture more examples, subtle changes in the environment could easily make the problem difficult to solve.

Instead of brute forcing the problem by adding more data, a better approach proposed in this dissertation is to have primitive motion skills which solve small subtasks that are dedicated solely to the task they solve. Our multi-modal planner takes this approach by searching among parametrized motion primitive skills within separate modes. This allows many instances to compete for the best solution, while guaranteeing that they solve the partial tasks within their respective modes. Since different motion skills may be implemented using different local planners, the multi-modal search allows skills to compete for the best solution independent of their implementation.

Because locomotion is the most important part of human motion, the proposed multi-modal planner uses a data-driven motion primitive skill for locomotion. There are two main reasons for this: motion graphs are good and efficient (with precomputation) at generating locomotion, and, most importantly, locomotion defines most of the human motion and therefore it is important that its solutions are of high quality.

The locomotion skill is implemented using feature-based motion graphs and its goal is to find a motion that displaces the character in the environment and prepares it to achieve other desired motions. The user begins by specifying a start position and the multi-modal planner, then expands the search by picking new nodes from the locomotion motion primitive, while at the same, time expanding other motion primitive skills that might be activated at given modes. Depending on the skill which gets activated, many skills may define the final motion of the human character. For example, the locomotion skill might define the entire posture of the character as captured by one node of the graph, but then the reaching motion primitive may change the arm motion instead of the locomotion skill, in order to reach a specified target location for the hand. In this way, we can see many motion skills contributing to the final result simultaneously. Because the multi-modal planner activates motion primitive skills by their parameter list, there are many motion instances which try to reach the final goal and produce the desired motion. Moreover, since skills may implement different local planners, we might have, for example, a reaching skill implemented using a sampling based approach but also a data-driven approach by blending examples of reaching actions. Both these implementations have their advantages and disadvantages and depending on the situation, the multi-modal planner may prefer one over the other.

Another advantage of the multi-modal approach is that human motion is, by itself, multi-modal. For example, humans do not plan for locomotion while they are seated in a chair. In that situation, the only concern would be to plan the motion of the arms and the torso. Alternatively, when walking down the corridor, only the walking trajectory needs to be maintained. Also, even when performing seemingly related tasks such as opening a door our proposed approach separates the tasks into modes where the character walks, tries to reach the door, opens the door, releases the door, and walks through it. These decompositions allow the planner to plan for motions at fewer dimensions than that of the entire character, which is done by associating a mode with a part of the human character or a special state of the character, depending on the tasks that needs to be solved.

This dissertation proposes that human-like motion planning for virtual human characters can be well addressed by a multi-modal data-driven framework. This approach employs a set of parametrized motion primitive skills that successfully plan and synthesize human motion for complex tasks in environments with many obstacles.

# CHAPTER 2

# Literature Review

In this chapter we discuss prior works which are related to the methods proposed in this dissertation. We begin by reviewing the approaches used in robotics and then we discuss approaches used in computer graphics. Although these two fields have attempted to solve the general problem, the algorithms proposed within these fields have solved different aspects of the motion planning problem. The algorithms from robotics often search on continuous configurations and try to solve generalized instances, whereas the algorithms from computer graphics are concerned with quality of the generated solutions and are particularly designed for articulated characters. The latter approaches are, therefore, usually data-driven due to the high fidelity of the results.

In more recent years, there has been an effort to combine different aspects of these solutions to efficiently solve more generalized problems for human character and obtain high quality results. Hence, data-driven algorithms have moved from discrete search spaces to more parametrized spaces whilst continuous motion planning has evolved into multi-modal spaces in order to better address the problems related to humanoid motion planning.

We review prior work in this order:

- Sampling-Based Planning
- Physics-Based Planning
- Data-Driven Planning

## 2.1 Sampling-Based Planning

The problem of motion planning was proved to be an instance of the class of PSPACE-compete problems [Rei79]. An analytical solution for this problem was found by Canny [Can88] by cell decomposition of the free configuration space. The computational complexity of this analytical solution is exponential in the number of degrees of freedom (DOF) of the robot and, therefore, it is considered a difficult problem, especially in high dimensional instances. To mitigate this computational requirement, approximate methods where introduced which could find solutions for environments with simple obstacles [Lat90].

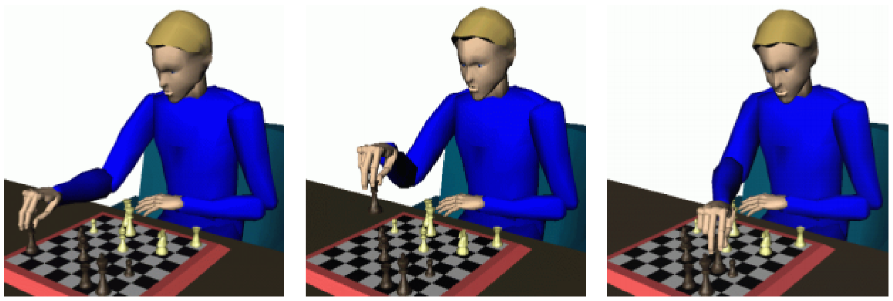

Figure 2.1: 7 DOF arm motion generated by an RRT planner by Kuffner et al. [KL00c]. Copyright 2000 IEEE.

One major breakthrough was achieved by the use of sampling-based methods [LaV06]. One such major algorithm was developed by Kavraki et al. [KSL96]. This method did not need to know the exact definition of the occupied configuration space; it only required a function which could evaluate whether a sampled configuration is valid or not. As the configuration space is sampled, a roadmap of connected samples is built. Two configurations are connected if the interpolated intermediate samples also lie within a collision free path. For every query a graph search is run on the Probabilistic Road Map (PRM) to find a collision-free path between the initial and goal configurations.

Later methods were developed to improve PRMs. Lazy PRM [BK00] and Fuzzy PRM [NK00] used a similar sampling approach by using a different searching method which delayed collision checking as much as possible. Because collision checking is a time-consuming procedure, these methods improved the time complexity of the PRM method.

Another important sampling-based method was developed by LaValle [Lav98] using a new data-structure called Rapidly Exploring Random Trees (RRT). This method was similar to that of the PRM method, however instead of building a graph, RRTs expanded a tree and connected configuration samples which usually were in close vicinity to the tree. A more elaborate algorithm using RRT trees called RRT-Connect was introduced by Kuffner and LaValle [KL00c] (see Figure 2.1 for an example). The method maintained two RRT trees: one at the start configuration and one at the goal configuration, which were expanded until the tree could be successfully connected. A PRM-like approach, which employed bidirectional trees similar to RRT with lazy collision checking, was devised by Sanchez et al. [SL01].

RRT became a popular method for robots that have constraints to their motion model, such as non-holonomic vehicles because these constraints were easier to encode within RRTs. Other problems which accounted for the dynamics of the robots, such as velocity and acceleration also widely used RRT based methods to solve instances of kinodynamic problems. [LK01, PKV07]. More recently, learning methods were developed by Li and Kostas [LB11] to learn better metrics for dynamics systems. Sampling based methods were further improved by employing some form of computer parallelism. Carpin et al. [CP02] introduced a parallelized version of RRT, whereas Plaku [PBC05] assigned an RRT tree to each computer node for the purpose of solving high dimensional problems. Other improvements were made to account for dynamic obstacles, such as the methods by Kallmann et al. [KM04], Berg et al [BFK06] and Leven et al. [LH02]. More recently, Karaman et al. developed RRT$^*$ which introduced optimality guarantees to the generated solutions and showed that their algorithm is provably asymptotically optimal while maintaining a computational complexity which is within a constant factor of that of the RRT counterpart.

The third group of sampling-based algorithms were developed by Hsu et al. called Expansive Configuration Spaces (ECS) [HLM99]. These works were the first papers to introduce the notion of expansiveness and give guarantees on the coverage of the configuration space. Similar works used entropy-based sampling based methods [BB05] to improve the sampling heuristic. The work by Geraerts [GO07] evaluated different sampling techniques for different environments.

Although these methods are capable of solving general robot motion planning algorithms, they are not suited for humanoid motion planning for the following reasons: humanoid motion planning is a high dimensional problem, human motion requires an elaborate list of constraints to be specified in order to obtain human-like motions and finally the results are not realistic and look robotic.

More dedicated methods were employed to suit the problem of humanoid motion planning. Due to the high dimensionality of the problem, Kuffner, Chestnutt and their colleagues [KL00a, KNK03, CK04] used a discrete stepping planner to find motions for a biped humanoid. Various stepping motions where manually crafted and then combined to form a sequence of walking motions. These method are most similar to the methods used in computer graphics but are built with humanoid robots in mind. Kanoun et al. [KLY11] cast the foot placement problem as an optimization and used an IK-like approach to solve for a correct sequence of placements.

Foot step planning often gets stuck in local minima if the goal is strategically located behind obstacles. In order to better understand the free space of an environment and avoid such local minima, some form of decomposition was used. One such example is a triangulation-based method developed by Kallmann [Kal10]. Other methods include a Voronoi-based decomposition method by Geraerts [Ger10] and a prismatic-spacial subdivision-based method by Lamarache [Lam09]. Additionally, these methods return paths with guaranteed clearance to the obstacles which is beneficial to the foot step planning stage of the humanoid robot. Other works have improved the searching mechanism to speed up the search by using intermediate subgoals [SB02] and other works have evaluated human subjects to investigate the trajectories of human walks for a given start and goal position and orientation [ALH08]. This paper showed that humans

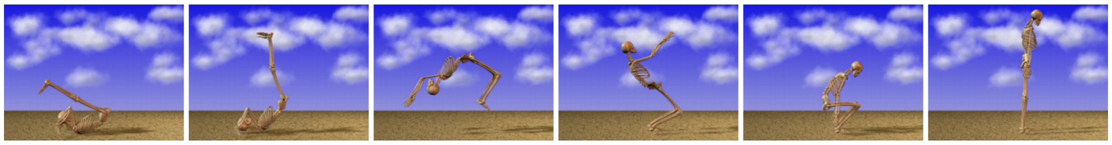

Figure 2.2: Physics based controller for a kip move by Faloutsos et al. [FPT01]. Copyright 2001 ACM.

minimize the time-derivative of the curvature the locomotion, and that they maybe be best modeled by a Cornu spiral whose curvature grows with the distance from the origin.

Our locomotion motion primitive skill deploys a triangulation-based search mechanism to obtain a global path with clearance between a start and goal position. In addition, our method is data-driven and does not require manually crafting stepping motions. Moreover, since our method is data-driven, our results are high quality and look human-like.

Upper body action planning for humanoid planning, such as grasping and manipulating, has also been extensively researched. Recent methods can automatically deduce the shape of the manipulated object and efficiently grasp, regrasp, and perform other complex manipulation tasks [BDN07, BC12] and interact with the environment. These interactions could also be taken into consideration during planning, such as, in the work of Stilmann et al. [SK04] where the humanoid robot moves obstacles (couches and chairs) in order to reach the desired goal.

## 2.2 Physics-Based Planning

Another approach to simulate the movement of objects is based on physics. Given a set of physics laws for that define the forces that act on a body and an initial configuration, the entire motion of a given body could be calculated. The work by Witkin and Kaas [WK88] used space-time optimization based on physical particle-motion definition. They specified a few intermediate frames as constraints to the optimization and generated the entire motion of the now-famous Luxo lamp. There has been a lot of work devoted to physics-based methods for generating human character motions [HWB95, FPT01, LP02, Liu08, WJM06]. All of these methods require an elaborate formulation of the human character, including mass distribution, torques and joint specifications. Moreover, each type of motion is determined by a physics controller, which requires an elaborate list of constraints to fully specify and also is time-consuming to animate. The results of these algorithms are not fully realistic and have some awkward motions. Furthermore, it is difficult to specify personal traits and gestures for human characters. For an elaborate review on physics-based character animation, please refer to Liu [Liu05]. See Figure 2.2 for an example of motion generated using a physics controller.

## 2.3 Data-Driven Planning

Traditionally, human motion was depicted by an animator who would draw the motion of a human character in a sequence of 2D images. With the advent of computer graphics, algorithms were developed to achieve the same animation techniques for 3D characters [Las87]. These animations were laboriously hand-animated using the method of key-framing. The animator would draw important keyframes and the method would generate in-between postures by interpolating these key frames. A great body of work was developed to algorithmically generate human motion. These included methods from pre-encoded procedural motions to methods which involved signal processing. For a review on key-framing, please refer to [WBE10].

### 2.3.1 Blending

Synthesis of high quality whole-body human-like motions was achieved with the help of motion capture. Due to its high fidelity results, methods based on motion capture became very popular. Initially, the recorded examples would only be used to replay the motions, however later, various techniques were used to modify the motion. An early technique to change the motion capture data was based on interpolation. Given a set of data, any motion could be generated by interpolating these input motions [WH97, RBC98]. By using this technique, an animator could increase the variety of the motion capture database. A locomotion blender was developed by Park et al. [PSS02] based on Rose's *Verbs and Adverbs* [RBC98]. The downside of this method was that the animator would have to carefully make sure that the interpolated motions are similar in space-time. To aid the animator, Witkin and Popović developed motion warping [WP95] and Kovar et al. introduced registration curves [KG03]. Later, work by Kovar et al. also automatically queried similar motions from a database and blended them in order to generate variety of motions such as kicking and punching, etc. [KG04]. Extracting and finding similar motions from a motion capture database was needed for correct blending of examples. Barbic et al. [BSP04] used a PCA-based method to segment motion capture data into distinct behaviours. Muller et al. [MRC05, MR06] devised spatio-temporal geometric features to segment similar motions without being effected by numerical differences.

While interpolation and blending created in-between motions by using multiple motion examples, other algorithms were developed to modify a single motion capture stream. Pullen et al. [PB02] used a texturing mechanism to fit a motion capture motion to a list of key frames. Li et al. [LWS02] learned statistical *textons* from motion capture and then combined these to modify the examples. Usually it is necessary to create transitions between different parts of the motions. The papers by Wang et al. [WB03, WB04] ran empirical experiments to evaluate the realism of the generated transitions. Ikemoto et al. [IAF07] used a

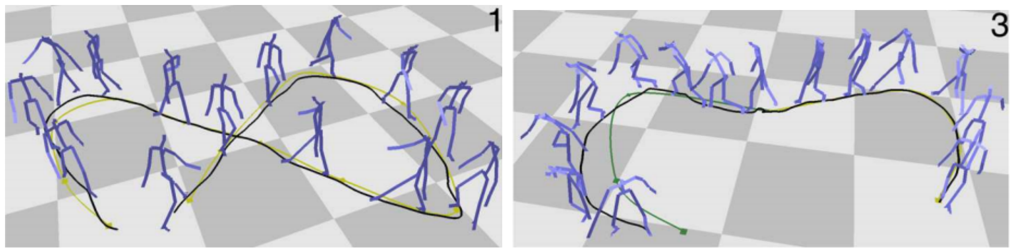

Figure 2.3: Motions generated from a motion graph [KGP02]. Copyright 2002 ACM.

multi-way cached blending technique to create more realistic transitions. Interpolation and blending often creates artifacts such as feet sliding. Because these artifacts can be easily detected, a few methods were developed to automatically correct feet sliding artifacts while blending motions. An IK-based approach which also involved skeleton bone modifications was developed by Kovar et al. [KSG02]. Other motion modifications involve stylistic changes to the human character. The work by Neff et al. [NK09] looked at posture correlations to extract parameters that control stylistic aspects of the motion. Various other algorithms were made available for constructing long pieces of motion by using motion patches [LCL06] as building blocks for more complicated motion generation. Similarly, Krüger et al. [KTW10] built a kd-tree of similar poses and proposed fast and local pose queries for motion synthesis. Another method to modify motion capture data involved splicing the upper and lower body parts and combining them together to achieve different combined motions [HKG06, LKN09, BE11]. These methods propose algorithms and coordination models which analyze what lower and upper body motions may be combined together while still retaining the human motion realism. Gleicher used a retargetting method to fit the motion of one skeleton to another with similar bone lengths [GSK03].

### 2.3.2 Motion Graphs

Although modifying motion capture increased the range of the motions, more automatic motions were needed to handle large amounts of motion capture data. A major step toward this goal was made by the *motion graph* data-structure [KGP02, LCR02, AF02], which was similar to that of move trees [MJC01] but built automatically. These methods capture frame similarities in large motion capture datasets and build automatic transitions between similar frames. By creating these transitions, a graph representing the connectivity of the motion capture database could be built and, hence, any graph walk on this structure would result in a different motion but still retain the quality of the initial motions. See some example in Figure 2.3.

The graph is constructed by first detecting frames that are very similar to each other by doing a pairwise checking of all the frames in the motion capture dataset. This creates a 2-D error image representing the cost of transitioning from one frame to another. All the transitions which are situated at local minima in the error image and pass a certain user-defined threshold are joined together by an artificial smooth transition, which interpolates between these two different parts of the motion dataset. The motion generation is then casted as an optimization problem, which tries to find the most suitable graph walk that satisfies user constraints.

Arikan et al. [AF02], had a very similar idea of constructing a motion graph but instead of having one single motion graph at a fine level, they maintain a hierarchy of graphs that represent the motion graph at different granularity in terms of motion quality. Their search algorithm is an anytime algorithm that starts the search with a rough solution that satisfies only hard constrains, and as time permits, the search mechanism starts searching for better motions that satisfy the soft constrains as fully as possible. A later algorithm developed by Arikan et al. [AFO03] uses motion clips such as: jump, run, etc. and generates combinations of these clips by using dynamic programming to satisfy the positional constraints set by the animator. Semi-automatic motion graphs were built for interactive control by finding hub frames that contain many transition connections. [GSK03]. Manually built motion graphs that represent distinct behaviour were also used by Lau et al. [LK05]. These behaviour finite-state machines were used to generate motions around obstacles by first running a global optimal path around the obstacles and then running an A*-like search on the finite state machine to search for an appropriate motion.

As motion graphs became more popular, methods that evaluated the connectivity, reachability and interactivity of motion graphs were developed [RP07]. These algorithms *unrolled* the motion graphs into an environment with obstacles and evaluated motion graphs by measuring how well the graphs reached different parts of the environment.

To increase the range of motions produced by motion graphs, blending and data-driven methods were combined. Fat graphs were used [SO06] which augmented a single motion graph node with many parametrized motion clips. Heck et al. [HG07] used parametric motion graphs and developed a method to transition from different sets of parametrized motions within a motion graph. Safanova et al. [SH08] interpolated $k$ motion paths within a motion graph in order to produce a wider range of motions.

As methods were developed to increase the expressiveness of the motion graphs, the search procedure got negatively effected by the increase in the size of the graph. Algorithms were developed to construct more compact motion graphs [BCP08, ZS08, LS09]. These methods automatically reduce the size of the graph, increase its connectivity and reduce the size of the motion capture database by finding methods to compress the data. Motion graphs were also combined with continuous optimization which improves the transitions of the graph and still retain its expressiveness [RZS10]. Motion graphs were also used for crowd simulation [SKG05], music motion generation [FXG12] and motion modeling [KS05]. They involve understanding the underling human motion and build a model which later extracts appropriate motions from a motion graph.

More recently, feature-based motion graphs were developed by Mahmudi et al. [MK11, MK13] which used geometric features to create transitions among the motion graph and introduce semantical information within the motion graph with an non-degrading deformation model. Similarly, Min et al. [MC12] use a deformable motion graph that is capable of generating infinite style variations of the actions inside a motion graph.

### 2.3.3 Precomputation

Extracting a desired motion from a motion graph requires some form of searching and planning. This procedure is often very time consuming and usually exponential in the number of nodes in the motion graph. Because the same motion graph is expanded for each query, pre-computation was used to expand many nodes at the same time. This idea was first used to precompute policies for two characters who are trying to kick each other [LL04]. Later, a precomputed tree was used by Lau et al. [LK06] to precompute their finite state machine of behaviours and follow a globally optimum path in the environment. Later, this work was expanded to find randomized techniques to build a precomputed tree with more even coverage density [LK10]. Usually one search tree will be precomputed for the entire motion graph. This required that all the leaves of tree connect to the root node. Srinivasan et al. [SMM05] relaxed this requirement by precomputing maps for all the nodes of the graph, but employed a simple path following algorithm with reactive obstacle avoidance. Mahmudi et al. [MK12] precomputed motion maps for all the nodes in a feature-based motion graph and developed a search algorithm that was capable of generating motions interactively among a complex environment with many obstacles.

Reinforcement learning could be used to learn control policies for human motion generation. This method usually involved a set of manually-crafted motion clips. A learning objective is set and learned through reinforcement learning. This way a character can be controlled by continuously specifying the walking direction, speed, obstacle positions, etc. [TLP07]. Similarly, reinforcement learning is used over a set of parametrized controllers to plan the motion of human character. Lee et al. [LWB10] construct *motion fields* which are learnt through reinforcement learning to set the current motion direction of the character. A later version by Lee et al. [LP10] uses inverse reinforcement learning to learn various behaviour styles.

Similarly, active learning was also used by Cooper et al. [CHP07] to learn controllers for tasks such as catching a ball. Wu et al. [WP10] used a biped lo-

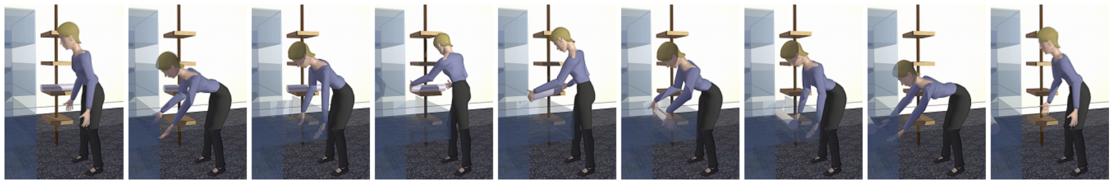

Figure 2.4: Manipulation motion among obstacles by a randomized planner [YKH04]. Copyright 2004 ACM.

comotion controller to adapt to uneven terrains. Their approach uses a footstep end-effector planner and a per-timestep generalized-force solver. The parameters of the planner for different tasks are solved by using an offline optimization procedure. A method proposed by Zong et al. [ZLX12] uses Q-learning and plans manipulation tasks for a human character by using domain-specific knowledge that governs characters behaviour.

### 2.3.4 Planning

One of the first papers to use motion planning for animation was introduced by Koga et al. [KKK94]. Their method used a randomized path planning algorithm to synthesize the manipulation of the arms of a character that could pick up his eyeglasses and put them on. Similarly, Kuffner and LaValle [KL00b] used a combination of RRT and IK to plan the arm of the human character (see Figure 2.1). The character could play chess and remove tools from a box while avoiding the obstacles in the scene. Yamane et al. [YKH04] uses an RRT planner to find the path for a manipulation task, and then uses IK to fill the posture of a character such that it can follow these trajectories (see Figure 2.4). The method uses the null space of the IK solution in order to find more natural-looking poses. It also has a coordination model, such as velocity profiling and gaze modeling to make the model look more realistic. A method developed by Kallmann et al. [KAA03] uses a PRM-based planner for manipulation and grasping tasks taking into account the whole-body of the character which includes automatic column control and leg flexion. IK methods used in these methods are based on the algorithms proposed by Tolani et al. [TGB00] and Baerlocher et al. [BB04].

On the other hand, Choi et al. [CLS02] developed a locomotion motion generator based on motion capture and a PRM planner. A PRM planner would generate a list of footsteps, and the motion synthesizer would try to generate motions that could best follow the footsteps. Sung et al. [SKG05] used a similar approach, but instead of planning on foot steps, they used a PRM planner to first find a 2D path in the environment and then used a bidirectional motion graph search to find a motion which could follow this path. Esteves et al. [EAP06] proposed a motion planner for a human character that could handle open and closed kinematic chains without colliding with the environment and other collaborators. The method used a triangulation-based data-driven locomotion planner for the lower part of the character, and a PRM-based planner to handle heavy objects and collaborate with other characters on these tasks. A method developed by Pan et al. [PZL10] decomposes the human character into a hierarchy of joints sets and uses RRT to plan various parts of the motion planning problem, then combines them together to achieve a whole body motion for a character that could avoid cluttered obstacles and perform complex tasks. An algorithm developed by Shapiro [SKF07] modifies a sequence of motion capture by utilizing an RRT planner that could modify the offending part of the motion, such that the obstacles are avoided. The planner runs on the arms of the character, and it is time-parametrized, allowing the character to plan for a motion while moving.

A method which combines locomotion and manipulation motions was developed by Huang et al. [HMK11]. The character would walk and find a suitable position to achieve a manipulation task, such as pointing, pouring water, etc. The manipulation task was achieved by blending many motion examples, whose blending weights were computed by using an RRT planner on the blend space. This allowed the manipulation task to avoid obstacles. Basten et al. [BEG11] reviews different combinations of path planning algorithms with use motion graphs for motion generation with a strong emphasis on the naturalness of the generated results. A later method uses path abstraction to modify the pelvis of the human character to increase the naturalness of the motions [BE09]. More effort was put into analyzing human subjects and identifying parameters that quantify the naturalness of the human motion. Egges et al. looked at human subjects to build a coordination motion for human walking and manipulation tasks [Egg08a] and other motions, such as door opening tasks [Egg08b]. These works help in constraining the search procedure of motion graphs in order to help generate more natural motions and avoid motions which would not be performed by human subjects.

### 2.3.5 Deformation

Planner which combine locomotion with manipulation tasks require precise placement for the character to achieve the manipulation without colliding with obstacles. More recently, this has lead to the development of deformation models for the synthesized motions. Mahmudi et al. [MK13] proposed a continuous motion deformation based on CCD method [WC91], which deformed the motions at the transition points without exceeding the transition error threshold. Kim et al. [KHK09] and Choi et al. [CKH11] used a Laplacian deformation model, which deformed the motion and still preserved the contacts and other constraints within the environment. Similarly, Min et al. [MCC09] used a motion deformation model based on a maximum a posteriori (MAP) framework.

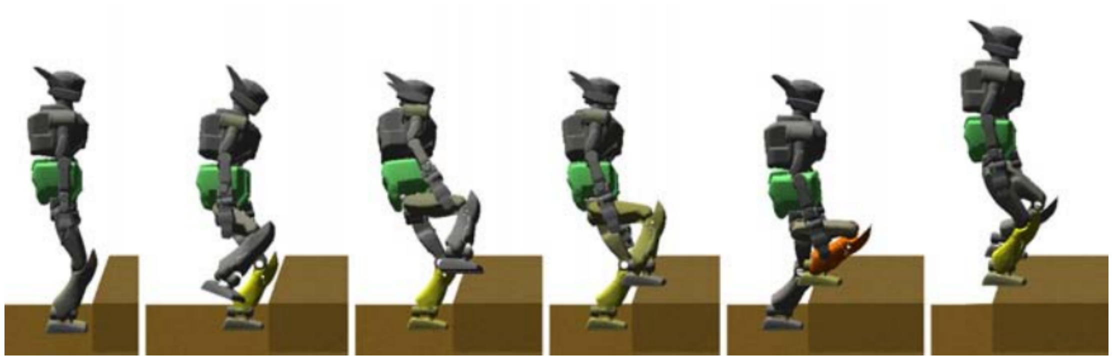
Figure 2.5: A stepping motion using a sampling-based multi-modal planner [HBH06]. Copyright 2006 Springer.

## 2.4 Multi-Modal Planning

Because of the complexity of human motion, a significant body of work has been dedicated to understanding how humans move. Studies from neuroscience and human biology have indicated that humans use what is called motion primitives to perform their movements. These motion primitives may be seen as atomic motion blocks and are believed to be "hardwired" in the human brain. Through repeated random trials through one's life, these motion primitives are calibrated and adapted by extensive learning via repeated experience [PKM06, ICD05, TS00].

A new approach in robotics has increasingly used a multi-modal approach to address the problem of motion planning for high dimensional instances. A mode is generally associated with a subset of the configuration space, although in this dissertation, our multi-modal framework uses a mode to solve a subtask of a larger problem. From traditional motion planning point of view, by searching within different modes, the effective dimensionality of the configuration space is reduced. Although in robotics, the same planner is used to solve the partial trajectory within a mode, in our framework more specialized sets of motion primitives may be employed while at the same time competing with each other to be a part of the final solution. The multi-modal is an efficient approach in solving complex tasks due to the observation that subset of configuration spaces are often independent of each other in terms of the final solution. This is also observed in human motion. For example, when we eat, we do not use our legs as we would do while walking; hence we wouldn't need to solve a problem using the entire configuration space, only a part of it. At some times, we might only be interested in walking motions and not in manipulation tasks, and at other times in both, depending on the task. Multi-modal planning helps in decoupling the problem into subtasks and solving the problem with efficient local planners.

Hauser et al. [HBH06, HNG07] first used multi-modal planning for humanoid robot motion planning. Later, work used to include manipulation tasks [HN11].

An example is shown in Figure 2.5. Kallmann et al. used a list of parametrized skills to control a simple biped robot. [KBM04]. Later a more elaborate algorithm was developed to plan the motion of a humanoid robot by a multi-modal skill planner [KHB10]. In computer animation, methods were proposed to use parametrized controllers, which may be seen as a multi-modal approach of searching motions for a human character [LLK11, BSL12].

In this dissertation, we present a new approach which combines multi-modal planning with data-driven methods, based on motion capture and continuous randomized planners in accordance with coordination models, to maintain the realism of the synthesized motions.

# CHAPTER 3

# Feature-Based Motion Graphs

In this chapter we present feature-based motion graphs for realistic locomotion synthesis among obstacles. Among several advantages, feature-based motion graphs achieve improved results in search queries, eliminate the need of post-processing for foot skating removal, and reduce the computational requirements in comparison to traditional motion graphs. Our contributions are threefold. First, we show that choosing transitions based on relevant features significantly reduces graph construction time and leads to improved search performances. Second, we employ a fast channel search method that confines the motion graph search to a free channel with guaranteed clearance among obstacles, achieving faster and improved results that avoid expensive collision checking. Lastly, we present a motion deformation model based on Inverse Kinematics applied over the transitions of a solution branch. Each transition is assigned a continuous deformation range that does not exceed the original transition cost threshold specified by the user for the graph construction. The obtained deformation improves the reachability of the feature-based motion graph and in turn also reduces the time spent during search. The results obtained by the proposed methods are evaluated and quantified, and they demonstrate significant improvements in comparison to traditional motion graph techniques.

## 3.1 Introduction

Realistic animation of virtual characters remains a challenging problem in computer animation. Successful approaches are mostly data-driven, using motion capture data, and often involving motion blending and search. While blending operations over motion segments adequately grouped are suitable to real-time performances, search techniques based on motion graphs provide minimum distortion of the captured sequences, and naturally allow the exploration of solutions in complex environments.

Given the achieved simplicity and quality of produced motions, motion graphs remain a popular method for motion synthesis. The fact that motion graph applications often require the use of search algorithms makes real-time performance to be difficult to achieve, in particular when searching for long motions. The focus of this work is to improve the performance of motion graphs for synthesis of locomotion among obstacles.

We propose a feature-based approach for constructing motion graphs. Feature-based motion graphs can be constructed significantly faster than traditional motion graphs by avoiding the pairwise comparison between all frames in the database. Instead, only suitable pairs of frames are chosen for transition evaluation. The chosen pairs are the output of feature detectors encoding relationships of interest among key joint positions of the character's skeleton. Once these suitable frames are detected, transitions are evaluated with the usual threshold-based comparison metric, thus maintaining the same quality of results when blending the two motion segments defined by one transition.

Depending on the application, different feature detectors can be employed. A wide range of applicable feature detectors have been proposed by Müller et al. [MRC05]. These feature detectors can be quickly evaluated and are based on spatial relationships between the joints of the character at any given frame. We have noticed that, for the purpose of locomotion synthesis, a very small set of feature detectors is sufficient to successfully segment various walking motions. For example, the forward walk detector checks for a crossing event at the ankle joints, leading to motion segments with one foot always planted on the floor. This is a desirable property as it eliminates the need for post-processing due foot skating artifacts. Similar strategies were demonstrated by previous authors [TLP07], but employing manually crafted clips.

We also present a search pruning technique based on planar channels with guaranteed clearance from obstacles. This is achieved by projecting all obstacles on the floor plane and maintaining a local clearance triangulation of the environment [Kal10]. For any given start and goal positions, the triangulation returns a collision-free channel that is used to prune the branches that lie outside the channel during the unrolling of a motion graph search. This results in improved search times, especially in environments with many obstacles.

One inherent problem of discrete search methods is the difficulty of reaching precise targets. In order to address this problem we introduce a simple and efficient Inverse Kinematics (IK) deformation technique. Each branch of the search is treated as a kinematic chain of motion segments with joint limits representing the allowed rotational deformation at transitions. This technique produces motions precisely reaching given targets, and at the same time leads to an earlier successful search termination in several cases. These benefits do not sacrifice the quality of the motions as the quality of the deformed transitions will not exceed the same predefined threshold error used to construct the graph.

Lastly, we present extensive experiments with our methods solving locomotion synthesis among obstacles, and our results demonstrate significant improvements in time of computation, in finding solutions, and in the quality of the results.

## 3.2 Related Work

A large body of work has been devoted to animating characters using human motion capture data [KGP02, AF02, AFO03, LCR02, PB02, LWS02, PSS02, KS05, RBC98]. Motion graphs [KGP02, AF02, LCR02, GSK03] represent a popular approach that is based on the simple idea of connecting similar frames in a database of motion capture examples. Once a motion graph is available, graph search is performed in order to extract motions with desired properties.

Kovar et al. [KGP02] cast the search as an optimization problem and use a branch and bound algorithm to find motions that follow a user specified path. Arikan and Forsyth [AF02] build a hierarchy of graphs and use a randomized search to satisfy user constraints. Arikan et al. [AFO03] use dynamic programming to search for motions satisfying user annotations. Lee et al. [LCR02] construct a cluster forest of similar frames in order to improve the motion search efficiency. All these methods require quadratic construction time for comparing the similarity between all pairs of frames in the database. Our method improves on this operation by considering connections only between selected pairs of frames.

Many improvements to motion graphs have already been proposed. Ikemoto et al. [IAF07] use multi-way blends to create quick and enhanced transitions. Ren et al. [RZS10] combine motion graphs with constrained optimization. Shin and Oh [SO06] combine groups of parametrized edges into a fat edge for interactive control. Wang and Bodenheimer [WB03] evaluate cost functions for selecting transitions. Beaudoin et al. [BCP08] use a string-based model to organize large quantities of motion capture data in a compact manner. Our contributions focus on optimizing the motion graph construction, representation, and search, and can be used to improve any previous method relying on a motion graph structure.

Motion capture data has also been extensively used for locomotion planning among obstacles [EAP06, PZL10, KL00a, LK06, BEG11, SKG05]. In particular, Lau and Kuffner [LK05] manually build a behaviour-based finite state machine of motions, which in later work is precomputed [LK06] to speed up the search for solutions. Choi et al. [CLS02] combine motion segments with probabilistic roadmaps. These methods however require the user to manually organize motion examples in suitable ways.

The approach of *interpolated motion graphs* [SH07, ZS08, LS09] is based on an interpolation of two time-scaled paths through the motion graph. Although the method increases the solution space, this comes with the expense of increasing the time spent to build and search the graph structure.

In respect to our proposed IK motion deformation technique, a related approach has been explored before by Kanoun et al. [KLY11] for the problem of footstep planning for humanoid robots. The work describes an IK formulation that deforms a kinematic chain connecting footstep locations under specific constraints ensuring motion generation. Our method, however, is designed to solve

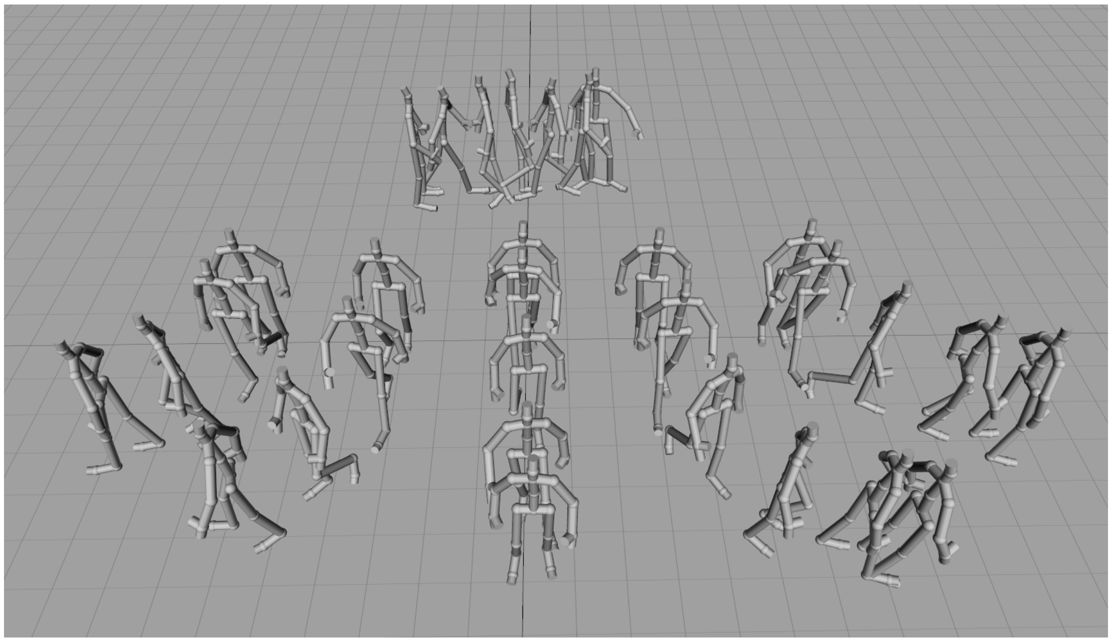

Figure 3.1: A motion capture database containing walking motions.

a quite different problem, that of deforming motion transitions under a global motion transition threshold, and without generating foot skating artifacts.

In general, the main drawback of traditional motion graph structures is the prohibitively large amount of data that may be needed in order to address practical applications involving obstacles and precise placements [RP07]. This chapter presents new techniques for addressing these problems and together they significantly improve the performance of motion synthesis from motion graphs.

## 3.3 Standard Motion Graphs

Before introducing feature-based motion graphs, we first explain how traditional motions are constructed. The procedure typically involves few steps: first, a subject is brought to a motion capture facility where one's motion may be recorded. Usually, the facility is a large room with with many cameras recording the character, if an optimal system is used. Then once the motions are recorded, they need to be post-processed, in order to make sure that they are of the highest quality. The end process is a list of motion files, which typically contains one type of motion. An example of a database, can be seen in Figure 3.1. It contains six files: straight walking motion, gentle left walking, gentle right walking, sharp left walking, sharp right walking and a 'u-turn' walking.

The next step is to combine all these files into one combined motion capture database. The goal is to find sweet points where artificial transitions may be created. Achieving this is not as trivial as it might appear first. The frames of the motion might have different global positions and orientations. Should the distance be calculated between the global joint positions or between the cloud points created by the down sampling of character's geometry? Which joints should be weighted more and which ones less? One could weight the root orientation more than the wrist joint but this depends on the current posture of the character. For instance, if the character is doing an acrobatic motion and it is holding itself on one of its hands, the wrist orientation might be more important then the root orientation. These considerations were discussed in the works of Kovar et al. [KGP02] and very convincing methods have been developed to address this problems. The distance metric introduced next has been used extensively and has proven to capture very well the similarity between two frames. We have deployed this metric for our own works.

The distance between two frames $A_i$ and frame $B_j$ is calculated by considering the point clouds formed over two windows of frames of user-defined length $k$, one bordered at the beginning by $A_i$ and the other bordered at the end by $B_j$. The use of a window frames incorporates derivative information into the metric hence preserving the speed and acceleration. In our test cases the length of the windows were set to 30, corresponding to a half of a second.

A motion is defined only up to a rigid 2D coordinate transformation. That is, the motion is fundamentally unchanged if we translate it along the floor plane or rotate it about the vertical axis. Thus, comparing two point clouds requires identifying compatible coordinate systems. With other words, we need to find a linear transformation $T_{\theta,x_0,z_0}$ that minimizes the following function:

$$\sum_i w_i ||p_i - T_{\theta,x_0,z_0} p'_i||^2 \tag{3.1}$$

where $p_i$ and $p'_i$ are two corresponding points from different point clouds. When the partial derivatives of the sum above are taken in regards to $x$, $z$ and $\theta$ we obtain the following close-form solution:

$$\begin{aligned}
\theta &= arctan\left(\frac{\sum_i w_i(x_i z'_i - x'_i z_i) - \frac{1}{\sum_i w_i}(\bar{x}\bar{z'} - \bar{x'}\bar{z})}{\sum_i w_i(x_i x'_i + z_i z'_i) - \frac{1}{\sum_i w_i}(\bar{x}\bar{x'} + \bar{z}\bar{z'})}\right) \\
x_0 &= \frac{1}{\sum_i w_i}(\bar{x} - \bar{x'}cos(\theta) - \bar{z'}sin(\theta)) \\
z_0 &= \frac{1}{\sum_i w_i}(\bar{z} - \bar{x'}sin(\theta) - \bar{z'}cos(\theta))
\end{aligned} \tag{3.2}$$

where $\bar{x} = \sum_i w_i x_i$ and the other barred terms are defined similarly.

Given the distance metric, the next step involves checking whether there is a smooth transition between any two motion frames. If the distance is less then a threshold, then a link is created between these two frames. Figure 3.4 shows a typically error image for a motion capture database. The red dots are the local minima where the transitions are created. At this point a traditional motion graph may be built. The entire motion capture database is segmented at points whether the low error local minima occur. The nodes of the graph are the segmented motion clips and the automatically created transitions and the edges of the graph are decision points were more then one motion clip is available. The last step is to prune dead ends. Without the pruning mechanism there is no guarantee that motion graphs can generate continuous motions. Dead ends are nodes of the graph where there are no outgoing transitions and sinks are nodes that restrict the motion generation into a small part of the graph. To handle this issue, a maximum strongly connected subgraph (SCC) of the motion graph is computed. A graph is strongly connected if for every node $u$ and $v$ there is a path connecting $u$ and $v$. The maximum SCC is computed with the help of the Tarjan's algorithm [Tar72] which is linear on the number of nodes and edges. All the nodes and edges that are not part of the maximum SCC are removed from the graph, leaving the rest of the nodes and edges fully defining a motion graph.

## 3.4 Feature-Based Motion Graphs

We build our feature-based motion capture database by concatenating multiple motions together. Each motion $M$ is composed of a sequence of frames. A frame $F = (p_r, q_0, q_1, ..., q_k)$ defines the pose of the character where $p_r$ is the root position and $q_i$ the orientation of the $i$-th joint. A motion is defined in respect to a skeleton $S$, which defines the joint hierarchy and contains the joint offsets. When a frame $F_i$ of a motion $M$ is applied to its skeleton $S$, the global joint positions of the skeleton can then be calculated.

We start by segmenting the motions into semantically similar clips using feature detectors. A feature detector $\phi(F_i) \rightarrow \{0, 1\}$ is a binary function that evaluates whether a frame satisfies a feature property as defined by the feature detector. A motion is segmented at a frame $F_i$ if $\phi(F_i) \neq \phi(F_{i+1})$. In this manner, for each motion type of interest, a feature detector is assigned to it.

In this chapter, our focus is locomotion and for this purpose we have noticed that the following two feature detectors suffice for an elaborate locomotion planner. We use these feature detectors:

- Walking Feature Detector
- Lateral Feature Detector

For walking motions (straight and turning) we wish to segment the motions such that a clip contains one walk cycle. In order to achieve this, we use the walking feature detector devised by Müller et al. [MRC05]. This feature was originally used to extracts relatively similar motions from a huge motion capture database by providing an example motions. However, in our case we use for the purpose of efficient motion segmentation. The walking feature works in the following way: each frame is tested against a foot crossing binary test which looks whether the right ankle of the character is behind or in front of the plane created by the left hip, right hip and the left ankle joint positions. This rule leads to a robust segmentation procedure as shown in Figures 3.2 and 3.3.

Because our motion capture database also contains lateral stepping motions, we devised a new lateral stepping feature detector. This novel feature was not among the feature detectors developed by Müller et al. [MRC05]. For these type of motions, we needed the segmentation points to occur at frames were the character exerted forced to the floor, similarly to the walking feature detectors. This was best achieved, by a feature detector that segments the motion when the velocity of both the left and right ankle joints is close to zero. See Figure 3.3 (top) for an example.

The segmented clips start and end with frames that are very similar to the equivalent ones in the other segmented motion clips. This segmentation procedure is, therefore, suitable for a motion graph construction. Additional feature detectors can be designed in order to segment motions of different nature. Adding new feature detectors is straightforward and simple rules can achieve robust segmentation. For the purpose of locomotion synthesis, the two described segmentation criteria suffice to ensure that useful clips are obtained. These clips are semantically similar in form and length and could potentially be categorized, parametrized or dropped if sufficient amount of motion segments have already been segmented.

Another advantage of feature-based segmentation is the automatic avoidance of foot-skating artifacts. Since blending operations are performed only at the extremities of each segmented clip, where there are only frames with one foot in contact with the floor, the skeleton can be re-parented at the contact foot before the transition blending operations, ensuring that the contact foot is not altered from its original position. This is done during transition generation avoiding any IK-based post-processing step for foot-skating correction. Motions extracted from a feature-based motion graph contain no foot skating.

Our motion graph is then formed by performing a pairwise test between the initial and final frames of each pair of segmented clips. A transition is created whenever the frame comparison metric returns a value under the transition threshold pre-specified by the user. We use the same distance metric and alignment transformation as in the original motion graph work [KGP02].

We also compute during construction time the allowed rotational range at each transition, as required by our IK-based deformation method (described in Section 3.8). We start with the initial transformation as returned by the metric

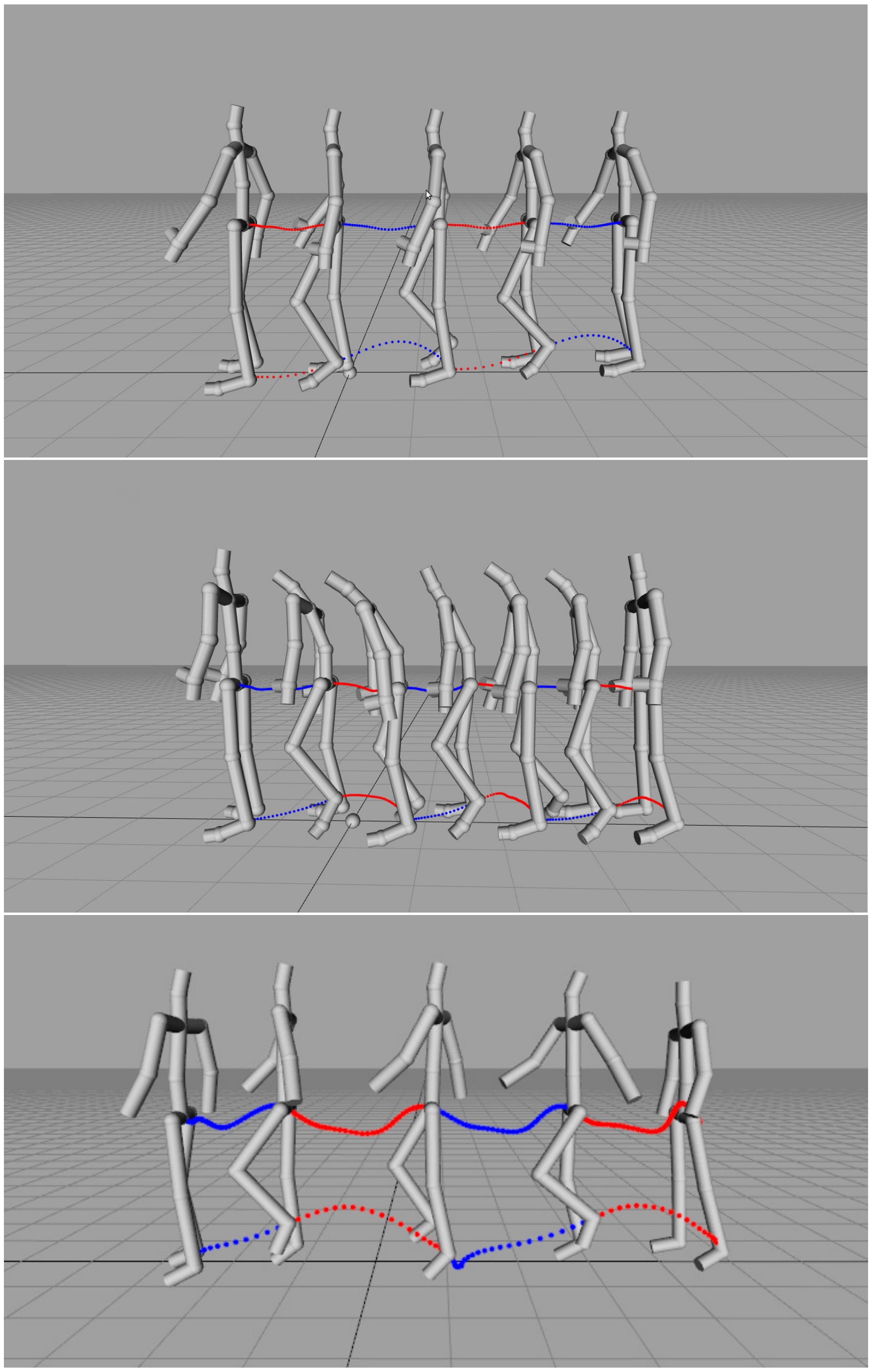

Figure 3.2: Feature segmentation robustly segments walking motions into walk cycles. The images here show the correct segmentation obtained for a normal walking motion, sad walking motion and happy walking motion. Alternating colors indicate segmentation points.

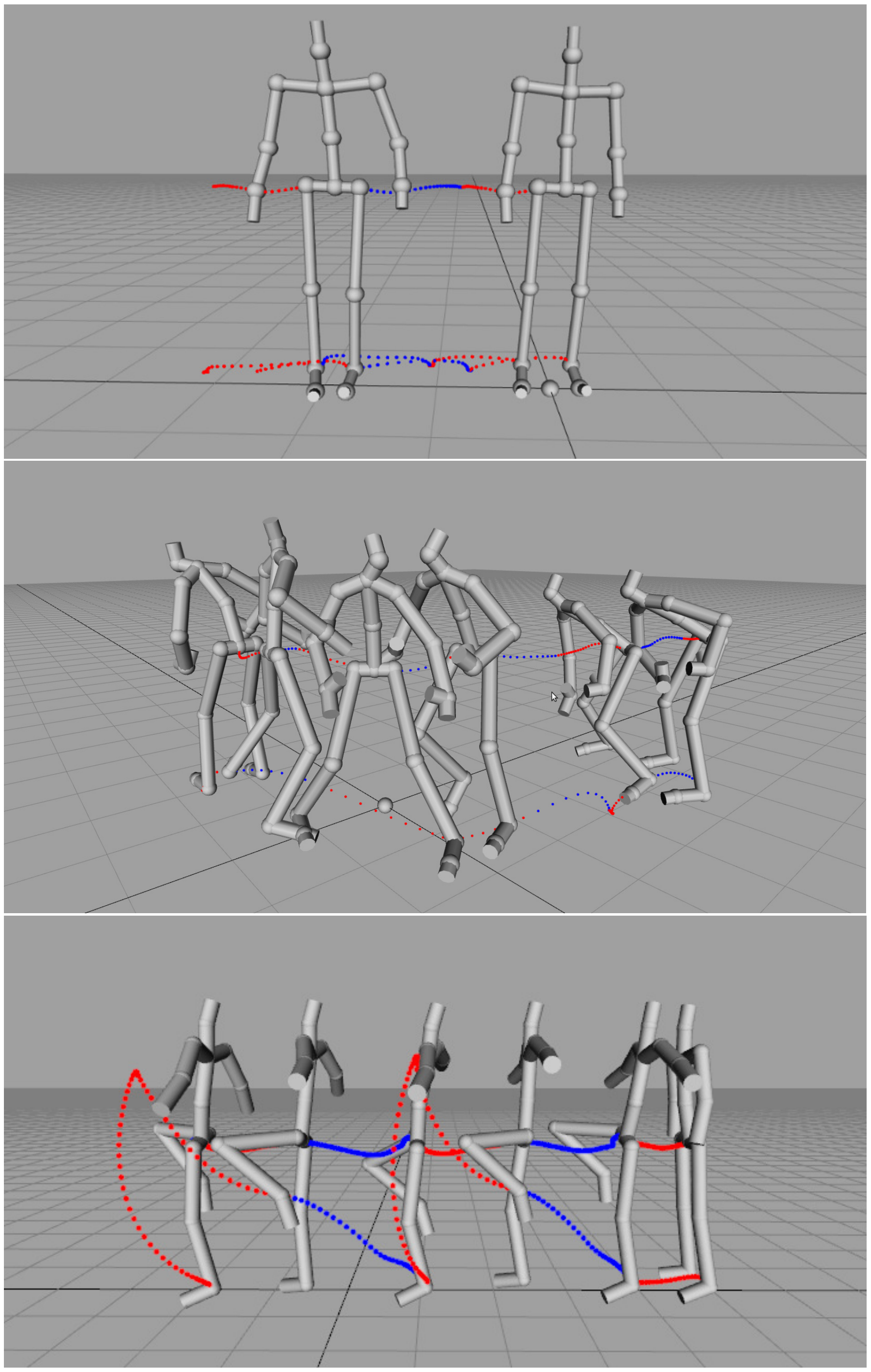

Figure 3.3: Feature segmentation is robust for different types of motions. The top image shows the segmentation of a lateral stepping motion, the middle image shows the segmentation of a basketball motion and the bottom image shows the segmentation for a ballet motion. For clarity, the character posture is shown only at every second segmentation for the lateral and basketball motions.

|  | SMG |  |  |  | FMG |  |  |  |
|---|---|---|---|---|---|---|---|---|
| Frames | Time (s) | Nodes | Edges | BF | Time (s) | Nodes | Edges | BF |
| 693 | 208.1 | 42 | 16 | 1.27 | 1.5 | 51 | 80 | 1.61 |
| 1009 | 467.2 | 69 | 28 | 1.28 | 5.9 | 22 | 11 | 1.33 |
| 1539 | 1107.4 | 137 | 67 | 1.32 | 7.0 | 33 | 137 | 1.81 |
| 1660 | 1297.7 | 135 | 59 | 1.30 | 6.3 | 35 | 125 | 1.78 |
| 2329 | 2577.5 | 188 | 81 | 1.30 | 6.5 | 22 | 46 | 1.68 |
| 3347 | 4853.5 | 310 | 211 | 1.40 | 6.6 | 19 | 47 | 1.71 |
| 6887 | 22272.5 | 1245 | 933 | 1.42 | 27.0 | 100 | 403 | 1.80 |

Table 3.1: Numerical comparison between standard motion graphs (SMG) and feature-based motion graphs (FMG). Column "BF" illustrates the connectivity of each graph with its average branching factor, which is computed as the number of edges divided by the number of nodes.

and change the rotational component about the vertical axis in incremental steps, in both clockwise and counterclockwise directions, until before the transitional cost exceeds the pre-defined threshold. The achieved range is stored at each transition and defines the allowed rotational range during IK motion deformation.

As a final step, the largest strongly connected subgraph of the graph is selected. The obtained subgraph represents the final feature-based motion graph.

It is important to observe the significance of devising good feature detectors capable of picking appropriate transition points for the types of motions to be used in the motion graph. If the presented feature detectors are not adequate for a given type of motion, the resulting graph may have poor connectivity and achieve poor performances in search queries. The presented methods were devised to work well with locomotion sequences and an extensive analysis of the obtained results are presented in the next sections.

## 3.5 Analyzing Feature-Based Graphs

Figure 3.4 compares the transition locations selected with the standard motion graph procedure against the proposed feature segmentation. These transitions are superimposed on the 2D error image of the entire motion database. The same frame comparison threshold is used for both methods. Note that the computation of the 2D error image is required by the standard motion graph but not by the feature-based motion graph. The transitions in the standard motion graph are selected by detecting local minima in the 2D error image; whereas for feature-based graphs, candidate transitions are determined by the feature segmentation directly and are independent of the 2D error image.

Figure 3.4 shows that the feature-based transitions often occur in similar locations as the local minima ones. Since the possible transition frames are segmented

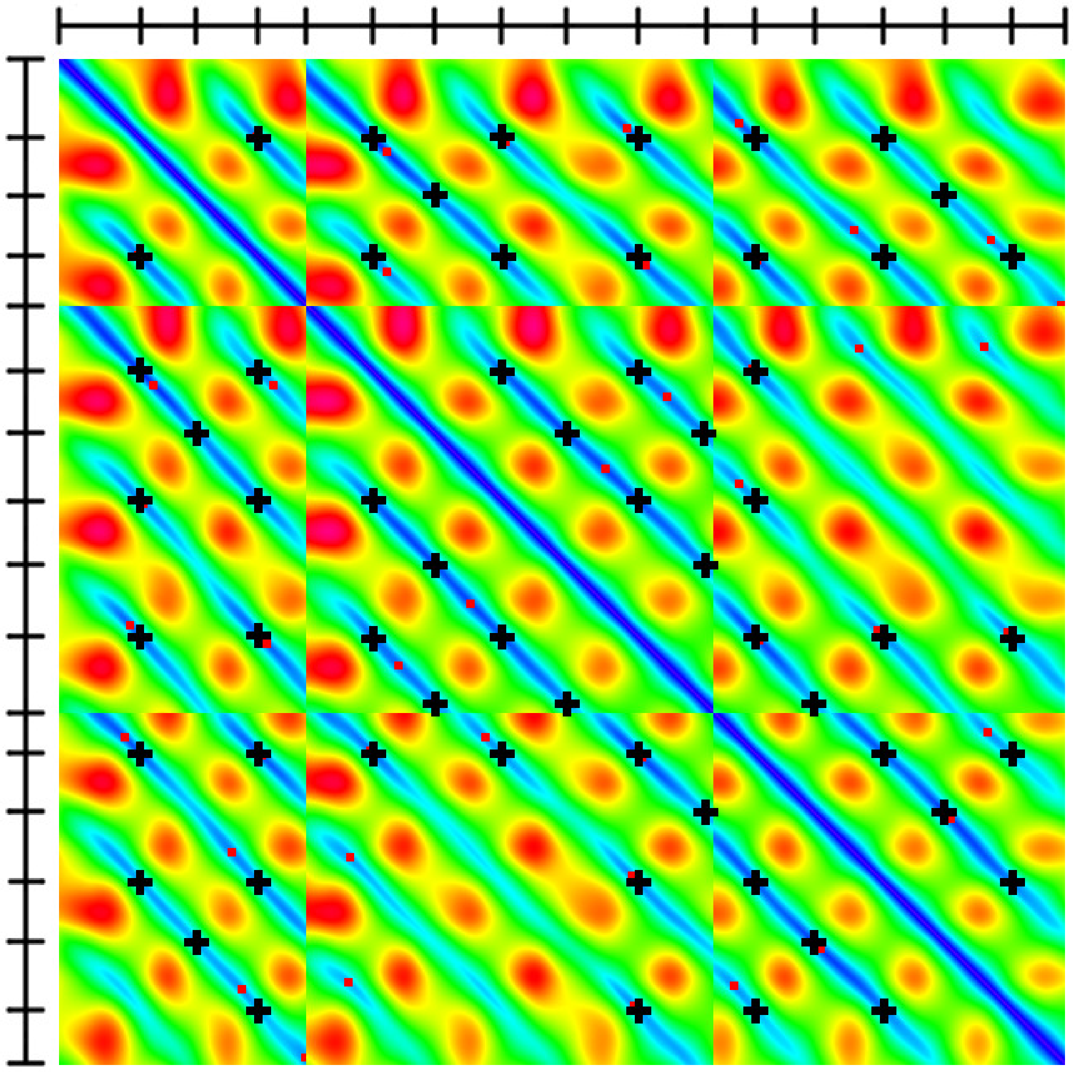

Figure 3.4: 2D error image between the frames of 3 walking motion cycles containing 693 frames sampled at 60 Hz. The red regions represent highest error and the blue regions represent lowest error. The red points marked are the local minima and the black crosses are transitions detected by the feature segmentation. There are 57 black transitions and 42 red transitions. The bars at the top and left of the image indicate the frames that were selected during the feature segmentation phase. Black transitions are always located at intersections of segmented frames.

by the same feature detectors, they satisfy the same geometrical constraints and thus have high chances of forming transitions. The motion capture database is also segmented at less points; moreover, our experiments show that the feature segmentation criterion will result on transition points that have higher connectivity with other transition points.

In order to evaluate our approach, we have computed both a standard motion graph (SMG) and a feature-based motion graph (FMG) from 6 different motion capture databases containing an increasing number of frames. Table 3.1 shows numerical comparisons between the structures, using the same transition threshold. It is possible to notice that FMGs have less nodes and more edges in most of the cases. The "BF" column in the table shows the average branching factor obtained in each case. This column clearly indicates that FMGs exhibit higher connectivity compared to SMGs in all cases. This property makes FMGs particularly suitable for our IK deformation method since more connections (and at suitable frames in the walk cycle) are considered for deformation.

FMGs are also computed much faster. The time spent for constructing a feature-based motion graph is often improved from several minutes to just a few seconds. Figure 3.5 depicts this comparison in logarithmic scale (with a base of 10) and shows that FMGs can be constructed 2 to 3 orders of magnitude faster than SMGs. For instance, it took 27s to create a feature-based graph from 6887 motion frames while the standard motion-graph took about 6h.

We have also analyzed the obtained size and connectivity in respect to the selected transition threshold. Figure 3.6 shows the number of nodes and average branching factor as a function of the transition threshold. The average branching factor is computed as the average number of edges per node. It can be observed that FMGs are more compact and better connected than SMGs. The higher branching factor for the FMGs is a direct consequence of checking transitions at selected suitable frames, allowing more than one transition to occur in a same lowest error connected component of the error image.

It is useful to observe why FMGs are more compact than SMGs with an example. In SMGs transitions can happen at any point $p = (i, j)$, where $i$ and $j$ are frame numbers. In FMGs transitions are only allowed at points generated by the feature segmentation. Suppose a motion $M$ with 20 frames (enumerated from 0 to 19) is given as input and the transitions from the local minima segmentation are: $T_m = \{(5, 7), (8, 14), (15, 10)\}$. In this case $M$ will be segmented in clips $C_m = \{0-5, 5-7, 7-8, 8-10, 10-14, 14-15, 15-19\}$, resulting in a graph with 7 nodes. FMGs will first generate a set $S$ of feature-based segmentation points, for example $S = \{5, 7, 10, 14\}$. Then, all transitions $\{(5, 5), (5, 7), (5, 10), ..., (14, 10), (14, 14)\}$ will be candidates but only those below the similarity threshold will be kept. Given that $M$ is the same in both cases, it is highly probably that the transition set will be $T_f = \{(5, 7), (7, 14), (14, 10)\}$. Note that $(8, 14) \in T_m$ is close to $(7, 14) \in T_f$ and $(15, 10) \in T_m$ is close to $(14, 10) \in T_f$. Given transitions $T_f$, motion $M$ will be

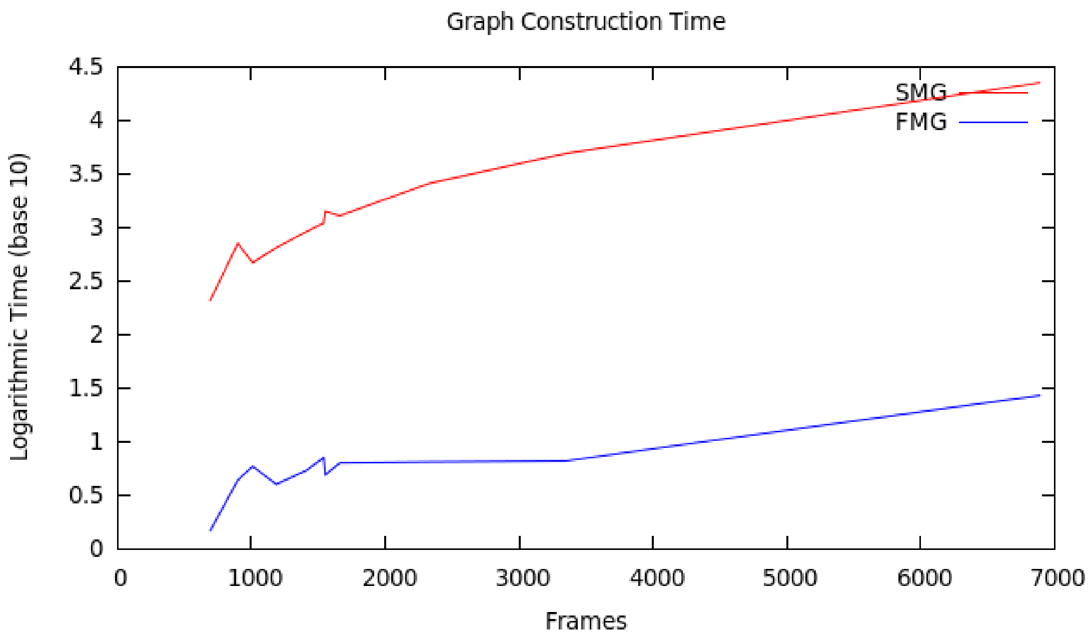

Figure 3.5: Construction time spent for SMG (top/red line) and FMG (blue/bottom line) as a function of the number of frames. The vertical axis represent time on a logarithmic scale (base 10). See also Table 3.1.

segmented in clips $C_f = \{0-5, 5-7, 7-10, 10-14, 14-19\}$; resulting in a graph of 5 nodes instead of 7. This example illustrates how the structured segmentation of FMGs often lead to graphs with fewer nodes, but also well representing the same motion capture database.

## 3.6 Locomotion Synthesis

After building the motion graph, the next step involves extracting the desired motions. Every node is a piece of motion and generating a motion is a matter of putting these pieces together in the correct position and orientation. That is, every node has to be transformed by an appropriate transformation to align the nodes correctly. This is achieved by applying the transformation which has already been computed to all of the frames in the outgoing node.

Searching for a motion that satisfies the user constrain could then be seen as an optimization problem that tries to minimize a given function [KGP02]. The cost of a graph walk can be seen as:

$$f(w) = f([e_1, e_2, ..., e_n]) = \sum_{i}^{n} g(w, e_i) \tag{3.3}$$

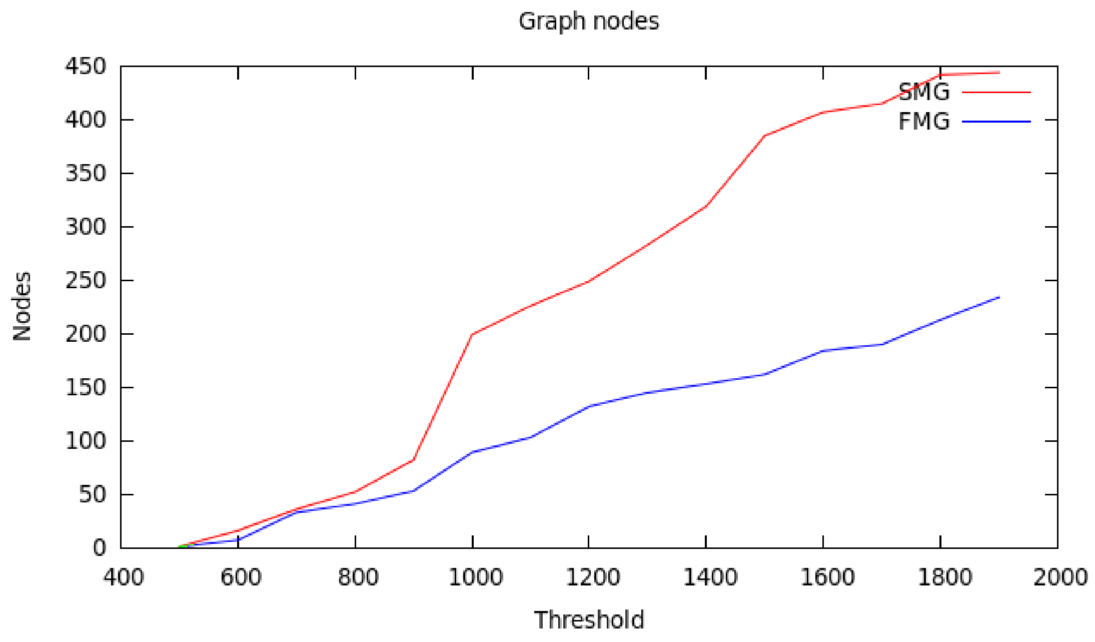

Figure 3.6: Number of nodes in the standard motion graph (SMG, in red/top) and in feature-based motion graph (FMG, in blue/bottom) as a function of the transition threshold.

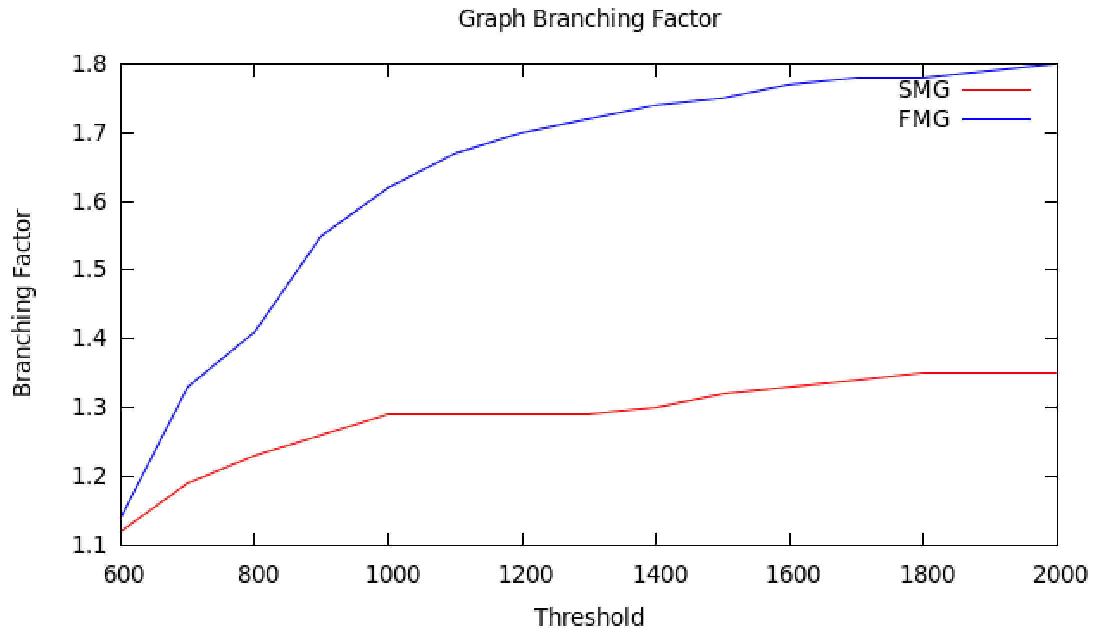

Figure 3.7: Average branching factor in SMG (red/bottom) and in FMG (blue/top) as a function of the transition threshold.

where function $g(w, e_i)$ represents the cost of appending an edge $e_i$ to the path $w$.

One typical example is to search for a path that will plan the motion of the virtual character moving it from a location $p_0$ to a location $p_1$. For this case, a graph search would suffice. However, a graph search alone is not an appropriate search because it tries to reach the goal position $p_1$ without worrying about how it reaches there. We would rather want the character to follow a path given by the user. In order to achieve this, the $g(w, e)$ function is defined as the area between the requested path and the path currently being searched:

$$g(w, e) = \sum_{i=1}^{n} ||P'(s(e_i)) - P(s(e_i))||^2 \tag{3.4}$$

where function $s(e_i)$ returns the arc-length of the path at frame $e_i$ and the functions $P(l)$ and $P'(l)$ return a point at an arc-length of $l$ from the searched path and requested path respectively. The arc-length is computed from the start of the path $w$ therefore the function $g(w, e)$ takes the traveled path as an input.

Although this formulation minimizes the differences between the path and the solution motion, it has one major disadvantages—there does not exist a suitable heuristic that could estimate the residual cost of $g(w, e)$. Therefore, the search is reduced from an A* search to a uniform-cost search, which is much slower. Feature-based motion graph solved this problem by running an A* search within a channel. The search was still expanding nodes in A* search fashion, but nodes that were outside the channel were pruned. Hence, indirectly the search was following the path because the channel was defined by the path that needed to be followed.

As showed in Table 3.1, FMGs contain less nodes and higher connectivity between nodes. The higher connectivity is key for improving the solutions generated from search queries. In order to quantify and evaluate the solution space of the graphs, we now present several experiments measuring and comparing the quality of our solutions in different environments with obstacles.

Figure 3.13 illustrates the solution space of both graphs using color-coded error comparisons similar to the comparison methods by Reitsma et al. [RP07]. Each error image spans an environment with dimensions of 10 m by 10 m, and with cells of dimensions of 10 cm by 10 cm. For these comparisons we have not employed the triangulation-based pruning technique (Section 3.7) and the IK-based deformation (Section 3.8) as we are only interested in measuring the difference between FMGs and SMGs without any optimizations. The right-most column shows the error ratio $E = e_{smg}/e_{fmg}$ between both methods. A cell with black color means either an obstacle or that SMG is better, i.e. $E < 1$. For $1 < E < 2$, when FMG is better, the color scale is used ranging from blue, meaning slightly better, to red, meaning 2 times better. It is possible to notice that the errors obtained with the

FMG are almost always lower.

Table 3.2 summarizes the obtained statistics from 4 different environments with increasing number of obstacles. Both SMG and FMG were constructed using the same transition threshold. SMG had 318 nodes with an average branching factor of 1.31, whereas FMG had 79 nodes with an average branching factor of 1.59. Three metrics were used to compare the performance of the graphs: average optimal solution length, average expansion count and average search time (in seconds) spent during the graph search. A maximum expansion count of 100,000 was chosen for all the queries. If the maximum expansion count was reached and a solution was not found, a failure was reported. For sake of fairness, we only compared queries when both SMGs and FMGs were successful. The comparisons for SMGs and FMGs are presented in percentages and they range between $-100\%$ and $100\%$.

For the length comparisons we have defined the length error as follows: let $P$ be the length of the 2D path that needs to be followed and let $L$ be the length of the projection of the root joint trajectory of the solution motion that follows $P$. The length error is defined as $E = L/P$. Since the same set of trials were used for the comparison experiments, the value of $P$ is the same, and therefore we only use $L$ as a metric for comparing the length quality of the generated trials.

As Table 3.2 shows, in average, FMGs expand less nodes, spend less time searching and produce smaller errors in all of the cases. In average, FMGs show an improvement between 1-3% in length error and between 40-60% in search time in comparison to SMGs thanks to the increased solution space achieved with the higher connectivity. Our results show that choosing transitions at local minima does not always yield better results. Instead, transitions at key points detected by the feature segmentation generate better results and lead to faster searches.

## 3.7 Triangulation-Based Search

The search procedure used in the previous section represents the standard search solution for extracting locomotion sequences from a given motion graph. We now describe our improved search for faster motion extraction, which is based on constraining the search to pre-computed channels. While the idea of pruning the search has been explored before [SH07, CK04], our approach employs a new technique based on fast geometrically-computed channels.

We first compute a free 2D path on the floor plane between the current position of the character and the target position using an available triangulation-based path planning technique [Kal10]. The computed paths are obtained well under a millisecond in the presented environments. The path is computed with guaranteed clearance, therefore guaranteeing that sufficient clearance for the character to reach the goal is available (see Figure 3.8 and 3.9).

| Searches | | | SMG | | | FMG | | | Mean (%) | | | Std. Dev. (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total | Mut. | Perc. | Time | Len. | Exp. | Time | Len. | Exp. | Time | Len. | Exp. | Time | Len. | Exp. |
| 7920 | 7386 | 93.26 | 0.41 | 602.51 | 3623 | 0.32 | 582.88 | 1805 | 45.43 | 3.13 | 66.99 | 40.08 | 6.69 | 35.32 |
| 6435 | 5930 | 92.15 | 2.18 | 777.84 | 17463 | 1.85 | 752.11 | 9498 | 39.48 | 3.85 | 59.76 | 39.72 | 5.41 | 36.45 |
| 5092 | 4423 | 86.86 | 2.81 | 890.79 | 22259 | 0.94 | 879.28 | 4429 | 60.84 | 1.31 | 76.07 | 16.65 | 2.13 | 13.48 |
| 4286 | 3451 | 80.52 | 2.91 | 862.18 | 23128 | 0.96 | 850.87 | 4565 | 59.31 | 1.33 | 75.20 | 18.70 | 2.40 | 15.12 |

Table 3.2: Statistics when searching for locomotion sequences in different environments. The same transition threshold was used in both graphs. “Total” is the total number of attempted queries. “Mut” is the number of mutually successful queries. “Perc” is the percentage ratio between “Mut” and “Total”. “‘Time” is the average time spent searching for each solution in seconds, “Len” is the average arc-length of the solution motions measured along the character’s root trajectory projected on the floor, and “Exp” is the average number of node expansions during each search. The last six columns show the mean and standard deviation of improvements (in percentage ratios) of FMGs over SMGs.

Once a 2D path is available, we perform a graph search by unrolling the motion graph in the environment and expanding only the nodes that remain close enough to the path. Since the path is guaranteed to be free of obstacles within its channel (i.e. within a distance $r$ to the path), only nodes generating motion clips completely inside the free channel are expanded, and collision tests with the environment are not needed. As a result, faster searches are achieved by avoiding expensive collision checks, which represent a major computational bottleneck when employed.

We test if a motion clip remains inside the free channel by projecting the position of key extreme joints of the character (like the hand and feet joints) to the floor, and measuring if their distances to the path are smaller than $r$. The projected positions are taken from the final frame and few intermediate frames of each motion clip. Collision tests are, therefore, reduced to point-path distance computations.

The overall search procedure starts from the node in the motion graph containing the initial character pose. This node is expanded, and every valid expansion remaining inside the free channel is inserted in a priority queue $Q$ storing the expansion front of the search. The priority queue is sorted according to an A* heuristic function $f(node) = g(node) + h(node)$, where $g(node)$ is the cost-to-come value and $h(node)$ is the distance to the goal. The search stops when a node is within a distance $d$ to the goal or when the expansion count exceeds a certain limit, in which case failure is reported. In our experiments $d$ is set to 10cm and the expansion count limit is set to 1 million. The expansion count limit is necessary because otherwise the unrolling process could continue indefinitely.

Another advantage of finding a path in the environment before starting the unrolling process is to avoid local minima. In general, the A* search will expand nodes blindly towards the goal without any consideration on the placement of the obstacles in the environment. As such, the search spends a lot of time expanding nodes towards one local minimum which might not be a part of the final solution. Such cases occur, for example, when the goal is right behind an obstacle. With channel pruning, a collision free path is already known in the triangulated free space of the environment. Therefore, the path returned by the triangulation serves as a guide to the A* search by confining the search within the free channel.

Figures 3.10, 3.11, 3.12 clearly depict the advantage of confining the search within the computed channel. For the four environments showed in Figure 3.14, the improvements in search times were up to 40%. For a more complicated environment, such as the example in Figure 3.12, the pruning technique was able to find a solution 10 times faster.

Failure in the described search procedure can happen if the motion graph does not have suitable connectivity. In other words, failure will occur if the search frontier runs out of leaves due to the obstacles or channel pruning. Feature-based motion graphs present themselves as a better choice for preventing the search to

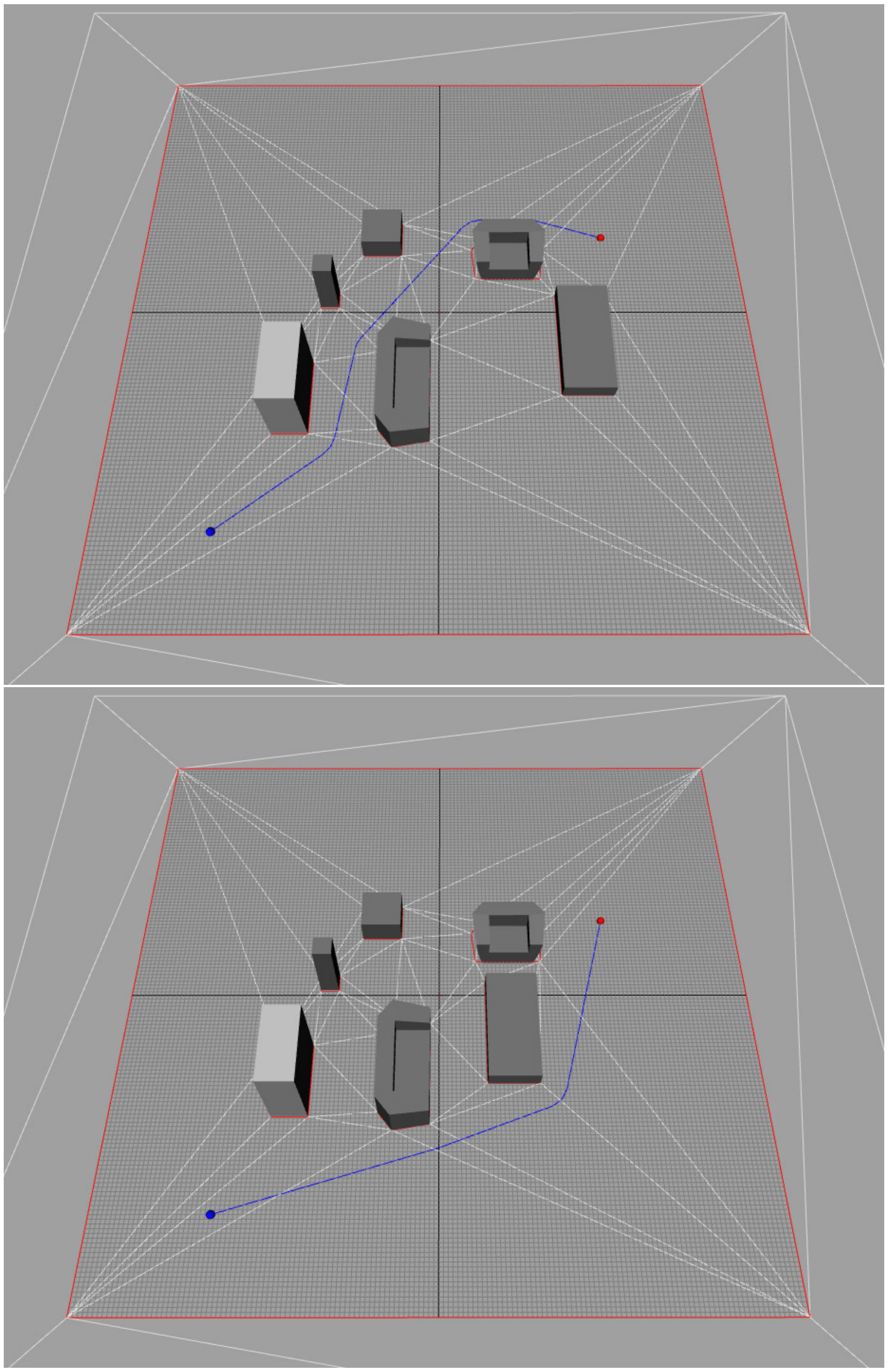

Figure 3.8: Example of paths with clearance. The sequence of images illustrate that the floor triangulation can be updated very efficiently if obstacles move (under a millisecond in an average computer when up to about 100 vertices are involved).

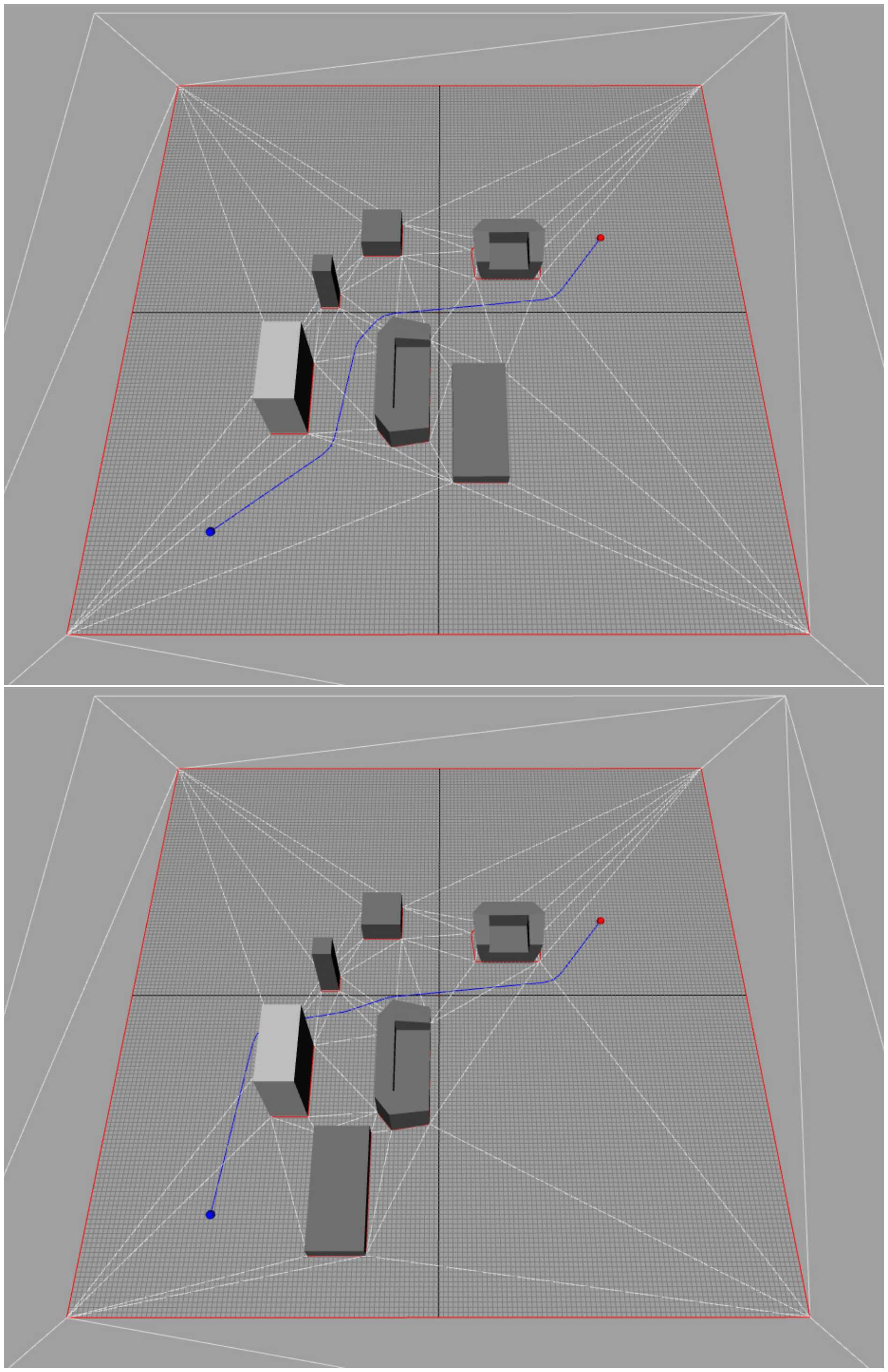

Figure 3.9: Channel pruning does not impose any requirements on the environment to be static and no other computation prior to a search query is needed.

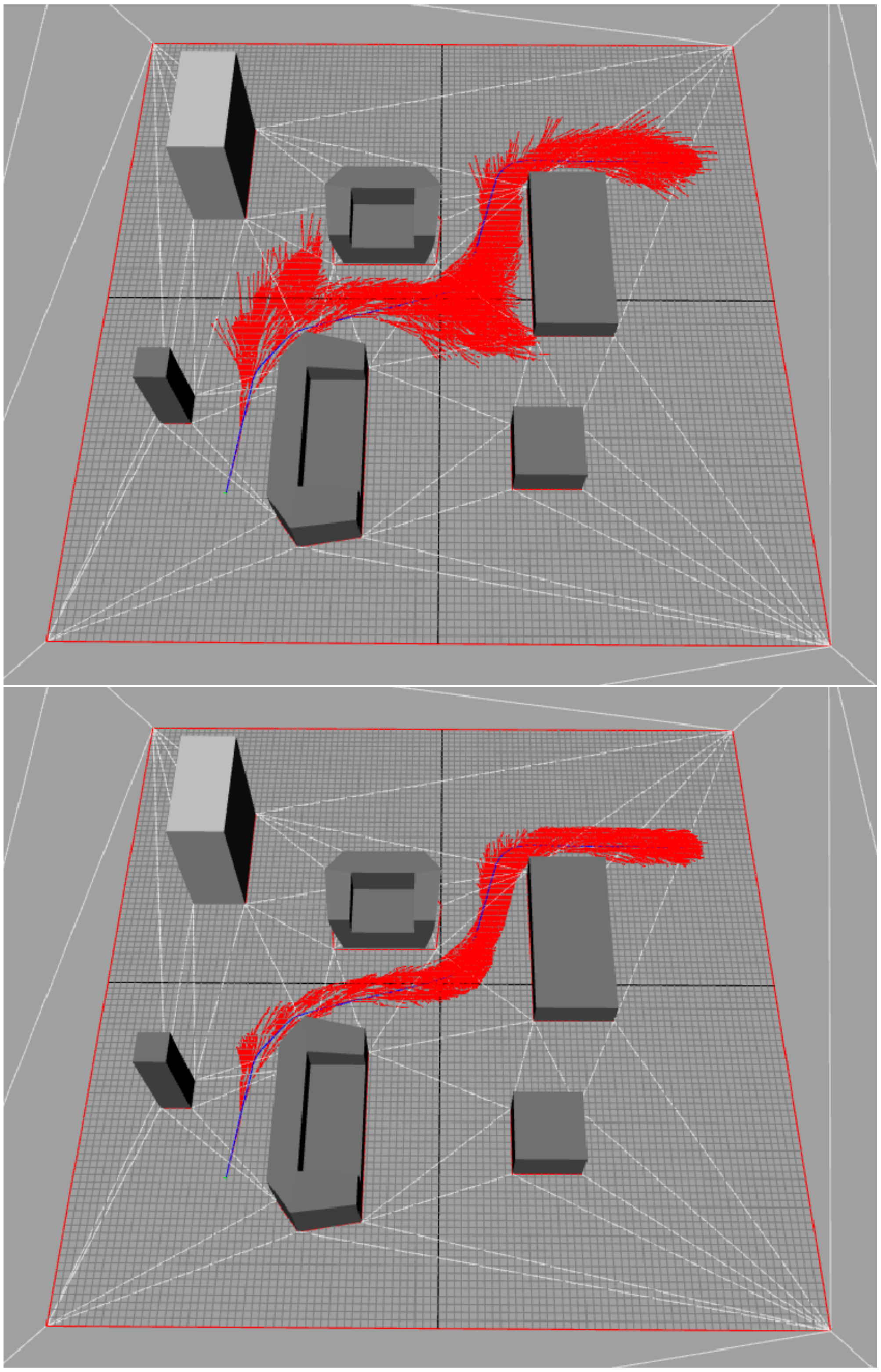

Figure 3.10: Unrolled branches in two different environments by a motion graph search with channel pruning disabled (top) and enabled (bottom).

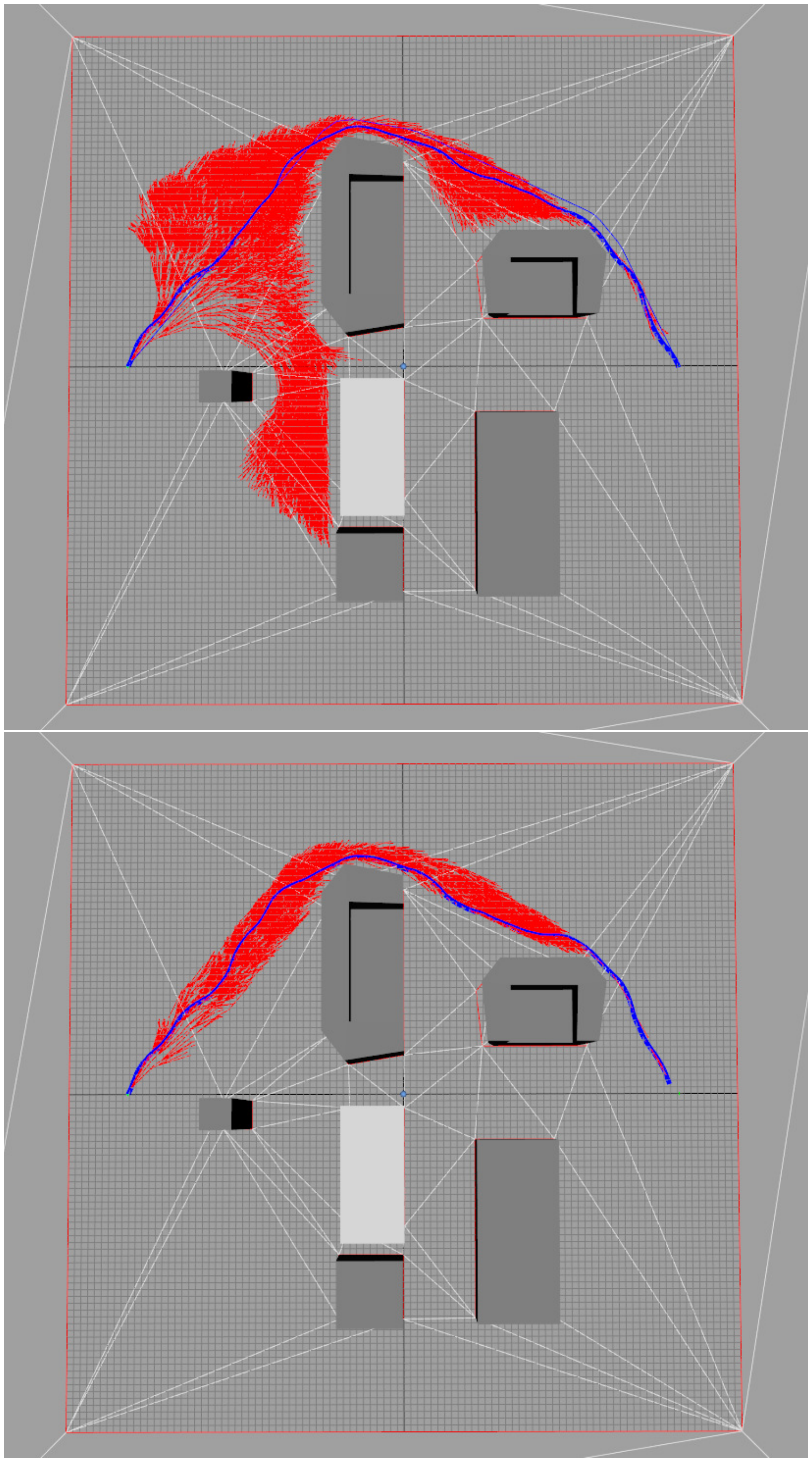

Figure 3.11: Unrolled branches in two different environments by a motion graph search with channel pruning disabled (top) and enabled (bottom).

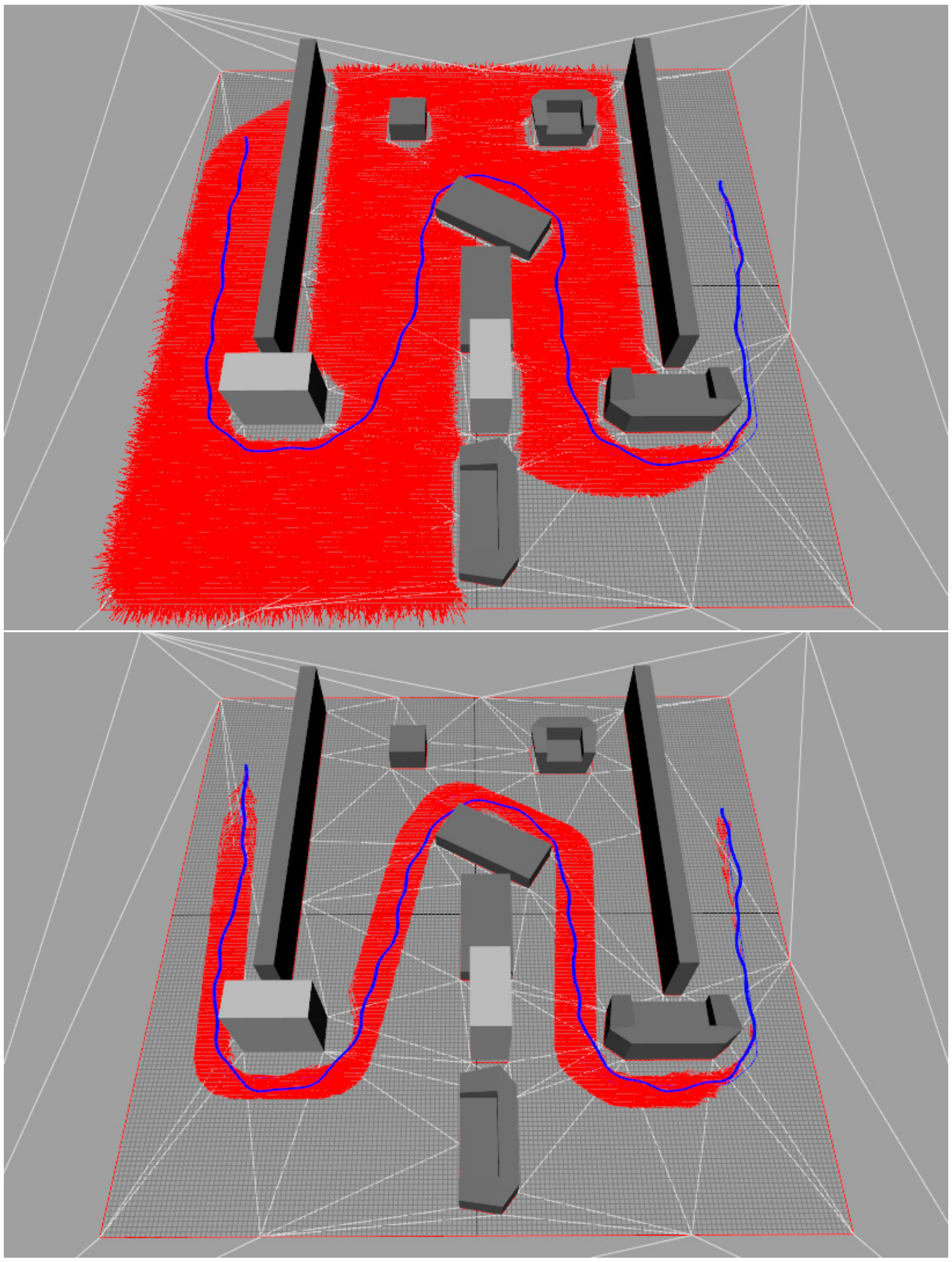

Figure 3.12: Unrolled branches in two different environments by a motion graph search with channel pruning disabled (top) and enabled (bottom).

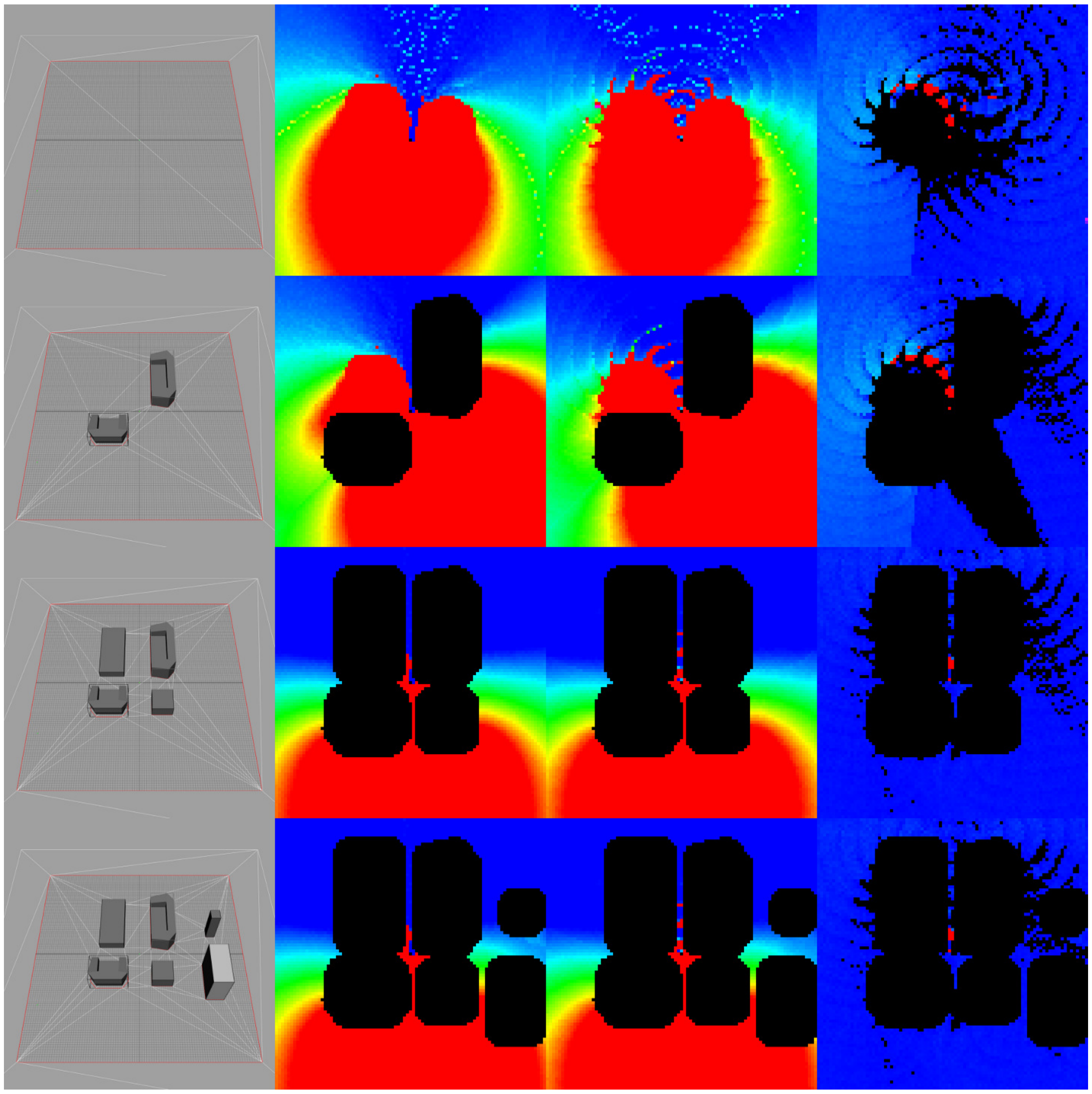

Figure 3.13: Color-coded error comparisons when searching for locomotion sequences. The first column shows the used environments. The second and third columns show the errors obtained with the SMG and FMG respectively. The error $e = l/p$ is the ratio between the length $l$ of the obtained motion from each method and the length $p$ of the Euclidean shortest path on the floor plane (including a path clearance). A blue color means a solution length very similar to the shortest path, a red color means maximum error, which here is set to 2 times the length ($2p$) of the shortest path. The character starts by facing the up direction and needs to first rotate down in order to reach the lower targets, which explains the large red areas in the images.

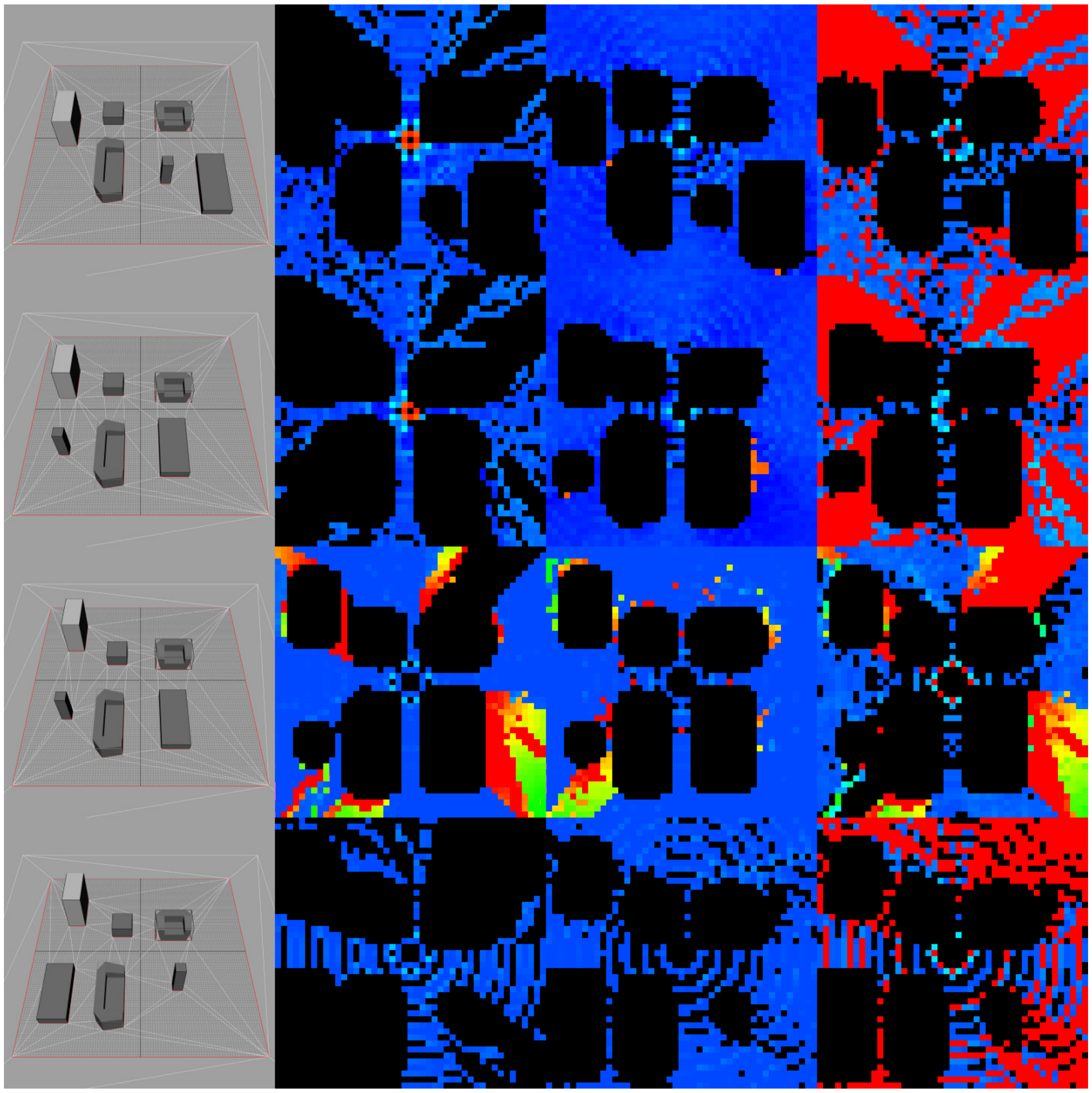

Figure 3.14: Color-coded error comparisons when searching for locomotion sequences with the precomputed channel pruning enabled. The color coding for the comparisons here is the same as in Figure 3.13, with the exception that in the right-most column a cell is set to red (maximum error) when FMG successfully finds a solution but SMG fails. Another difference in respect to Figure 3.13 is that, instead of always starting oriented upwards, here the character's starting orientation is set to face the channel prior to the search, what makes the character to always start with a forward walking motion. The presented comparisons show that FMGs have lower errors and higher success rates than SMGs in most of the cells.

stop before reaching the goal location of the path. For example, Figure 3.15 illustrates a typical situation where FMGs are able to produce a locomotion sequence successfully following the entire path, while the SMG structure makes the search to run out of connections before reaching the goal. Figure 3.14 shows several comparison results obtained with the channel-based search procedure. It is possible to notice that, due the higher connectivity obtained, FMGs practically always produce better results.

## 3.8 Inverse Branch Kinematics

Combined with our FMG and triangulation based pruning technique, a new *Inverse Branch Kinematics* (IBK) procedure is proposed for improving the obtained solutions.

As previously mentioned in Section 3.5, we compute a lower and upper limit for the rotational component of each created transition during the graph construction step. A transition $Tr(i, j)$ is generated between the $i$th and $j$th frame of the motion capture database by using a transformation $T_m$ that minimizes the distance between the interpolated motions as defined by the employed frame similarity metric. The limits are computed by measuring the error associated to rotational increments. For each new angle increment, the new transformation is calculated in the following manner:

$$T(\theta) = P_i^{-1}\ R_\theta\ P_i\ T_m, \tag{3.5}$$

where $T_m$ is the transformation as returned by the original metric introduced by Kovar et al. [KGP02], $R_\theta$ is the rotational transformation of angle $\theta$ about the Y axis and $P_i$ is the global position of the root joint of the $i$th frame of the motion database (see Figure 3.16). This procedure is repeated for both the upper and lower rotational limits while the transition cost does not exceed the global transition error threshold.

Later, for each motion search query, the search procedure is performed as previously described, stopping when a branch becomes close enough to the target in respect to a user-specified distance $d_o$. Then, the IBK solver is employed to iteratively optimize the solution towards the exact target location, up to a given tolerance $d_i$. Therefore, when $h(node) < d_o$ and $g(node) > dist(start, goal)$, $d_i < d_o$, the IBK procedure is invoked.

In other words, when the distance between the node being expanded and the goal is under $d_o$ and the length of the current path is longer than the Euclidean distance between the start and goal positions (meaning that there is room for a branch deformation), the search is then paused and the branch is deformed as a 2D kinematic chain with joint limits taken from the transition limits stored in the transitions. See Figure 3.17 for an example. In our experiments, we have set $d_i$ to

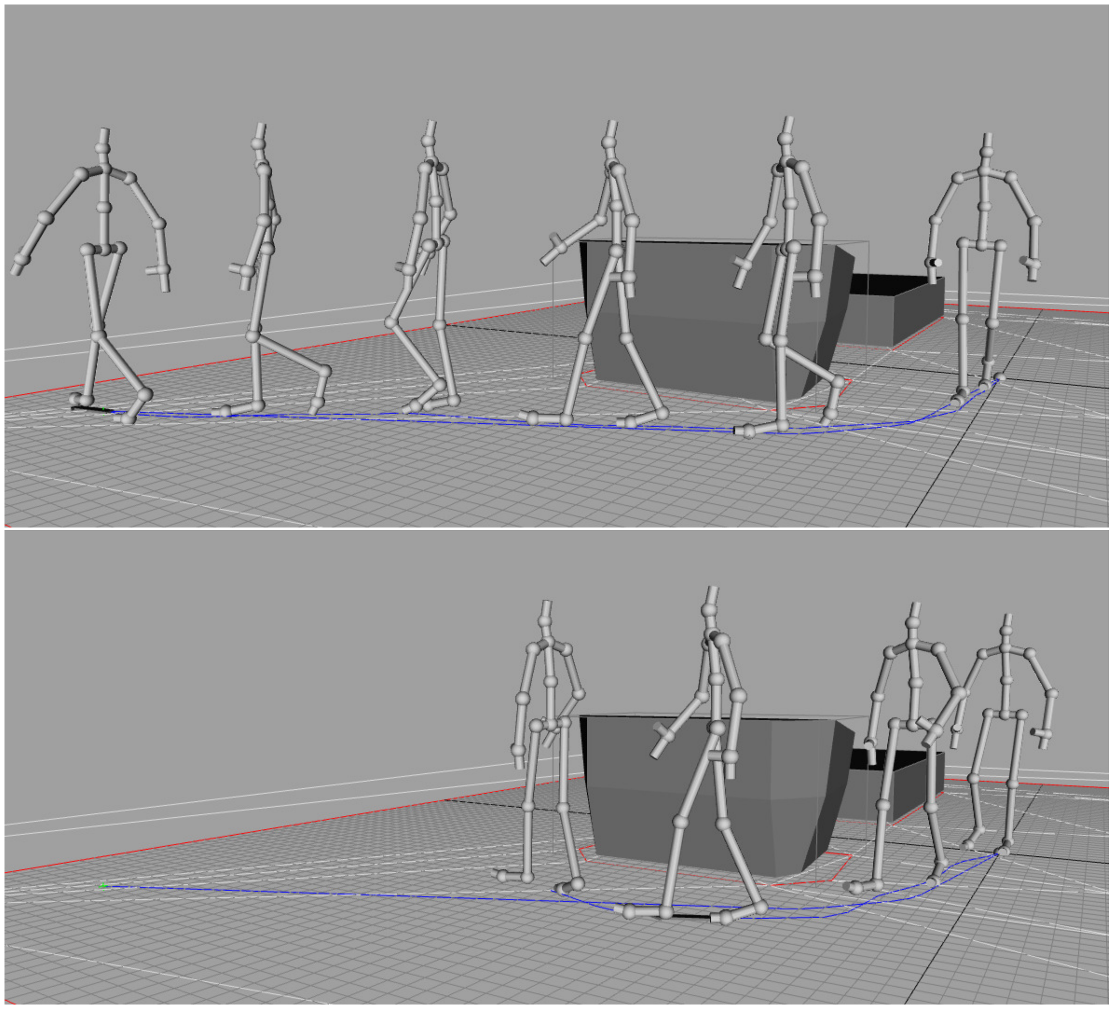

Figure 3.15: Top image: the channel search procedure on a feature-based graph successfully computed a motion following the entire path. Bottom image: the same search failed in the standard motion graph. The same motion capture database and transition threshold were used in both cases.

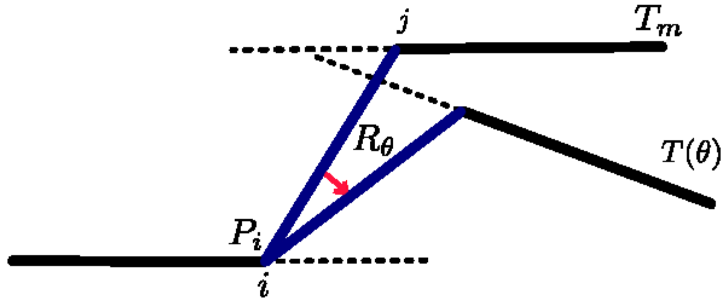

Figure 3.16: Computing rotational joint limits for the IBK search procedure. The blue segment represents a transition from $i$th to the $j$th frame of the motion capture database.

have the same value as parameter $d$ used in Section 3.7 (10cm), and we have set $d_o$ to 50cm. Parameters $d$ and $d_i$ are kept the same in order to achieve meaningful results in our performance evaluations.

We have determined the values of $d$, $d_i$ and $d_o$ empirically after experimenting with the method in a few motion queries. These values have worked well in all our examples and we have not observed the need to modify them according to each query. One important factor that influences the choice for these values is the average length of the motions being computed. If improvements are needed, one possible extension is to fine-tune the pre-defined value of $d_o$ as a function of the length of each obtained branch. In this way it is possible to have longer deformable branches being able to reach larger areas.

Depending on the nature of the transitions, the chain might have different joint limits. In Figure 3.17, the transition between the third and fourth node of the branch is not flexible; thus, this joint of the chain remains fixed. Also, the lower and upper joint limits do not have to be symmetric. For example, the second link has only room to move in respect to the upper limit. Once a candidate solution chain with its joints limits is obtained, the IBK solver can then evaluate rotational values at the joints in order to reach the target with the end-node of the search path.

Several experiments were performed and our solver achieved best results with a Cyclic Coordinate Descent (CCD) solver [WC91]. We have in particular experimented with a Jacobian-based pseudo-inverse solver, however, in our highly constrained 2D kinematic chains, the much simpler CCD solver was faster.

Each CCD iteration increments rotations at each joint, starting from the base joint towards the end-effector joint. At each joint two vectors $v_{end}$ and $v_{goal}$ are calculated. Vector $v_{end}$ is from the current joint to the end-effector and $v_{goal}$ is the vector from the current joint to the goal. These two vectors are shown in

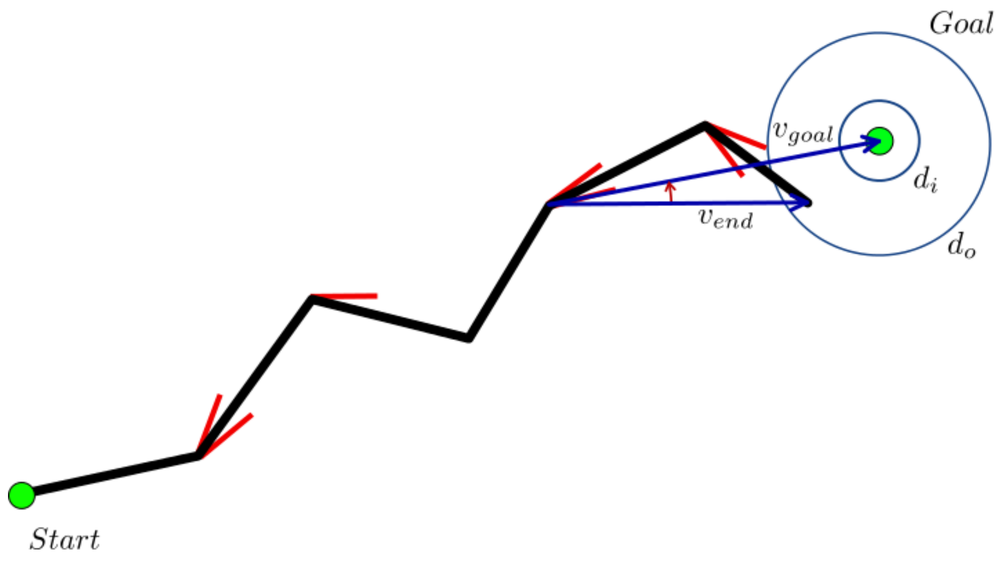

Figure 3.17: A graph branch represented as a kinematic chain. Motion transitions are represented as rotational joints and the red segments represent the joint limits which are identical to the corresponding rotation limits stored in the transitions.

Figure 3.17 for the fifth link of the chain. The angle between the two vectors is incremented to the current joint and the result clipped against the joint limits. The last step of the CCD iteration consists of calculating the improvement from the previous iteration, which is given by how much closer the end-effector is to the target.

The CCD iterations stop when no improvements are detected after a number of iterations. At this point, if the distance between the end-effector and the goal is less then $d_i$ then the solution with its new rotation values is evaluated for collisions. If no collisions occur, success is reported otherwise the optimization search continues until another candidate branch is obtained. Figure 3.18 illustrates that in several cases the IK deformation is able to achieve a solution that otherwise the alternative solution without deformation would not be acceptable.

Table 3.3 shows the effect of employing the IBK solver for SMGs and FMGs with both the channel pruning enabled and disabled. As it can be seen from the table, IBK improves the generated solutions and reduces the search time in all the cases. The average improvement in the length of the motions when channel pruning is enabled is about 17% and 5% when pruning is disabled. The improvement on average on the search time is 31% when channel pruning is enabled and 21% when channel search is disabled. The reduced search time is a direct consequence of being able to terminate the search process early. This is possible because branches that are close to the goal can be deformed to meet the goal precisely.

The IBK optimization does not impose noticeable performance degradation since only a few iterations are performed in each search query and iterations only

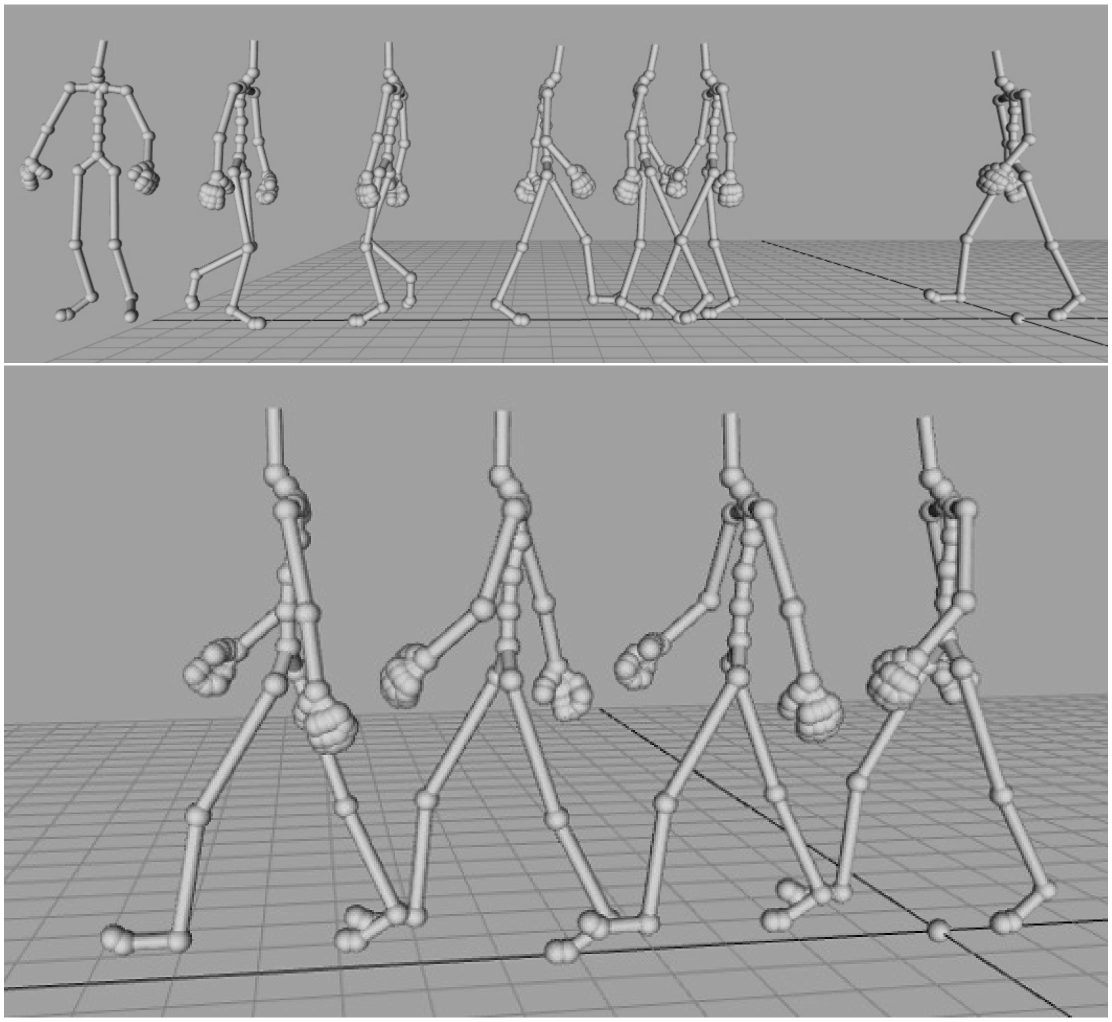

Figure 3.18: The top image shows a typical problematic motion graph solution where an overly long motion is returned as solution to a given target. The bottom image shows the correct solution obtained by coupling the search with the IBK solver, which is able to deform nearby motions to the exact target without degrading the quality of the solution.

|  | SMG |  |  |  |  |  | FMG |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | Channel Pruning |  |  | Without Channel |  |  | Channel Pruning |  |  | Without Channel |  |  |
| Env. | Length | Exp. | Time | Length | Exp. | Time | Length | Exp. | Time | Length | Exp. | Time |
| 1 | 17.9 | 73.3 | 36.8 | 1.5 | 55.0 | 29.2 | 18.0 | 80.0 | 57.2 | 5.0 | 49.1 | 30.8 |
| 2 | 17.6 | 67.4 | 29.9 | 2.5 | 53.3 | 21.8 | 16.6 | 75.6 | 43.9 | 4.5 | 49.3 | 13.5 |
| 3 | 17.6 | 62.7 | 23.4 | 5.6 | 58.1 | 23.9 | 13.7 | 67.1 | 19.9 | 4.2 | 56.5 | 19.0 |
| 4 | 17.6 | 67.3 | 29.9 | 5.6 | 48.1 | 15.1 | 12.9 | 55.2 | 6.2 | 4.4 | 47.4 | 14.6 |

Table 3.3: Improvements gained when deploying IBK during search. Comparisons for both SMGs and FMGs with and without channel pruning for four different environments are shown. All values are represented as percentages. Each value is calculated as follows: if $v$ is the value measured without deploying IBK and $v_{ibk}$ is the value with IBK deployed then the reported percentage $p$ is calculated a $p = -(v_{ibk} - v)/v$.

| | None | | | Small | | | Medium | | | Large | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Techn. | Len. | Exp. | Time | Len. | Exp. | Time | Len. | Exp. | Time | Len. | Exp. | Time |
| F | 8.7 | 60.1 | 31.0 | 10.3 | 61.9 | 33.4 | 13.4 | 60.8 | 27.2 | 14.7 | 68.5 | 42.9 |
| FC | 12.9 | 79.0 | 49.2 | 13.8 | 78.8 | 37.1 | 15.1 | 74.3 | 15.3 | 16.7 | 78.2 | 27.1 |
| FCI | 10.4 | 91.8 | 67.3 | 11.2 | 87.0 | 42.6 | 11.4 | 14.2 | 19.9 | 14.6 | 85.3 | 34.7 |

Table 3.4: Improvements gained against SMGs for four different environments as the proposed techniques are deployed. All values are represented as percentages as explained in Table 3.3. “Techn” stands for technique and “Len” for length. Technique “F” uses only FMGs, technique “FC” uses FMGs and channel pruning, and technique “FCI” uses FMGs, channel pruning and IBK.

require extremely simple 2D operations.

The IBK deformation procedure not only improves search time and convergence to the goal point, but it is also formulated in a way to not introduce any undesirable artifacts such as foot skating. IK rotations are only allowed at transition points, which are always blended with the re-parenting strategy to the support foot in order to not generate foot skating. As a result, although IBK may introduce additional rotations to transitions, the resulting motion will automatically remain without foot skating artifacts.

## 3.9 Discussion

We now present several evaluations demonstrating the advantages of the proposed feature segmentation, channel pruning and IBK deformation. A video illustrating the presented results is available from the website of the authors [1].

The first obvious advantage of the proposed FMG is that the construction time is dramatically improved in comparison to the standard motion graph procedure as our method does not need to compute a full 2D error image of the motion capture database (see Table 3.1). The fact that we do not search for transitions in the quadratic space of possibilities does not impose any drawbacks. On the contrary, we have shown that feature-based graphs have more connectivity and most often lead to improved results when applying search methods for locomotion synthesis around obstacles, which is always a challenging problem for discrete search structures to address. For instance, Table 3.2 shows up to 60% improvement on the time spent searching in all four environments.

In addition to the significant improvement in construction time, feature-based segmentation also introduces semantics. For example, when specific feature detectors for forward and lateral steps are employed, each created motion segment in the graph can carry the label of its generating detector. This ability allows the user, for instance, to control the number of motion segments per feature type and to achieve compact graphs from large motion capture databases, or to specify search queries with only given types of motion segments in the solution. The employed feature detectors have also shown to robustly segment walking motions in several different styles.

Feature-based segmentation can furthermore help the user in determining the frame similarity threshold. For example, in the forward walk segmentation, all motion clips start and end with only one foot planted on the floor, with alternated support foot at the start and end frames. Naturally, there should not be a transition from the end to the beginning of a same motion clip, i.e. a reflexive transition. Lowering the threshold until there are no reflexive transitions in the graph is a good way to determine an initial suitable transition threshold. The

[1] http://graphics.ucmerced.edu/publications.html

segmentation also naturally prevents foot sliding effects during transition blending because all transition points are suitable for re-parenting the skeleton to the support foot.

We have also showed comparisons demonstrating the several improvements obtained by the channel pruning and the IK-based deformation technique (see Table 3.3). In all scenarios, these methods were able to improve both the quality of the synthesized solutions and the time taken during the search process. The channel pruning technique was able to significantly speed up the search without affecting the optimality of the solutions. The procedure eliminates the need for collision checking and guides the A* search by confining the search to the collision-free channel and, thus, avoiding getting stuck in local minima.

Figures 3.10, 3.11, 3.12 and Figure 3.14 show that FMGs present themselves as a better option for motion synthesis in comparison to SMGs since they are less likely to run out of leaves and terminate the search prematurely. This is due to the fact that FMGs create suitable transitions. While SMGs minimize the transition cost, FMGs consider transitions from pair of frames detected by meaningful feature detectors (without exceeding the same error threshold as used in SMGs). Selecting transitions in this manner also enables the A* search to further benefit from channel pruning.

The IK-based procedure represents a novel, simple, and effective way to optimize motions composed of concatenations of motion capture segments. If more deformation capability is needed, the IK deformation can be extended to consider every frame of a motion as a joint and to also include translational deformations. However, feet slide cleaning operations would be required after deformation in order to maintain the original quality of the motions. By only considering rotational deformation at transitions, a simple, clean and reasonably effective deformation method is achieved.

Overall, the presented evaluations demonstrate that our methods significantly improve the usability of motion graphs. The simplicity and motion quality obtained by motion graphs are excellent and difficult to surpass with other methods. Motion graphs do not require manual preparation of motions and can be built automatically from a single parameter defining the allowed error in transitions. Motion deformation is only needed to form transitions (with blending), and thus the overall distortion is minimum. This is also the case when using the proposed IK-based deformation since it was designed to always respect the overall allowed error threshold.

### Limitations and Avenues for Future Work

The main drawback of the proposed method is that relying on feature segmentation requires good feature detectors. If a feature detector is not adequate for a given set of motions, sufficient transitions may not be found. For example, the described walking feature detector will probably not be applicable to cartwheel motions, and if applied, it will probably lead to a motion graph with poor connectivity and, thus, not able to efficiently synthesize new motions. However, it is our experience that effective feature detectors can be developed with little effort, by determining where the most logical transition point should be. For instance, we have also used the presented walking feature detector to successfully segment different styles of walking and even ballet motions (see Figures 3.2 and  3.3).

An extensive collection of relevant feature detectors has been developed for the problem of motion comparison and retrieval [MRC05], and they could be well applicable to segment a variety of motions. We have not analyzed how additional feature detectors would impact the construction of a motion graph from generic motions. However, it is also possible to devise a hybrid approach where the presented robust locomotion features are responsible for segmenting locomotion clips, and the standard local minima metric is used to find generic segmentation points in areas not covered by the feature segmentation. The investigation of such a hybrid strategy is, however, left as future work.

Another point to observe is that FMGs lead to less nodes in the motion graph and this could lead to an increased response time for interactive character control, since less transition points are available for switching to a new motion when required. We have taken this possible implication into consideration when choosing our locomotion features. For motions involving locomotion, the character will most often face a choice only when one foot is planted on the floor, therefore, the feature segmentation should not affect interactivity. Similar observations were made by previous methods using manually crafted motion clips [TLP07].

More generally, FMGs are more compact because transitions are packed into "hub frames" that avoid over segmentation of nodes at many locations. The real aspect influencing interactivity would be the distance (in time) between nodes and not the number of nodes. Having poor interactivity performance would also imply poor search performance since less transition nodes would be available for finding solutions. FMGs have lower number of nodes but higher branching factors, the involved trade-offs were thoroughly examined in this chapter and the results demonstrate only benefits. This indicates that our method, at least for locomotion, should not loose interactivity.

In terms of performance, the computation time taken during search can still be significantly improved, possibly getting close to real-time performances, with the pre-computation of limited-horizon search trees, a method that has been already explored for manually-built move graphs [LK06]. In the next chapter we show that more structured segmentation of FMGs is well suited for achieving precomputed trees that can connect to each other even when built from an unstructured database.

## 3.10 Conclusion

We have presented new segmentation, search and deformation techniques for improving motion graphs. Our techniques significantly reduce the time spent for constructing the graph and at the same time lead to better solutions from search queries. We have demonstrated these benefits with several experiments in environments with obstacles, using both standard search procedures and the proposed channel pruning and IK-based motion deformation techniques. The proposed methods have showed to produce significantly improved results in all cases. The major result demonstrated in this chapter is that choosing transitions at local minima will not lead to optimal reachability or optimal search performance in the resulting graph. Instead, well-designed feature detectors will lead to improved motion graphs in several aspects, in particular with superior performances in search queries.

# CHAPTER 4

# Precomputed Motion Maps

In this chapter we present a solution for extracting high-quality motions from unstructured motion capture databases at interactive rates. The proposed solution is based on automatically-built motion graphs, and offers two key contributions. First, we show how precomputed expansion trees (or motion maps) coupled with new heuristics and backtracking techniques are able to significantly decrease the time taken to search for motions satisfying user constraints. Second, we show that when feature-based transitions are employed for constructing the underlying motion graph, the connectivity of motion maps is greatly increased, allowing the overall method to perform search and synthesis at interactive frame rates. We demonstrate the effectiveness of our approach with the problem of extracting path-following motions around obstacles from a motion graph structure at interactive performances.

## 4.1 Introduction

High quality motion search and synthesis from unstructured motion capture examples among obstacles has proven to be a challenging problem. Successful methods are often based on motion graph structures [KGP02, AF02, LCR02], which represent a popular approach for the purpose of character animation. Motion graphs have several good properties: they can be built automatically, they can be built to represent any kind of motion, and most importantly, they are able to produce realistic high fidelity results.

However, one of the main drawbacks of motion graphs is that they can easily become too large, preventing search algorithms from quickly finding motions that satisfy user constraints. Not only is the size of the graph a limiting factor, but also its connectivity since graphs with many transitions will have a higher branching factor eventually slowing down the underlying search algorithms. As such, motion graphs are mostly unable to support search and synthesis at interactive rates. For applications where interactivity is essential, the practical alternative often involves relying on a small hand-crafted set of motions with guaranteed transitions between themselves.

We present a solution for allowing motions to be extracted from a motion graph structure at interactive rates. We achieve this with the use of precomputed motion maps (see Figure 4.1) coupled with new heuristic and backtracking techniques

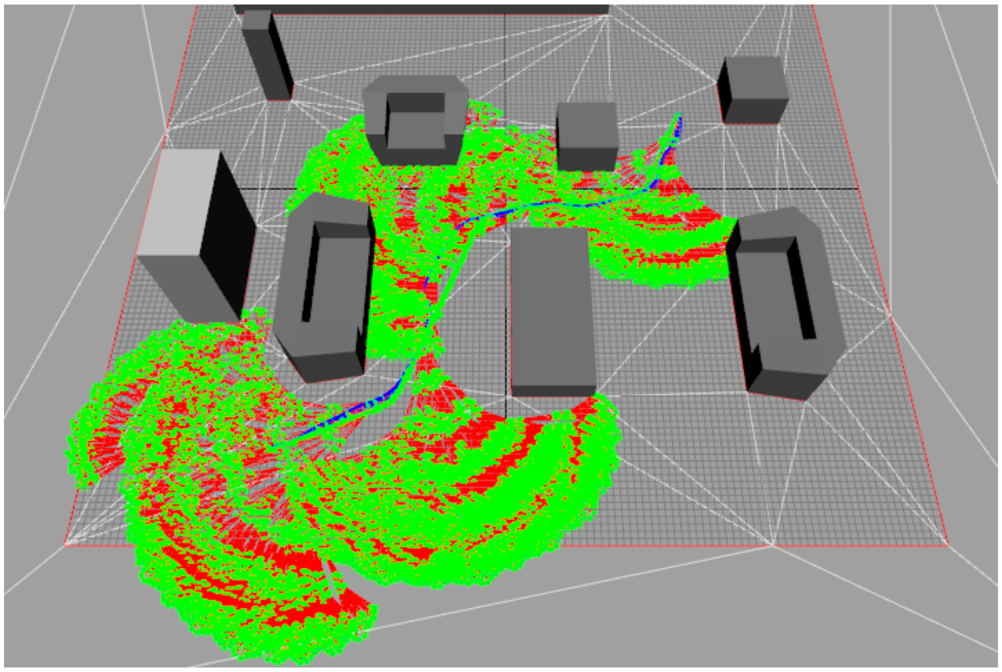

Figure 4.1: The image shows four motion maps used in a path following query. Motion maps are search tree expansions efficiently precomputed and stored; and then employed in run-time queries.

that significantly improve motion search performance. To eliminate the need of manually-crafted motion representations with full connectivity, we also show that a feature-based segmentation of the motion capture database [MK11] is able to produce motion maps with enough density and connectivity for the overall method to successfully search and synthesize motions at interactive frame rates.

In this chapter we focus on applying motion maps for solving the problem of navigation around obstacles. The obtained motions are realistic and are computed at interactive rates. Furthermore, due to the employed precomputed motion maps, motion search is often reduced to processing only the next best motion map, allowing search and synthesis to be performed in parallel.

Our overall method is divided in three main phases: motion map precomputation, path finding and path following. The precomputation phase is an off-line phase where the motion capture database is transformed into a feature-based motion graph (FMG) according to a locomotion feature segmentation [MK11], and motion maps are then computed for each node of the FMG. Given a query in run-time, the path finding phase will employ an efficient triangulation-based path planning method [Kal10] able to quickly return paths with guaranteed clearance from any given set of polygonal obstacles. The returned paths, therefore, provide channels with enough clearance for the character to move, minimizing collision checking queries after this point. In the path following phase, an optimized motion search algorithm is employed taking into account the precomputed motion maps, which are transformed and superimposed on the path during the search process. See Figure 4.1 for an example.

As a result, precomputed motion maps can be applied to efficiently solve path following queries from unstructured motion databases. Path following is one basic behavior important to several animation areas such as in simulation of populated environments and computer games. Our proposed solutions are automatic and applicable to generic locomotion data, and can therefore impact several of these applications.

## 4.2 Related Work

Motion graphs are built by connecting the frames of high similarity in a database of motion capture examples [KGP02, AF02, LCR02, PB02, LWS02]. Once the motion graph is available, a graph search is performed in order to extract motions with desired properties.

Many methods based on motion graphs have then been proposed. Kovar et al. [KGP02] used branch and bound to find motions that follow a user specified path by considering the motion synthesis problem as an optimization problem. Arikan and Forsyth [AF02] used a randomized method to extract motions from a hierarchy of motion graphs. Lee et al. [LCR02] constructed a cluster forest of similar frames in order to improve the motion search efficiency. Dynamic programming was used by Arikan et al. [AFO03] to search for motions satisfying user annotations (such as first *run* and then *jump*).

The approach of Safonova and Hodgins [SH07] and Zhao and Safonova [ZS08] is based on an *interpolated motion graph* with *anytime A** used to search for solutions. The resulting motion is an interpolation of two time-scaled paths through the motion graph. Although it represents an improvement in respect to satisfying constraints, this approach requires additional search time for finding solutions. The difficulty to run at interactive frame rates is a common limitation of motion graph approaches.

Many other planning methods have been proposed for synthesizing full-body motions among obstacles [EAP06, PZL10, KL00a]. In particular, Choi et al. [CLS02] combine motion capture data with probabilistic roadmaps [KSL96] to generate motions for a given start and goal positions. Although such methods improve the planning capabilities for addressing constraints, the quality of the results are not improved in respect to motion graph approaches. Sung et al. [SKG05] make use of probabilistic roadmaps to guide a bidirectional search, achieving realistic results but relying on unrolling a manually-crafted motion graph structure.

Structures based on motion graphs still carry the benefit of minimally deforming the original database of collected motions, therefore achieving high quality results. In general, the main drawback of motion graphs is that a prohibitively large structure would be needed in order to produce motions satisfying many constraints, such as around obstacles and addressing precise goal locations. The problem of increasing the motion database is that, as the size of the graph grows, the underlying search methods will require additional computation time for finding solutions.

This inherent difficulty of motion graph structures is well-known and methods based on simplifying the database have also been proposed. In particular, Gleicher et al. [GSK03] note that one main difficulty of motion graphs is its unstructured nature, and they propose a method to simplify the graph to a small structured graph suitable to interactive control. In the same direction, fat graphs [SO06] and parametric graphs [HG07] have been proposed as attempts to improve the structure of the motion capture data so that interactive controllers can be devised.

The approach taken in this chapter seeks to fully handle the entire given motion capture database, and to use precomputed search trees in order to achieve interactive performances. Precomputation of search expansions has been already employed for the problem of motion synthesis using motion capture data. For instance, the work of Lau and Kuffner [LK06, LK10] efficiently employs precomputed search trees for synthesizing goal-driven interactive motions. However, in order to achieve efficiency, their approach is designed around a manually-built finite state machine of motions [LK05] that is fully connected. In contrast, the specific techniques we propose enable precomputation to be applied to unstructured motion graphs.

Srinivasan et al. [SMM05] apply precomputed trees for all nodes of a motion graph, and report difficulties handling tight spaces around obstacles due the standard search expansion technique employed. The related approach of precomputed avatar behavior policies has also been proposed by Lee at al. [LL04].

Our approach is most similar to these works, however, we focus on handling unstructured motions around obstacles at interactive rates. Our method introduces several benefits: 1) it works with generic automatically built Feature-based Motion Graphs (FMGs) [MK11] and hence does not require manually-built finite state machines of motions; 2) it precomputes expansion trees for all nodes in the FMG and does not require full connectivity, i.e. it does not require all leaves of a precomputed search tree to link to the root of the tree for seamless transitions; 3) it employs a fast triangulation-based path computation [Kal10] that computes paths within a collision free corridor with given clearance, allowing the search to be pruned to the corridor and minimizing collision detection queries during the search; and 4) it employs a search strategy that considers backtracking for well handling difficult situations involving tight spaces, turns and precise arrivals; achieving excellent overall performance due the improved search and the well-formed FMGs.

The proposed method therefore well handles unstructured motions around obstacles and is able to produce long paths efficiently even in complicated environments with many obstacles. The method is suitable to interactive applications and the results are always of high quality.

## 4.3 Finding Paths with Clearance

Given an initial point $p_{init}$, a goal point $p_{goal}$, and a clearance distance $r$, the collision-free path $P(p_{init}, p_{goal}, r) = (p_0, p_1, \ldots, p_n)$ is described as a polygonal line with vertices $p_i$, $i \in \{1, \ldots, n\}$ describing the solution path. As the path turns around obstacles with distance $r$ from the obstacles, every corner of the path is a circle arc which is approximated by points, making sure that the polygonal approximation remains of $r$ clearance from the obstacles.

Any path planning method with clearance can potentially be used to compute $P(p_{init}, p_{goal}, r)$. Due its efficiency, we employ the Local Clearance Triangulation (LCT) method [Kal10], which leverages 2D meshing algorithms for maintaining a suitable triangulation of the free space for path planning with clearance. We are using an extended version of the method that includes dynamic obstacle updates only requiring local updates each time an obstacle changes position [Kal10]. We can therefore handle dynamic environments very easily, with updates and path queries in relatively complex environments being computed in the order of milliseconds. Figure 4.2 shows examples of collision-free paths with clearance obtained by the method.

Given that path determination takes obstacle clearance into account while solving at the path finding level, we almost entirely eliminate the need of using costly collision checking queries during the FMG search and synthesis.

In our path following application the obtained 2D path is used for guiding and pruning the FMG search and a fine polygonal approximation of the curved sections of the path is not needed. We therefore approximate the curved sections very coarsely with only a few points.

## 4.4 Precomputation of Motion Maps

We start by constructing a motion graph similarly to Mahmudi et al. [MK11]. Although, standard motion graphs might very well be used with our path following methods, our results indicate that FMGs have higher success rates because they are better at generating denser and more evenly distributed motion maps (see Figures 4.3 and 4.4). As later shown in Section 4.5, good density and distribution in motion maps are important requirements for the path following procedure to run successfully.

The next step is to precompute motion maps for all the nodes of the FMG. The rationale behind the precomputation is to be able to follow the input 2D path

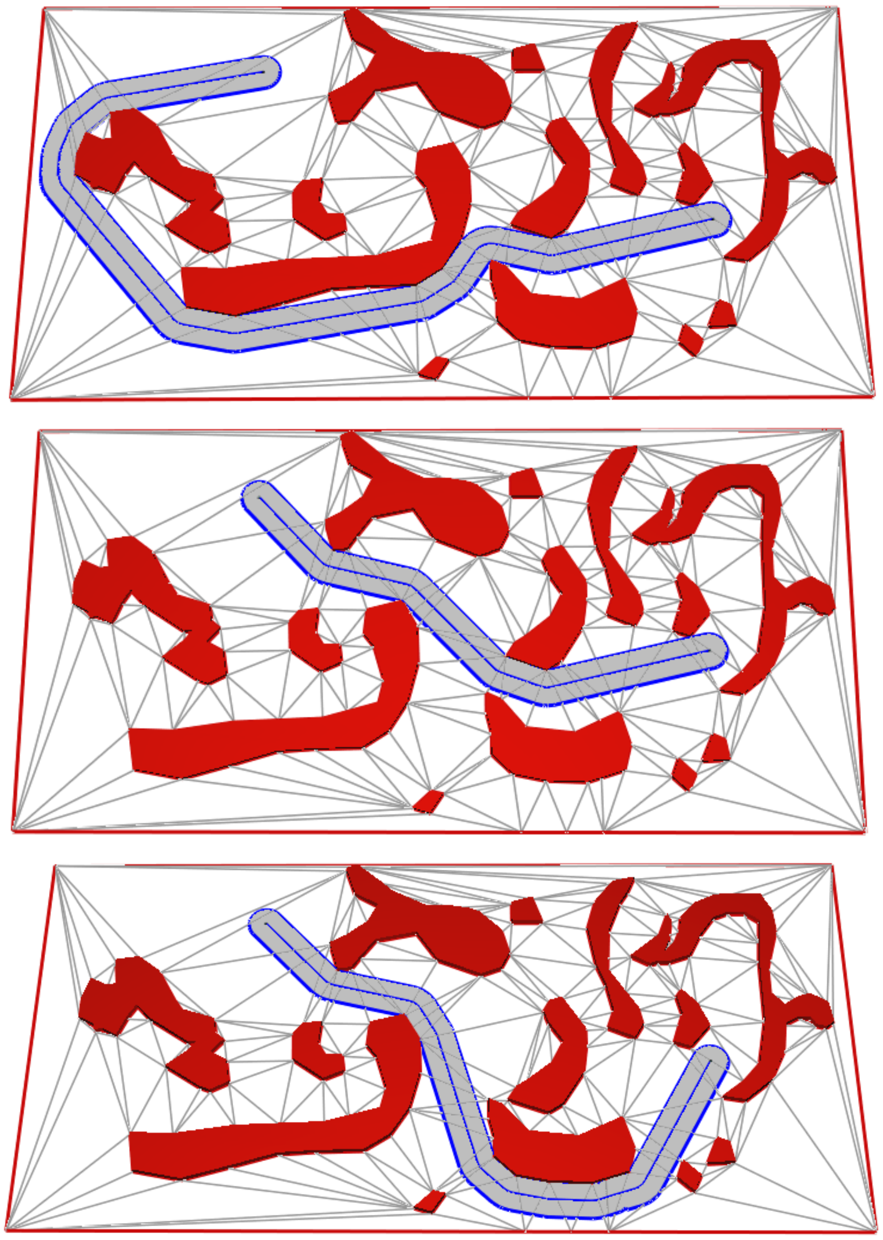

Figure 4.2: Different paths obtained to connect the same initial and goal points, as they adapt to a few changes in the obstacles. The clearance of the paths is always maintained, and the LCT representation is updated only with local operations for each time an obstacle moves.

by repeatedly making efficient queries to motion maps for partial solutions that can follow the path closely. This process is repeated until the goal is found or all the possible candidates are exhausted. Unlike previous methods that precompute only one expansion tree, we precompute motion maps for all the nodes of the motion graph. The benefit of precomputing all the nodes is that we do not have to rely on manually crafted motion graphs with full connectivity but can instead use any automatically-built motion graph.

Let $n$ represent a node of the FMG, which in our representation contains one segmented motion clip. A motion map $T_n$ of a node $n$ represents a tree $T_n$ of motions that can be generated starting from the node $n$. The motion maps span a three dimensional space $X$ defined as:

$$X = \{(x, z, \theta) \in \mathbb{R} \times \mathbb{R} \times (-\pi, \pi]\}\,. \tag{4.1}$$

The $x$ and $z$ parameters determine the position of the character on the floor and the $\theta$ specifies the orientation about the Y vertical axis of the root joint of the skeleton; the positive Y axis points up on the XZ plane. For efficient storage, we discretize the motion maps into cells of 10 cm by 10 cm for the $x$ and $z$ parameters and into 12 groups of 30 degrees for the $\theta$ orientation. Furthermore, we represent motion maps as hashed maps instead of 3D grids. There are two reasons for this: first, by using hashed maps we avoid setting spatial limits on how far our motion maps can be unrolled; second, motion maps populate $X$ sparsely and unevenly and, as such, a significant number of cells in our discretization remain empty. Hash maps will only store occupied cells and, therefore, no space will be allocated for unoccupied cells.

Algorithm 1 illustrates the overall precomputation process. We start by *unrolling* each node $n$ of the FMG. The procedure starts by placing the first frame of the node $n$ at the origin, with the character facing the positive Z axis, and building a motion map $T_n$ with its root set to the node $n$ (lines 1-3). Then, we run a Dijkstra search starting on the node $n$ and expand nodes that minimize the arc-distance traveled by the expanded motions (lines 15-17). At every expansion, the motion map $T_n$ is augmented with the position $(x_i, z_i)$ and orientation $\theta_i$ of the character and the corresponding cell $T_n(x_i, z_i, \theta_i)$ is annotated with the path $P_i$ that leads the character from the initial node $n$ to the current state $(x_i, z_i, \theta_i)$ (line 14).

Note that since the same cell might be reached by different paths we store all possible paths at each stage. This increases the number of available candidates during the path following search phase. The unrolling stops when the Dijkstra search reaches a user-defined depth (line 6). Examples of typical precomputed motion maps obtained for FMGs are shown in Figure 4.3. As comparison, Figure 4.4 shows the obtained motion maps for a standard motion graph built from the same motion capture database. It is possible to note that the well-defined segmentation rule of FMGs lead to much denser motion maps, an essential property

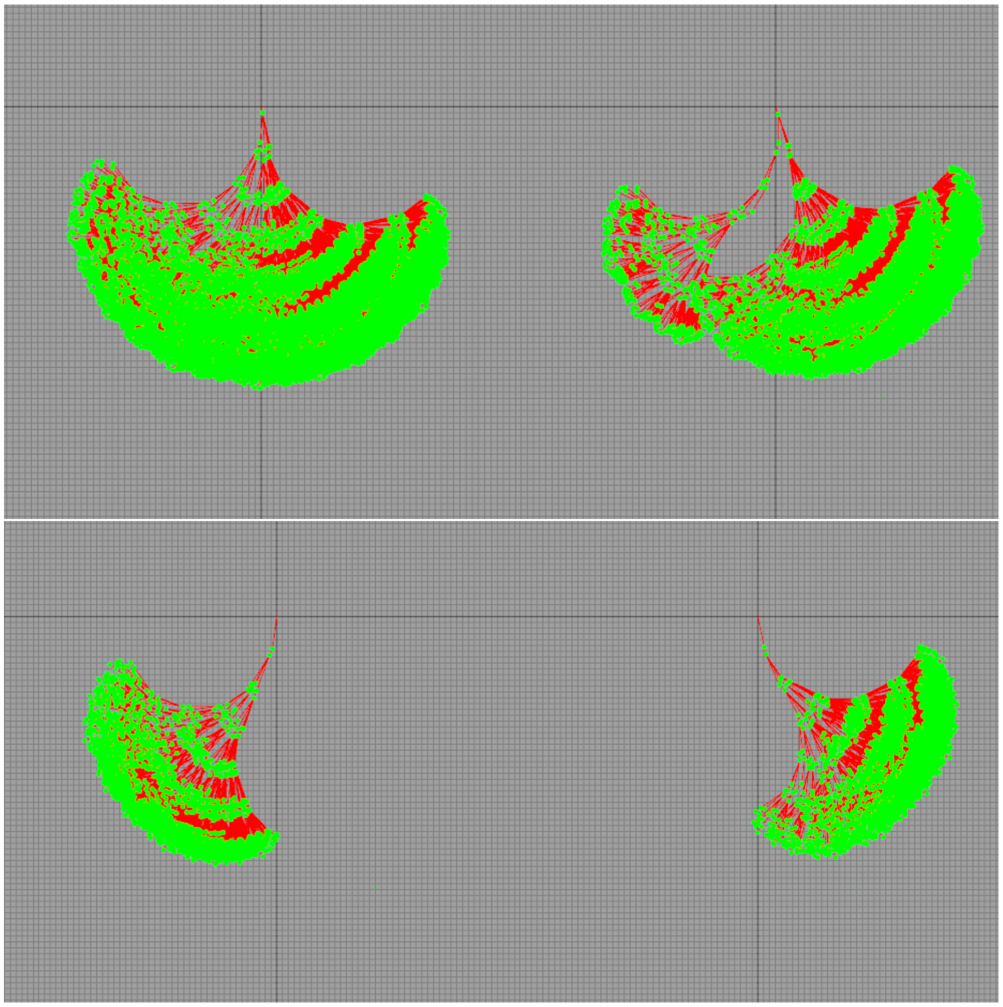

Figure 4.3: Four typical motion maps precomputed for a Feature-Based Motion Graph (FMG). The used expansion depth is 12. Note the high density achieved in the regions covered by the motion maps.

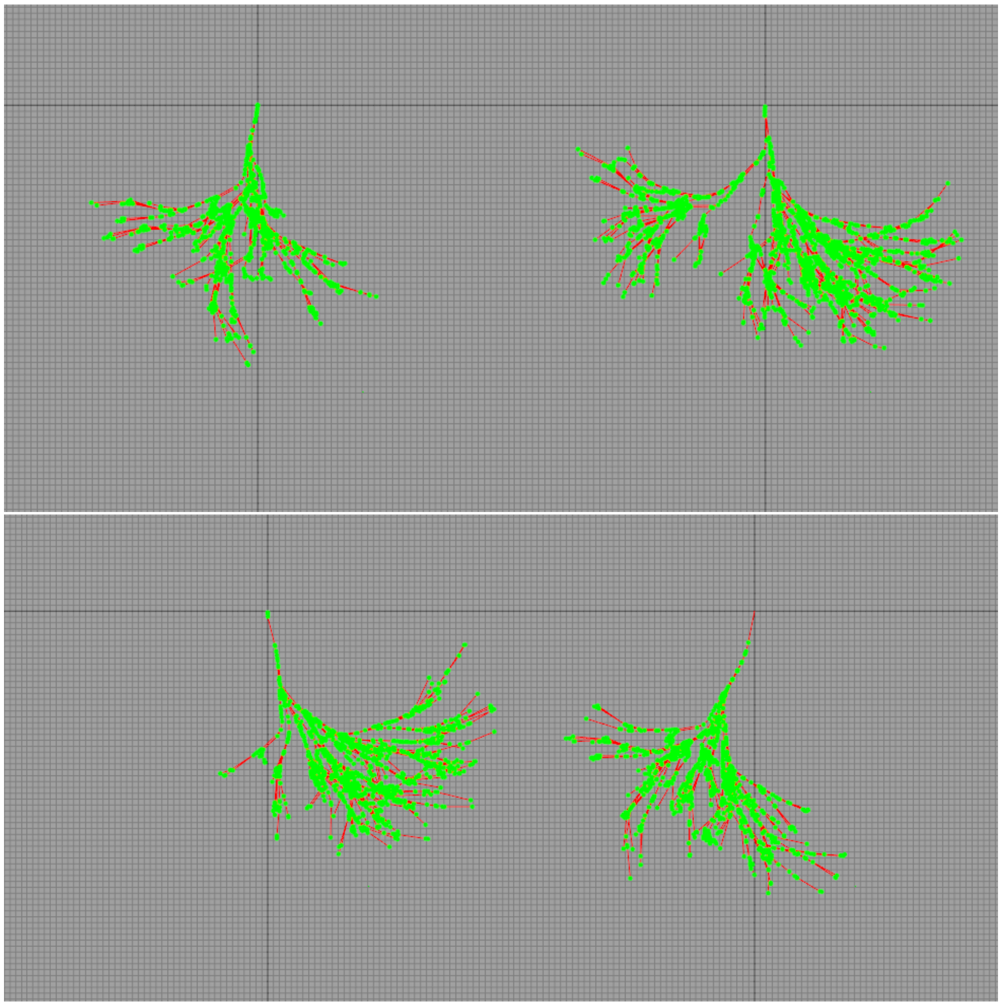

Figure 4.4: The four equivalent motion maps to the ones shown in Figure 4.3, when precomputed for a standard Motion Graph (SMG). These maps were precomputed with depth 22 in order to achieve a size (number of nodes) equivalent to the FMG motion maps.

**Algorithm 1** Precompute($n, max\_depth$)

```
1: F.init();
2: Q.init($n$);
3: T.init($n$);
4: while (Q not empty) do
5:     node $\leftarrow$ Q.pop();
6:     if (node.depth $\geq$ max_depth) then
7:         continue;
8:     end if
9:     if (F(node) is not occupied) then
10:        F.occupy(node);
11:    else
12:        continue;
13:    end if
14:    T.insert(node, path(node));
15:    node.expand();
16:    for all (child $\in$ children(node)) do
17:        Q.push(child, child.length);
18:    end for
19: end while
20: return T;
```

to guarantee that solutions are mostly always found. Additional comparisons are discussed in Section 4.7.

Each motion map $T_n$ also stores the transformation $\Phi$ that aligns the map to the origin with the positive Z axis direction and the average length of a motion map, which is defined as:

$$\mathcal{A}(T_n) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(T_n^i), \tag{4.2}$$

$$\mathcal{L}(m) = \sum_{i=0}^{n-1} ||m_i - m_{i+1}||, \tag{4.3}$$

where $T_n^i$ is the $i$-th path of $T_n$ and $\mathcal{L}(P)$ returns the length of the 2D path $P$. The average length of $T_n$ will determine the sampling frequency of the path $P$ during the path following phase (Section 4.5).

Moreover, to speed up the precomputation, we also maintain a 4D frontier $F = \{(x, z, \theta, n_{id}) \in X \times \mathbb{N}\}$, which prevents expanding duplicate branches (lines 9-10). If the same node $n_{id}$ is about to revisit a cell $c \in X$ then the we can safely cull this branch as expanding $n_{id}$ at $c$ would not lead to any new paths. Once the motion map is built, the frontier $F$ is no more needed and it is discarded. Another significant speed up is to cache cells that were already queried. This facilitates the search during the path following phase since many queries are made to cells that were already used in prior queries.

The computational complexity of the precomputation procedure is $O(nb^d)$, where $n$ is the number of nodes, $b$ is the average branching factor of the motion graph and $d$ is the chosen horizon depth of the motion map. Depth $d$ is a significant factor influencing the precomputation time, however, we have noticed that after a certain depth is reached, increasing the horizon depth does not translate into further improvements to our path following procedure.

## 4.5 Path Following

As described in Section 4.3, the input path $P$ is a collision-free path with clearance $r$, with starting point $p_0$ and goal point $p_{goal} = p_n$. The start and goal orientations are also defined by the input path. The start orientation is determined by the vector $p_1 - p_0$ and the goal orientation is determined by the vector $p_n - p_{n-1}$, where $p_i \in P(p_0, p_n, r)$ are points from the polygonal path as returned by the LCT path planner.

The path following procedure assumes that all the nodes of the FMG are precomputed up to a depth $d$ and all their paths are stored in their corresponding motion maps. After the precomputation is finished, the goal of the path following procedure is to search for a sequence of nodes that generates a smooth motion closely following the collision-free input path $P(p_0, p_n, r)$.

The search procedure is presented in Algorithm 2, and all line references that follow are in respect to this algorithm. The search starts by selecting an initial node $i$ from the FMG as the starting node of the search, and aligning its precomputed motion map $T_i$ with the input path $P$. The alignment is determined by the transformation $\Phi_i$ associated with the motion map $T_i$ and a transformation $\Psi_j$ as determined by the last position and orientation of the partial solution up to the current point. Initially, $\Psi_0$ is set to the start position and orientation of the path $P$. Subsequent alignments are performed in a similar manner by concatenating the accumulated transformation $\Theta$ with the product of $\Phi_i$ and $\Psi_j$, as defined by the partial solution of the $j$-th iteration of the algorithm:

$$\Theta_j = \Theta_{j-1} \Phi_i \Psi_j. \tag{4.4}$$

The next step is to sample path $P$ to obtain query points $q \in X$ which are a subset of the space $X$ (line 3). The query points are equally spaced between the beginning of the residual input path up to the point with length along the path equal to the average length of $T_i$, as defined in Equation 4.2. In our experiments most points were about 10 cm apart, see Figure 4.9 for an example. Note that

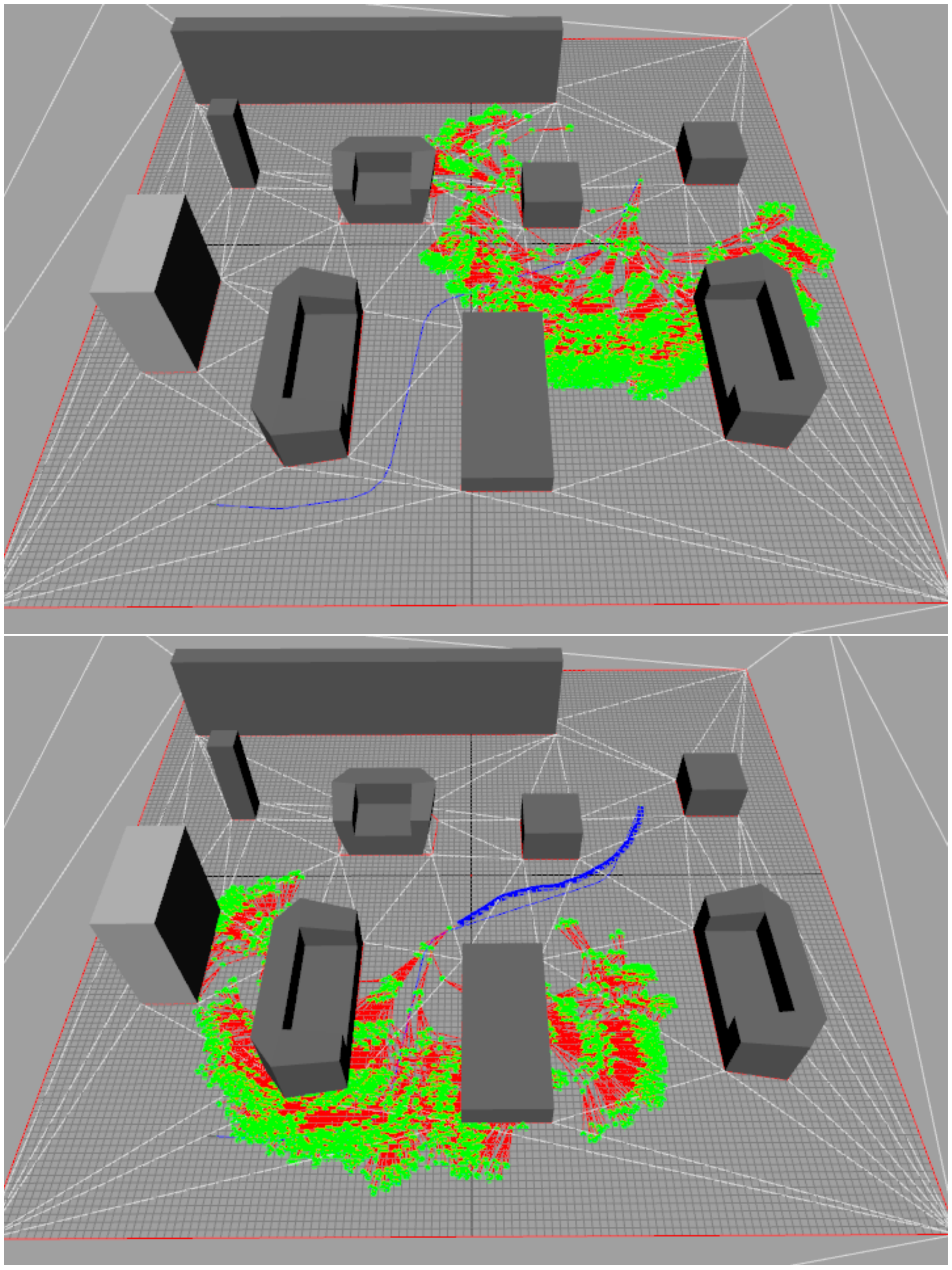

Figure 4.5: Two different motion maps while generating the happy walking motion showed in Figure 4.6.

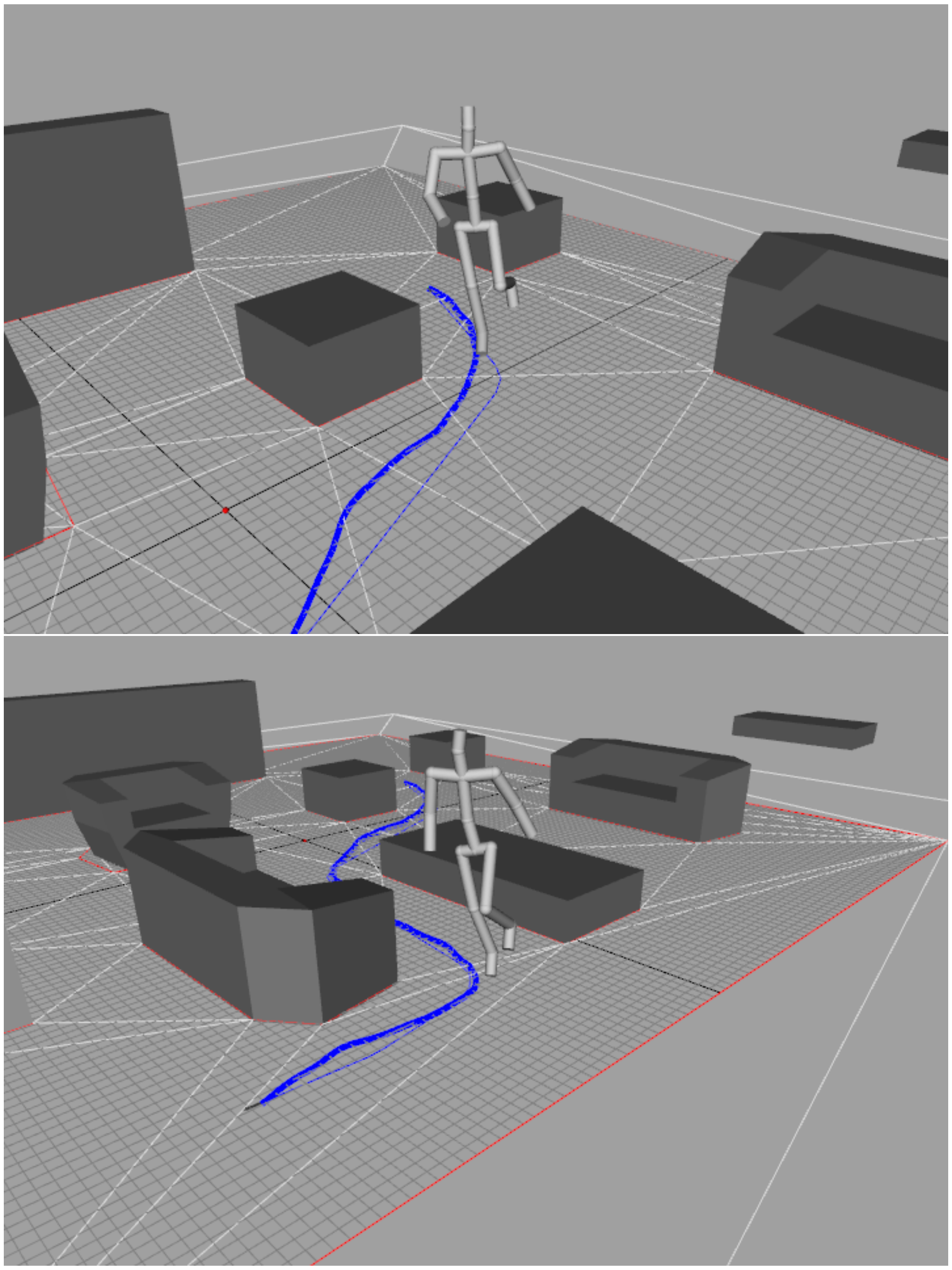

Figure 4.6: The resulting happy walking motions from the search done in Figure 4.5.

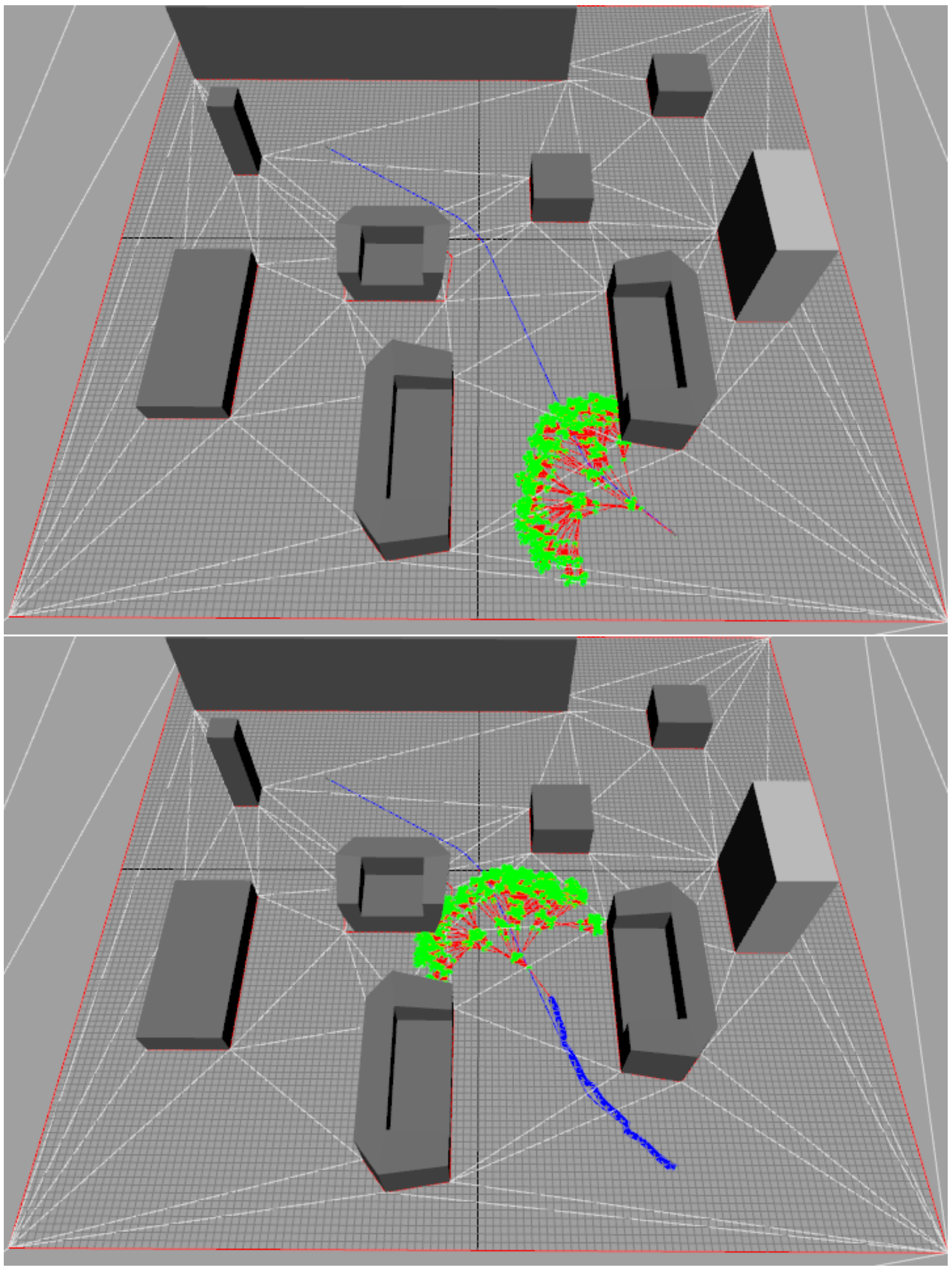

Figure 4.7: Two different motion maps while generating the ballet motion depicted in Figure 4.8.

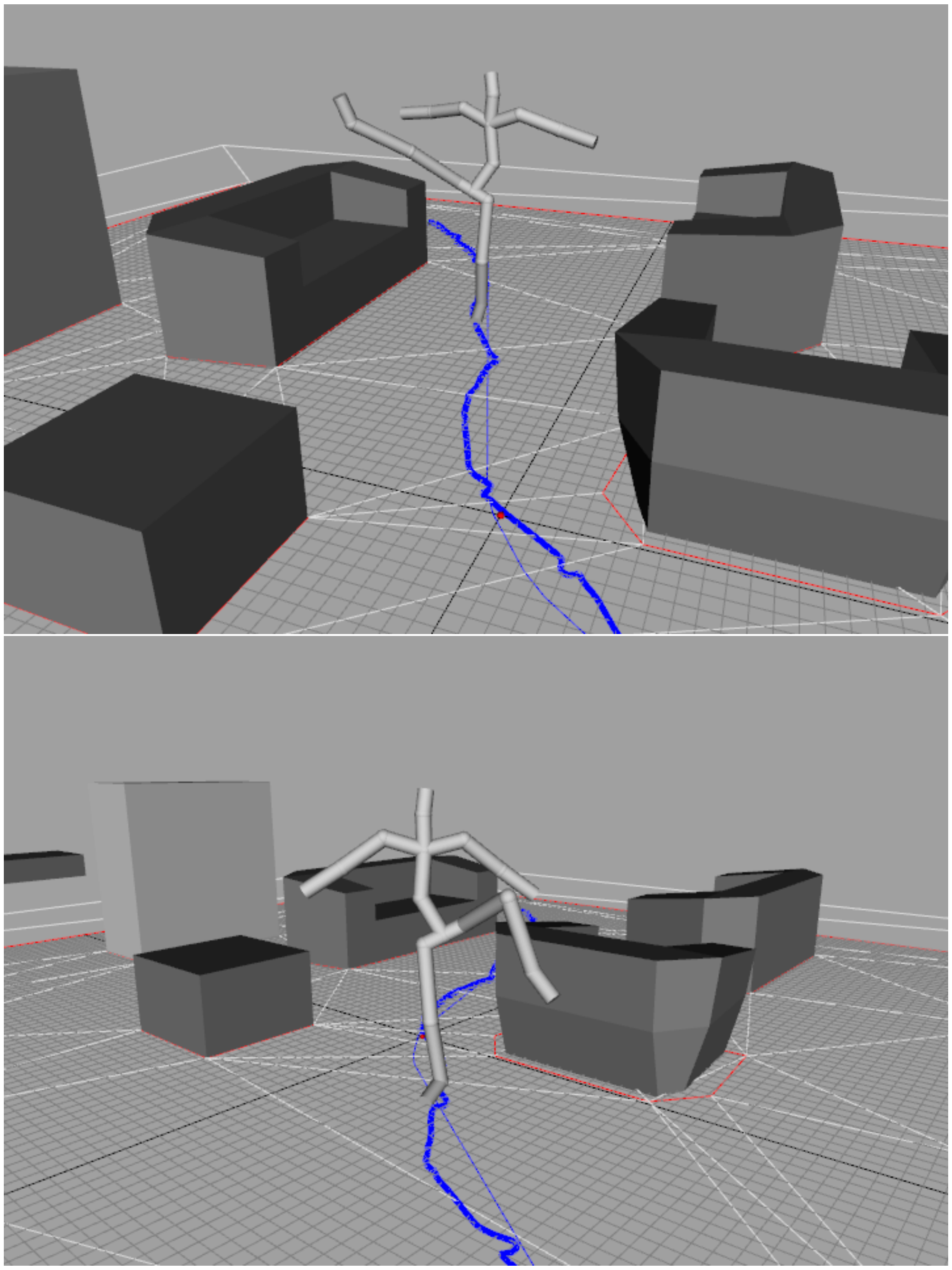

Figure 4.8: The resulting ballet motions from the search performed in Figure 4.7.

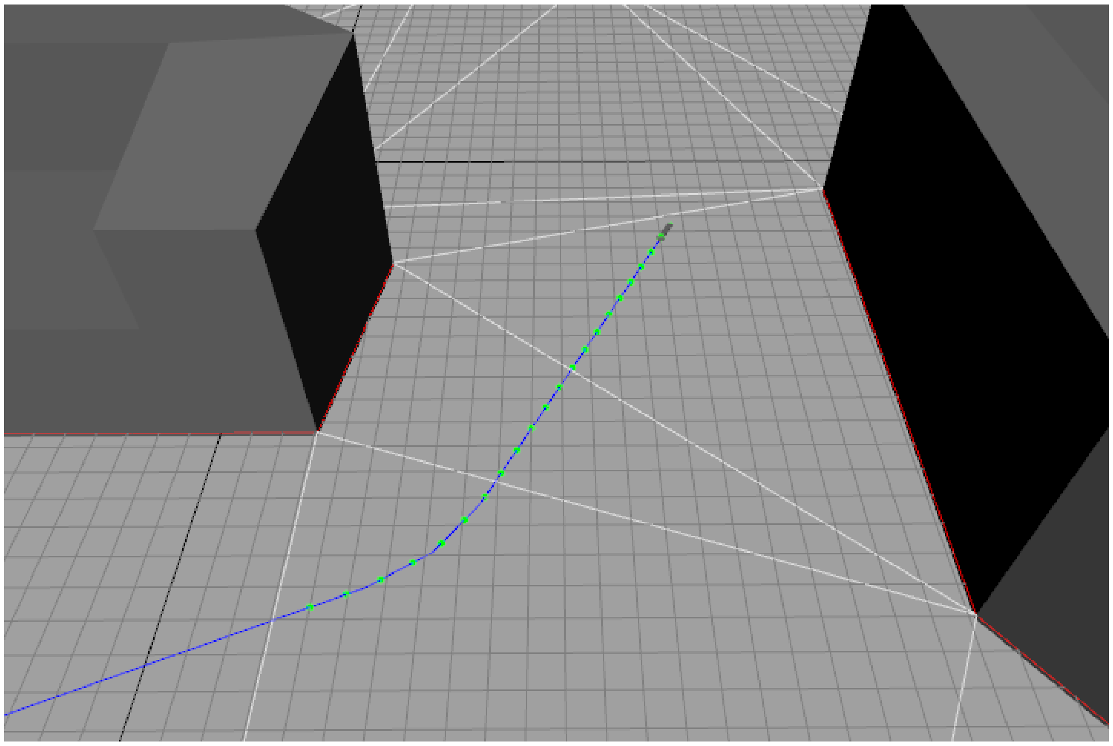

Figure 4.9: Sample points from the input path used during the path following search.

the sampled points are in world coordinates and before querying the motion maps they have to be transformed to the motion map's local frame. This is done by transforming the query points $q_i \in Q$ by the $\Theta_j^{-1}$ transformation.

At every iteration of the path following procedure, we first query for the goal $p_n$ and then all the query points $\Theta^{-1}p_i$ against $T_i$ to check for possibles partial paths (line 4). If none of the queried cells in the motion map are occupied, the algorithm backtracks to consider different candidates from its previous iteration. If during backtracking the root node is reached and all its alternatives are exhausted then the algorithm stops and reports failure (lines 5-9). When the goal query is successful then the final partial path is appended to the solution and success is reported (lines 20-21), otherwise, all the queried candidate paths are retrieved and sorted for their suitability (line 12).

The sorting function that picks the best partial path among the available candidates could be tuned to suit a particular application. Since our goal is to follow the path as closely as possible, we have defined our sorting function to compute the area formed between the candidate partial path $m$ and the input path $P$. However, since longer paths will lead to a larger area, we divide the cost by the square of the length of the candidate path to avoid favoring shorter paths.

The sorting function $f(m, M, P)$, is shown in Equation 4.5. It takes three

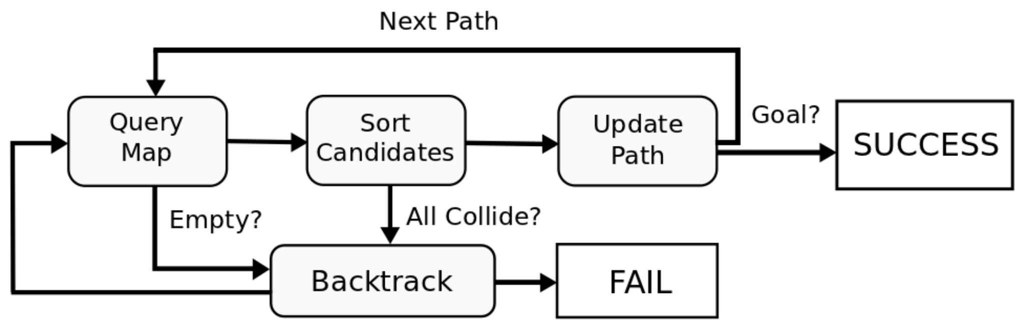

Figure 4.10: Main stages of the path following search procedure.

parameters: a candidate path $m$, the current partial solution $M$ and the input path $P$. The $\mathcal{L}(m)$ function computes the length of a path $m$ (if an index of a frame is indicated then it computes the length up to that frame). Function $\mathcal{P}(P, l)$ returns a point $p \in P$ at distance $l$ along the input path $P$. These functions read as follows:

$$f(m, M, P) = \frac{\sum_{i=0}^{n} ||m_i,\ \mathcal{P}(P,\ \mathcal{L}(M) + \mathcal{L}(m_i)))||}{\mathcal{L}(m)^2}, \tag{4.5}$$

$$\mathcal{P}(P, l) = \left\{(x, z) \in \mathbb{R}^2 \mid 0 \leq l \leq \mathcal{L}(P)\right\}. \tag{4.6}$$

Once all the candidate paths are sorted, we iterate through the sorted candidate paths and pick the best path $m_{best}$ (line 13). Once the best candidate path $m_{best}$ is chosen, we run collision checking on $m_{best}$ to determine if it is collision-free; otherwise, we consider the next best candidate. Once such candidate path is found, it gets appended to the current partial solution $M$ (line 17) and the last node of the best path $m_{best}$ determines which motion map gets used in the next iteration of the path following procedure (line 19). At this stage, the partial solution path $M$ is a motion that follows the input path $P$ up to the point $p_{best}$, with an orientation facing the residual input path $P$. In case all the sorted candidates are unsuitable, the procedure backtracks and resumes considering other alternatives from the previous iteration (lines 14-15). The path following search iterates through this procedure until either the goal is reached (lines 20-21) or the lengths of the partial solutions $M$ exceed the length of the input path $P$. Figure 4.10 overviews the main stages of the overall algorithm.

As the path following advances towards the goal, a stack of sorted candidates are stored at different stages. When backtracking needs to occur, the failed stage is popped from the stack and the next best candidates left from the previous

**Algorithm 2** FollowPath($T, P, i$)

```
1:  $i_{last} \leftarrow \emptyset$
2:  while $\mathcal{L}(M) \leq \mathcal{L}(P)$ do
3:      $Q \leftarrow$ sample_path($P$);
4:      $C \leftarrow$ query_map($T_i$, $Q$);
5:      if $C$ is empty then
6:          if $i_{last} == \emptyset$ then
7:              return FAIL;
8:          else
9:              $i \leftarrow i_{last}$;
10:         end if
11:     else
12:         $S \leftarrow$ sort_paths($C$);
13:         $m \leftarrow$ best_path($S$);
14:         if $m ==$ NONE then
15:             $i \leftarrow i_{last}$;
16:         else
17:             $M$.append($m$);
18:             $i_{last} \leftarrow i$;
19:             $i \leftarrow m$.last();
20:             if $M$.goal_reached() then
21:                 return $M$;
22:             end if
23:         end if
24:     end if
25: end while
```

stages are reconsidered without having to re-query the motions maps associated with the previous stage. Storing partial paths in stages offers a significant speed up and allows for efficient reuse of prior partial path. In this manner, as a whole, our motion search algorithm can be seen as an efficient cached A* search with very effective pruning by the 2D channel computed from the triangulation planner.

## 4.6 Concurrent Motion Synthesis

Since the time spent on searching is significantly lower than the duration of the synthesized motions, we can search and play motions concurrently. When a query is made to the path following procedure, the first node to be used is either specified by the user or automatically chosen by the procedure. Since this first node is known in advance, and is not subject to change for a particular query, our method can immediately start the path following search procedure while the first node is being played. While the first node is finished playing, the path following is

| Depth | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time | 8.14s | 11.58s | 16.36s | 22.93s | 31.92s | 44.23s | 1m 0s | 1m 23s | 1m 55s | 2m 38s |
| Cells | 12141 | 15361 | 19351 | 24367 | 30521 | 38391 | 48193 | 60356 | 75506 | 93992 |
| Size | 42MB | 59MB | 83MB | 116MB | 162M | 225M | 311MB | 428MB | 588MB | 805MB |

Table 4.1: Motion maps with various depths in a SMG: preprocessing time, average number of occupied cells, and average size.

| Depth | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time | 0.09s | 0.19s | 0.57s | 1.21s | 3.26s | 6.76s | 17.39s | 35.38s | 1m 28s | 3m 4s |
| Cells | 360 | 667 | 1396 | 2458 | 4956 | 8312 | 16173 | 25552 | 47413 | 70117 |
| Size | 0.39MB | 0.9MB | 3MB | 7MB | 17M | 35MB | 88M | 179M | 444MB | 889MB |

Table 4.2: Motion maps with various depths in a FMG: preprocessing time, average number of occupied cells, and average size.

already way ahead of the nodes that need to be played next. Usually the entire path following search concludes before the first few nodes are played.

In order to achieve concurrent playing and search we have two separate threads: the search thread and the rendering thread. The search thread is assigned the task of carrying the path following search procedure and the rendering thread renders the obtained partial solutions. The rendering thread initially awaits a signal from the search thread notifying the completion of the first partial path. This is usually done almost instantaneously. Then, after finishing rendering the first partial path, the rendering thread continuously polls the search thread for new partial paths until the goal is reached. The rendering threads tries to achieve a fixed frame rate of 30 frames per second. After the visualization buffer is updated the rendering thread is blocked and the path following searched is resumed until the next rendering cycle. The parallelization has shown to scale well and we have successfully run about 10 characters searching and playing motions at 30 frames per second in a quad-core CPU.

Due to the backtracking possibility of our path following search procedure, the rendering thread might start rendering a partial path that has been backtracked in the search thread. In that event, the rendering thread stops playing the motion and reports a failure. This does not happen very often as the path following search is very efficient, however, it may happen if the motion database lacks motions suitable for navigating in all areas of the environment. The user might always decide to improve the database, or to disable simultaneous search and animation in order to maximize the use of the backtracking mechanism.

## 4.7 Results and Discussion

For our experimental setup we have built three FMGs from different types of motions: regular walking database, happy walking database and one ballet motion database. Each of these motion capture databases contained 1 straight motion, 1 left sharp turning motion, 1 left gentle turning, 1 right sharp turning motion and 1 right gentle turning. Our method showed to work well with this relatively small number of turn variations. Additional variations would improve the already high success rate of the algorithm, and the only drawback would be additional storage space for the precomputed motion maps. The motions were sampled at 60Hz using 18 Vicon cameras in an environment of 5.5 m by 4 m. The total number of frames in the motion capture databases ranged from 1079 to 3181, corresponding to 20-60s of motions. The motions were captured and processed into the FMGs without any manual editing involved. All the measurements took place on a 2.7 GHz Intel i7 computer with 4GB of memory.

We built the FMGs as described in [MK11], and for comparison we also built a standard Motion Graph (SMG) as described in [KGP02]. For achieving adequate comparisons we have used the same error threshold for deciding transitions in both

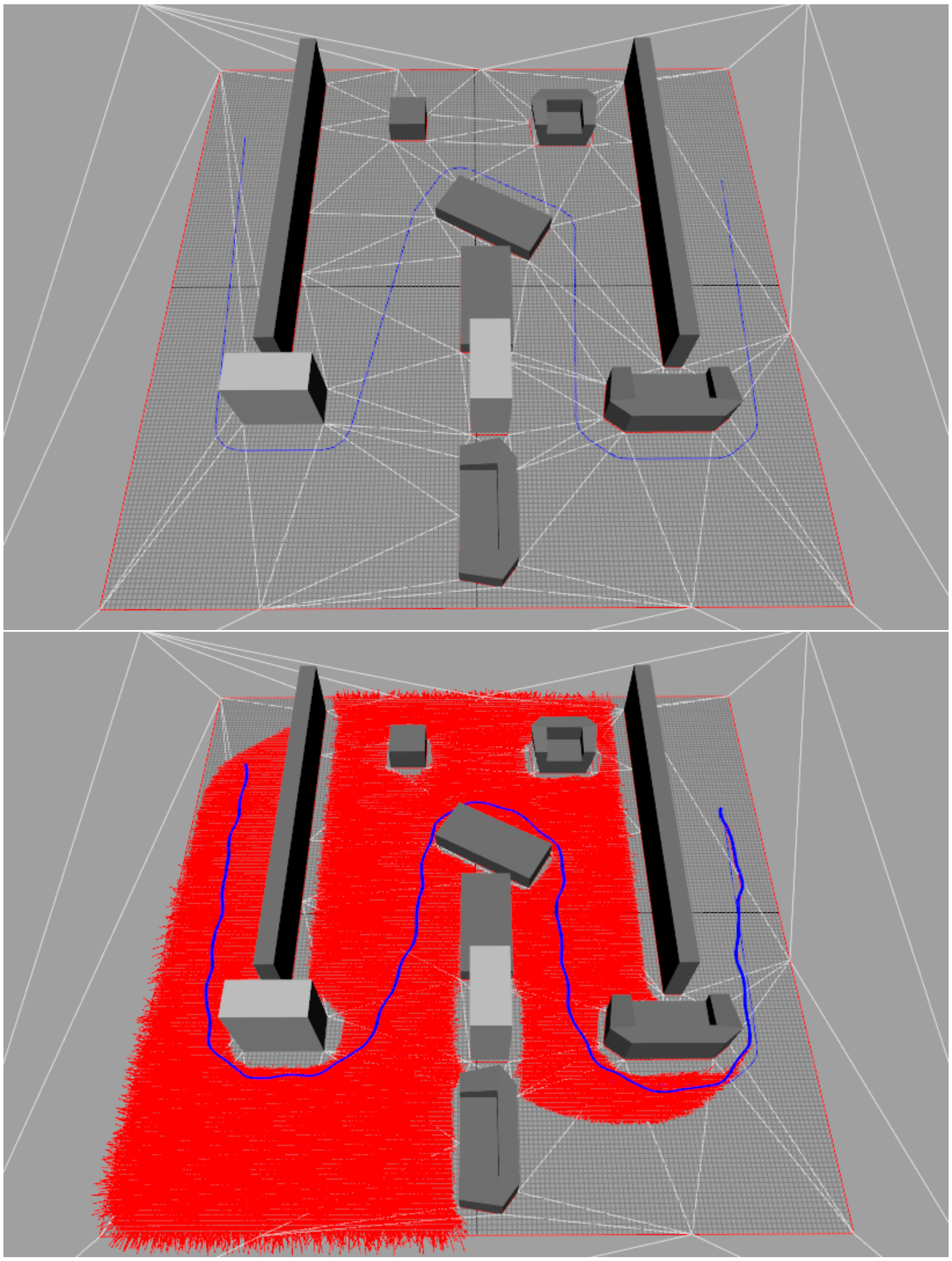

Figure 4.11: Comparison of search techniques. The top image shows the used environment and its blue path is the input path as returned by the LCT planner. The bottom image depicts the search tree expansion of A* and our proposed method based on motion maps. The search took 186s to complete.

graphs. In our trials, for the regular walking database, we have set the maximum error threshold to 6 cm, obtaining a FMG with 72 nodes and average branching factor of 1.56, and a SMG with 277 nodes and an average branching factor of 1.30. Similar graphs were obtained for the happy walking and ballet motions.

We then precomputed motion maps for all the nodes of both FMG and SMG, and at various depths. As mentioned earlier, motion maps were discretized into cells of 10 cm by 10 cm for the $(x, z)$ positional parameters and into 12 cells of 30 degrees for the orientation parameter. Table 4.1 (SMG) and Table 4.2 (FMG) show the time taken to generate all the motion maps. Also reported is the total number of occupied cells and the combined size of all the motion maps.

As it can be seen from the Tables 4.1 and 4.2, SMGs require higher depths, in comparison to FMGs, to occupy motion maps with the same amount of entries. The reason for this is the fact that the average length of a node in SMG is shorter than the average length of a node in FMG. In Figures 4.3 and 4.4 we show motion maps precomputed from a FMG (of depth 12) and from a SMG (of depth 22) and notice that FMGs are significantly better at evenly spanning the covered space of $X$ and thus they present themselves as a better choice for the path following algorithm. They offer a wider range of different candidate motions and ensure global connectivity (and concatenation success) during the path following search. This FMG property is essential for successful execution of the path following method.

We also measured the importance of the motion map depth and size in respect to the effectiveness of the path following algorithm by running 1000 trials on the environment shown on Figure 4.12. Each trial consisted of: randomly selection of start and goal locations on the environment, query to the LCT planner for a path $P$ with clearance of 50 cm, and full execution of the path following search procedure in respect to the input path $P$. The same trials were conducted for both the FMG and SMG, and the results are shown in Tables 4.3 and 4.4.

Table 4.3 reports the results for FMG. We can notice that the size of motion maps initially had a strong influence on the success rate and search time of the path following method, however, once a certain horizon was reached larger motions maps did not improve the path following method as effectively. We also notice that the lengths of the solution motions returned by the path following method were very close to the length of the input path, showing that our method is capable of generating solutions that are very close to optimal solutions. This is as expected since we sample the input path and only admit candidates that follow the path closely.

The same trials for SMG are shown on Table 4.4. Here, however, we see that SMG did not scale as well as FMGs. For maps of relatively same sizes, SMGs failed to achieve acceptable success rates and ran slower than FMGs. As noted earlier, the main drawback of SMGs were that they failed to provide a variety of candidates for the path following method to consider and thus failed to find

| Size | Depth | Success | Time (ms) | Len. (m) | Inp. (m) |
|---:|---:|---:|---:|---:|---:|
| 33 | 8 | 63% | 412.6 | 10.15 | 10.12 |
| 86 | 9 | 71% | 356.7 | 10.96 | 10.94 |
| 174 | 10 | 87% | 188.7 | 12.13 | 12.10 |
| 433 | 11 | 90% | 148.3 | 12.75 | 12.66 |
| 867 | 12 | 93% | 145.8 | 12.36 | 12.27 |

Table 4.3: The effect of the size (in MB) of motion maps on the success rate and performance of the path following search for FMGs. The right-most four columns show the success rate of the search, the average time taken to search for the solutions, the average length of the solutions, and the average length of the input paths.

| Size | Depth | Success | Time (ms) | Len. (m) | Inp. (m) |
|---:|---:|---:|---:|---:|---:|
| 28 | 12 | 17% | 175.6 | 6.39 | 6.19 |
| 81 | 15 | 32% | 120.0 | 8.24 | 8.03 |
| 158 | 17 | 45% | 335.6 | 7.82 | 7.66 |
| 417 | 20 | 55% | 352.1 | 9.40 | 9.10 |
| 767 | 22 | 70% | 556.9 | 10.13 | 9.86 |

Table 4.4: The effect of the size (in MB) of motion maps on the success rate and performance of the path following search for SMGs. The columns are the same as in Table 4.3.

solutions following the input that.

We then compared our path following method against two different search methods: A* search and A* search with channel pruning (A*-Ch) as described in [MK11]. The A* search represents a popular technique to generate optimal solutions when unrolling the graphs. A*-Ch runs the A* search only inside a channel around the input path by pruning all branches that go outside the path channel. It represents a simple way to improve the A* search, however loosing optimality. A comparison of the methods illustrating their speed of computation is shown on Figure 4.12 and Figure 4.11. The A* search took 186s (Figure 4.12, A*-Ch took 9.4s and the search with motion maps took only 760ms (Figure 4.11). In this example our method gave an improvement of 245 and 12.3 folds respectively.

We have also performed numerical comparisons between the three search methods, with 1000 random trials on the environment shown in Figure 4.12. We used motion maps of depth 11 for the FMG and of depth 20 for the SMG. We have only compared trials where all the three methods were able to successfully return a solution. The results are shown in Tables 4.5 and 4.6.

Table 4.5 shows the results for the FMG. The average length of the input path was 14.34m. As it can be seen from the table, motion maps were significantly

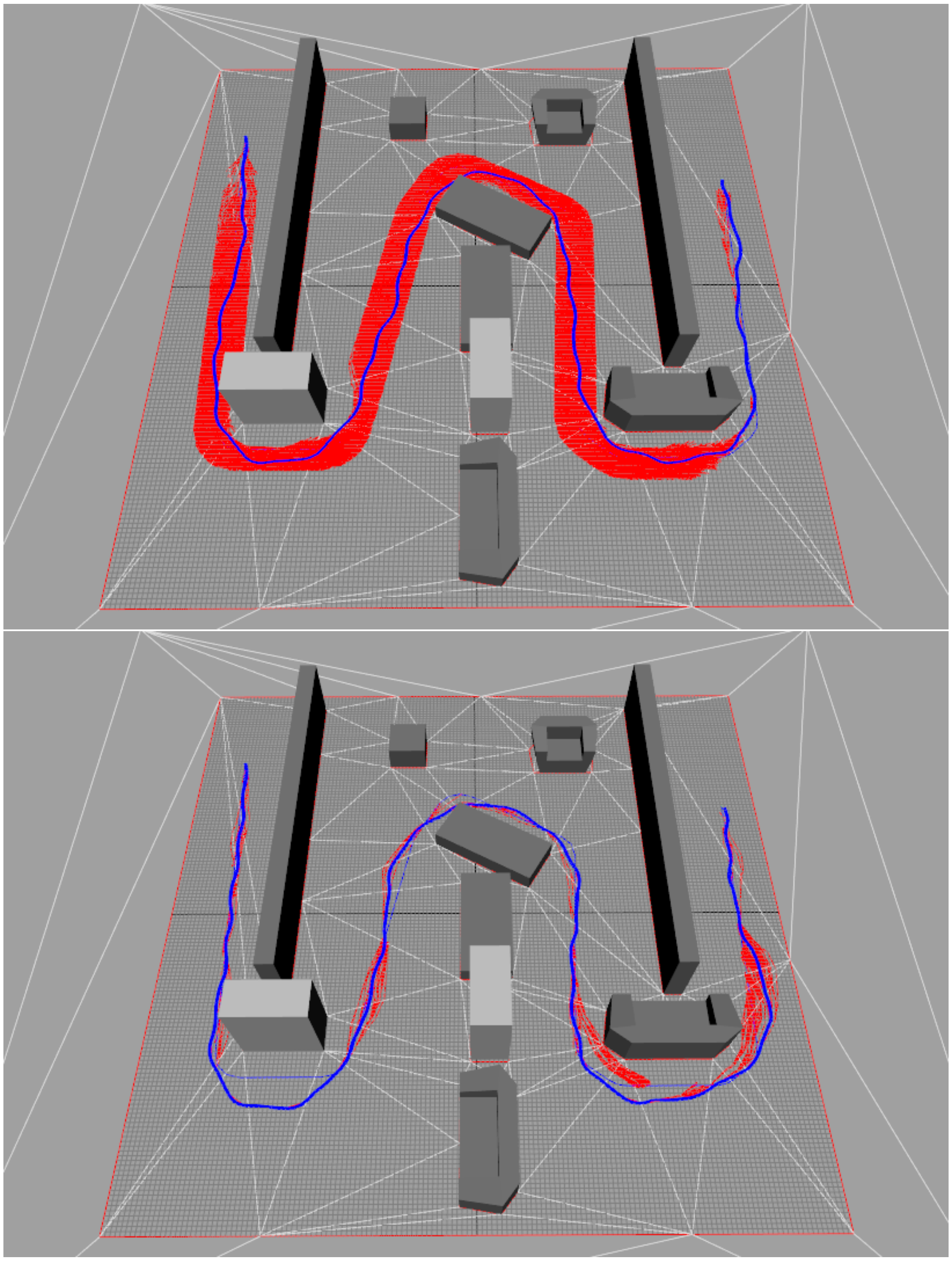

Figure 4.12: Similar searches as in Figure 4.11. Top image uses the A*-Ch channel pruning technique which took 9.4s to complete. Bottom image depicts the search with motion maps. The search in bottom image took 0.760s.

| Method | Success | Time (ms) | Length (m) | Speed up |
|---:|---:|---:|---:|---:|
| A* | 100% | 23527.8 | 13.95 | 157.6x |
| A*-Ch | 98% | 1742.1 | 13.95 | 11.7x |
| M. Maps | 93% | 149.3 | 14.51 | 1.0x |

Table 4.5: Search performance comparisons for A*, A*-Ch and motion maps in 1000 random trials using FMG. The columns show the overall success rate, the average computation time taken, the average length of the obtained solutions, and the relative speed improvement obtained with motion maps.

| Method | Success | Time (ms) | Length (m) | Speed up |
|---:|---:|---:|---:|---:|
| A* | 100% | 44184.0 | 12.48 | 50.6x |
| A*-Ch | 96% | 3192.4 | 12.58 | 3.6x |
| M. Maps | 58% | 872.4 | 13.30 | 1.0x |

Table 4.6: Search performance comparisons as described in Table 4.5, but here based on SMG.

faster than the other methods, and the length of the solutions were also very close to the original path length. Motion maps did show a reduced success rate of 93%. However, this rate is still excellent for a method based on precomputed search trees, and it is higher than reported results in previous related works. Furthermore, this rate can be improved to 100% by making sure that the motions in the database include all needed variations for a given environment. For instance in our paths around obstacles, more motions of different turning angles would be needed.

Table 4.6 shows the results obtained from running the same experiments with a SMG. The average length of the input paths was 12.87m. We notice that, although motion maps are always faster than the other methods, when SMGs are used, the success rate is decreased to 58%. FMGs are essential for motion maps to be effective, however, it is important to note that using SMGs with more elaborate precomputation techniques, such as the method by Lau et al. [LK10], could make SMGs more suitable for our path following method. However, we have not run any experiments to verify this.

The presented experiments clearly demonstrate the performance of the precomputed motion maps. The success rate of the overall method is highly related to the motion capture database used to build the underlying FMG. In our presented examples no special care was taken when selecting motion clips for building the used FMGs, and still the success rate achieved was of 93%. The presented experiments can also serve as a way to evaluate the suitability of the motion capture database, and additional methods can also be employed for measuring coverage [RP07].

Another important observation is that the granularity of the precomputed motion maps does not influence how well precomputed maps can store the motions. The reason for this is that we do not disregard paths that might lead to a cell that has been already occupied, and thus we never fail to capture all the available paths from the root of a precomputed motion map. Changing the resolution of the motion map might redistribute the paths into neighbouring cells or pack neighbouring cells into a larger cell but the number of entries in the motion map will not be altered.

However, the granularity of the motion maps does have a slight influence on the odds of finding a candidate path while sampling the motion map. If the resolution is very coarse, the sampling should be adjusted accordingly so that neighbouring cells are not queried, otherwise overly long motions might be considered. On the other hand, if the resolution of the motion maps is overly fine then a finer sampling should be employed in order to void missing possible candidates. Across all our experiments we kept the resolution of the precomputed motion maps constant and equal to the resolution of the sampling routine. One limitation of our method is that the storage space may become high for large databases.

## 4.8 Conclusion

We have presented new preprocessing and search techniques enabling unstructured motion graph structure to be efficiently employed for locomotion synthesis around obstacles in complicated environments. Several experiments were presented demonstrating the benefits of the proposed methods. The results are always of highly quality and are computed at interactive rates.

# CHAPTER 5

# Multi-Modal Data-Driven Motion Planning

In this chapter we present a new approach for whole-body motion synthesis that is able to generate high quality motions for challenging mobile-manipulation scenarios. Our approach decouples the problem in specialized locomotion and manipulation skills and proposes a planning algorithm that explores in an integrated way the search space of each skill and the transition points between skills. The locomotion skill is data-driven and ensures positioning coverage with quality guarantees. Manipulation skills can be algorithmic or data-driven according to data availability and the complexity of the environment. A coordination model evaluates possible transition points between locomotion and manipulation. Our overall method is able to automatically generate complex motions with precise manipulation targets among obstacles and at the same time achieve the same realism level of fully data-driven methods. We present example solutions for opening doors, relocating items in shelves, pouring water, etc. The produced examples were achieved with a small set of motion data and the generated motions remain realistic and human-like.

## 5.1 Introduction

High quality human-like character animation has attracted a great body of research in the past decades. Although many methods have successfully generated motions for a variety of problems, more work is needed to improve the fidelity and the level of control that these methods provide. Improving both of these aspects is particularly challenging because improving one comes at the cost of deteriorating the other. Therefore, methods whose solutions are particularly sensitive to this balance are currently most desired.

Previous methods have rightfully put a strong emphasize on realism. High fidelity, realistic and human-like motions can be best produced by the employment of data-driven methods which reuse and combine motion capture data and achieve high quality results. These methods can automatically provide a wide range of possibly infinite motions that may be generated from a finite set of previously captured motion capture data. Their main advantage is that they are generic and work with different styles and types of motions.

Although these data-driven methods can easily synthesize highly realistic motions that can follow paths and avoid obstacles, more complex tasks require not

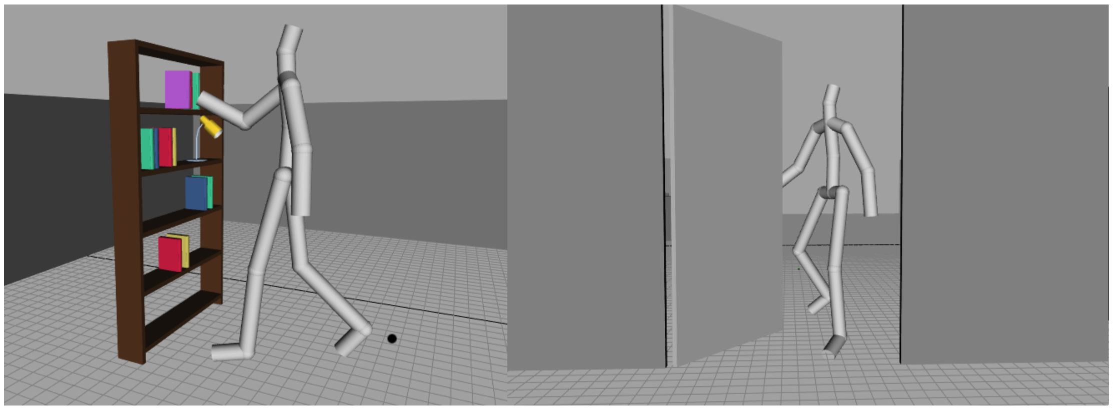
Figure 5.1: Various tasks as performed by the character. *Left*: The character is picking up a book from a shelf. *Right*: The character is opening a door.

only a wider range of input motions but also more dedicated algorithmic solutions in order to find valid solutions. For example, imagine a situation where a character needs to walk to a door, open the door and walk through the door. Rearranging many such examples will not easily account for doors of various sizes. Moreover, finding valid solutions as the user changes the position of the door handle relative to the door, makes the problem increasingly more difficult for a purely data-driven approach. Better approaches would utilize some inherent information about these tasks and try to solve the problem by exploiting this added information. For instance, for the task of door opening, it makes sense to decompose the problem in four different subtasks and solve them separately: first walk to the door, open the door, then walk though the door and finally go back to walking. In this examples, we see that the character goes through three different modes: walking, door opening and passing through the door.

Given the fact that data-driven approaches do not scale well with the size of their motion capture database, adding many examples of such complex tasks does not seem to be a feasible direction for an efficient planner. In light of this, we propose an approach that tries to fill this gap. Our method is a hybrid approach and produces high quality results while at the same time plans and solves examples of complex situations. This is achieved by dynamically combining motion capture examples with motion primitive skills based on localized planners. If the motion capture database well represents the solution space of the task, then the solution might solely be generated from motion capture examples; however, if the motion capture database does not contain all the desired motions, then the motion primitive skills will dynamically compete with motion capture examples in trying to find a valid solution, paving the way for an efficient motion synthesizer that utilizes motion capture data with parametrized and algorithmic approaches and thus efficiently solve complex problems and generate high quality results.

In this manner our planner may be seen as a global planner where numerous parametrized motions skills are activated depending on the various defined modes and compete to solve a particular problem. This approach greatly improves the reachability and the solution space of the motion graph and thus provides for a efficient method to solve complex tasks which involve obstacles and interactive with the environment.

## 5.2 Related Work

There have been three major group of approaches that have tried solve the problem of realistic human-like motion generation for virtual characters: key-framing, data-driven, and physics-based. While key-framing is still used for many purposes, with the advent of the motion capture technology, data-driven approaches have become increasingly more successful and popular; on the other hand, physics-based methods, although not as realistic as data-driven methods, are the only approach to account for external forces emerging from other characters or the environment.

**Motion Graphs:** First methods to automatically exploit the frame similarity within a motion capture database and extract new motions by means of optimization and search were introduced a decade ago by three similar papers [KGP02, LCR02, AF02, AFO03]. These works introduced the motion graph data structure which automatically created transitions between similar frames of the motion capture database. Using a search or optimization process, they were able to generate motions which were numerically similar but semantically different from the ones in the original motion database.

**Continuous Motion Graphs:** Many variants of motion graphs were developed to improve different aspects of the method: parametrized motions clips and techniques to transition between blended clips [SO06, HG07] made possible for fast motion synthesis, however, did not easily allow for planning motions in environments with many obstacles. Time interpolation of motion paths in order to improve the solution space of motion graphs [SH07] provides for wider range of possible solutions but this is achieved at the expense of increasing the size of the motion graph which further slows down the search procedure. Furthermore, complex tasks will require interpolation of more than two motion path which make the method time-consuming for solving various examples of the same task. Further efforts were made [RP07, ZS08, LS09] to construct motion graphs that are more compact and have better connectivity.

**Precomputation:** Other papers in the literature, such as the works of Lau and Kuffner [LK06], precomputed a finite state machine of behaviours in order to obtain a search tree which was queried during the motion extraction process. Similarly, motion maps [MK12] were precomputed for all the nodes of a motion graph in order to have real-time motion synthesis from an unstructured motion capture database. All these works use either manually crafted motion capture database or features to improve the structure of the motion graphs, or use precomputation to speed up the search; however they can not plan at the level of the configuration space of the character when needed as do the motion primitives in this chapter.

While these methods were the first to generate high quality motions in a generic fashion, solving complex tasks with motion graphs is not a suitable approach, as discrete rearrangement of motion examples might not solve problems where continuous joint-level control is needed to reach precise targets in complex environments.

**Blending:** Blending motion examples in order to generate in-between examples were effectively used for object manipulation problems. Kovar et al. [KG04] automatically extracted similar motions from large databases and built a space of motions. Huang et al. [HMK11] constructed blend spaces and used randomized algorithms to find action motions which successfully avoided obstacles. Basten et al. [BE11] built a coordination model which realistically combined spliced motions. Blending generates a variety of motions, however, usually the example data has to be carefully prepared to synthesize realistic results.

**Reinforcement Learning:** Treuille et al. [TLP07] used methods based on reinforcement learning by learning a set of continuous basis functions to control a character real-time. Levine et al. [LLK11] similarly uses reinforcement learning to control a character by the half of a set of parametrized motions. Although, reinforcement learning is a promising direction to generating human like motions, it is often very time consuming to learn control policies which do not easily work with complex environment where there are many obstacles.

**Physics:** Bai et al. [BSL12] introduced a physics-based method for concurrent object manipulation generation using a manipulation graph. Min et al. [MC12] enhanced the concept of motion graphs by building a compact model for structural and style variations.

**Planning:** Choi et al. [CLS02] combined PRMs and motion capture to solve the problem of biped locomotion in an environment with obstacles. Yamame et al. [YKH04], used traditional motion planning algorithms to solve for various object manipulation tasks involving interaction with the environment. Pan et al. [PZL10] combine a hierarchical model decomposition with sampling based algorithms to generate character motions in constrained environments.

**Multi-Modal Planning:** Multi-modal planning was first used by Hauser et al. [HNG07] to generate motions for humanoid robots within different modes which simplified the high dimensional motion planning problem. Kallmann et al. [KHB10] introduced parametrized motion skills and used them to generate motions for a character in complex environments while avoiding obstacles.

## 5.3 Multi-Modal Planner

In this section we explain our data-driven approach which simultaneously allows configuration level control. This is made possible by the employment of a set of motion primitive skills whose tasks is to control the character at a particular mode. A motion primitive skill is defined as specific local planner that generates partial solutions for the subtask which it is assigned to. Each motion skill usually controls a subset of the character's configuration but may also solve for the entire posture of the character. A skill gets activated depending on the mode of the character. Generally, there is a motion primitive skill for each mode. For instance, an example of a motion primitive skill is the locomotion skill which generates walking motions. This skill implements a motion graph and makes it possible for the character to move around in order to perform various tasks.

While the locomotion skill controls the entire posture of the character, other skills are usually responsible for a subset of the character's posture. As mentioned earlier, the door opening task is subdivided into three modes. One of the modes is to transition from walking to reaching motion. This motion places the hand to the character to the door handle in order to open the door. The reaching skill controls only the hand of the character and its root orientation. The rest of the posture comes from the locomotion skill. The local planner that synthesizes the reaching motion is based on a time-parameterized RRT planner [LaV06]. This motion is superimposed to the walking motion such that the character walks and reaches the handle at the same time. Because the reaching skill is parametrized by the position of the door handle, this skill can generate wider range of possible reaching motions.

The next mode generates the door opening motion. This mode has a door skill and the goal of the skill is to open the door incrementally by checking whether the arm of the character will can maintain contact with the handle while avoiding colliding with the door. The door opening skill does not involve any locomotion, so the locomotion skill is not active during this mode. The door does not have to be opened fully. The door opening skill will try to open the door as much as it possibly can and will add this newly generated clip into the frontier of the search queue and continue exploring other motion primitive skills until one finally reaches the goal. With other words, motion skills generate new parametrized search tree leaves and as such increase the possible solution space. After each motion skill expansion, the generated clips are inserted into the fringe of the motion graph search tree and the best candidate is popped from the search to resume exploring various other combinations.

Next, we explain our general search planner $\Pi$. Our planner operates over a set $\Sigma$ of parametrized motion skills. We indicate a motion skill as $\sigma_i \in \Sigma$. Each invocation of a motion skill $\sigma_i$ generates a motion $m_i$. The global solution is produced by appending the generated partial motions $m_i$. The generated motions

$m_i$ are a function of the motion skills $\sigma_i$ and the parameters $\pi_i$ specified to the motion skill by the global planner $\Pi$.

$$\sigma_i(\pi) = m_i \in \mathbb{R}^{l \times c} \tag{5.1}$$

where $l$ is the number of frames in the motion $m$ and $c$ is the size of the dimensionality of the configuration space $\mathbb{C} = (p, q_i)$ of a standard hierarchical character. The planner $\Pi$ also maintains a list of modes $\theta \in \Theta$, a function $\tau$ which specifies mode transitions and a set of rules $\lambda \in \Lambda$ which determine what motion skills $\sigma_i$ may be activated in given mode $\theta$:

$$\lambda : \Theta \to \Sigma \tag{5.2}$$

$$\tau : \Theta \to \Theta \tag{5.3}$$

The set $\Theta$ and $\Lambda$ are provided by the user depending on what tasks the planner is required to solve. Given these four parameters, the global planner $\Pi(\Sigma, \Theta, \lambda, \tau)$ begins the search by expanding motion skills until a solution is found. During the search procedure a search tree is formed which is akin to the unrolled search tree formed by a motion graph search. The planner maintains a frontier of motion skills which are prioritized depending on an A*-like objective function. The planner decides which skills can be active at a given mode and which other modes could be expanded from the current mode. The planner is responsible for maintaining the correctness of the modes transitions depending on the provided rules.

The modes are likely to depend on the problems that the planner is expected to solve. Similarly, the parameters specified by the planner to the motion skills are depended on the nature of the skills. They maybe be discrete and finite, or continuous with pre-determined expected range. The planner then randomly chooses parameters and expands all the skills that could be active on that mode. This makes it possible for motion skills with their local planners to compete with each other in order to provided the most efficient solution. Such a design makes it possible to increase the range of possible motions and at the same time efficiently satisfy the constrained of the problem. This approach makes our planner able to solve complex problems that can not easily be handled with current methods.

For many daily activities, the tasks that needs to be solved can be decomposed to modes which could be best partially solved with a dedicated motion skill. For instance, for a door opening task there are three modes: walking, door opening and walking through a door. The walking mode is solved by a locomotion skill, as is the walking through a door mode. The door opening, on the other hand, it is solved by the door skill. In the next section we describe a list of motion skills and a set of rules which efficiently solve a wide range of everyday tasks.

## 5.4 Locomotion and Manipulation Skills

### 5.4.1 Locomotion Skill

The locomotion skill $\sigma_{loc}$ is the most important motion skill as it moves the character around and prepares the character for other mode transitions. The locomotion skill is based on a motion graph and operates on the entire configuration of the character. We build the locomotion skill by creating a feature-based motion graph from our motion capture database which mainly contains locomotion and does not include motions such as opening doors or picking up objects. After the motion graph is built we use the connectivity of the graph to determine the parameter list of the skill and also which modes may be invoked in from the locomotion mode. The locomotion parameter $\pi_{loc} = (n_i, j)$ takes two parameters, the current node of the motion graph $n$ and number of $j$ children nodes of $n$. For example, $\sigma((n_i, j)_\pi) \rightarrow n_j$ which returns the motion $m_j$ of the $j$th child of node $n_i$.

A locomotion skill can transition to another skill if a transitional constrained is satisfied. One such constrain is the transfer skill which is always invoked within the locomotion skill. The transfer skill tests at each step whether an object of interest such as a door or a shelf is within the close vicinity of the character. If that is the case, the transfer skill will follow by trying to invoke the reaching skill. The reaching skill then will determine whether the character can reach the handle of the door or the book on the shelf.

### 5.4.2 Reach and Release Skills

The reaching $\sigma_{reach}$ and releasing skill $\sigma_{release}$ are based on a time-parametrized RRT planner. These skills control only the arm of the character. The rest of the characters posture is planned by the locomotion skill. They are activated on top of the previous locomotion skill. With other words, these skills takes the motion generated by the locomotion skill, makes a copy of it and modifies the motion corresponding the arm and reinserts its result into the frontier for further consideration. The initial locomotion still remains unchanged in the frontier and it will be considered by other motions skills. The manipulation skills take two parameters $\pi_{manip} = (p, p')$ which are the character postures. The first parameter $p$ comes directly from the first posture of the character from the initial locomotion. The second parameter $p'$ is the posture we wish the character to assume at the end of the motion generated by this skill. This usually is a posture where the character is holding the door handle or a book on the shelf. Given the parameter $\pi_{manip}$, the manipulation skill $\sigma_{manip} = \sigma_{reach}, \sigma_{release}$ generate a motion which reuses the locomotion motion and plans the motion of the arm such that character starts in posture $p$ and ends with the posture $p'$.

The manipulation skills are time parametrized in order to simultaneously walk and reach for the target. This is made possible by advancing the locomotion by the given time parameter of the arm motion. This allows for a realistic RRT motion when compared to a sequential solution when the character first has to walk and only then reach for the goal. Also, these skills are based on sampling based algorithm and thus are not based on examples motions. This makes the skill particularly important because it can solve for a wide range of arm positions.

### 5.4.3 Transfer Skill

The IK-based skill $\sigma_{tr}$ is the simplest motion skill, yet it is the most frequently used one. This skill has two primary tasks: the first is to establish whether mode transitions are possible. That is, the transfer skill determines if a mode change is possible. Such examples include mode changes from locomotion to the reaching mode. This skill implements an analytical IK for the arm of the character. Given a target position $p$ and orientation $q$ and the last frame of the previous locomotion $f$, the transfer skill solves for the arm such that the arm of the character reaches precisely the desired target location $t = (p, q)$. Thus, the parameter list of the transfer skill $\sigma_{tr}$ is $\pi_{tr} = (f, t)$. The result of the skill is a frame $f'$ which is contains all the configuration of $f$ with the arms configuration subset determined by $\sigma_{tr}$. The resulting frame $f'$ is also used as the target posture $p'$ for the reaching skill $\sigma_{reach}$. The transfer skill is special in a sense that it does not generate a result motion $m_{tr}$ unlike other motion skills in $\Sigma$.

### 5.4.4 Action Skill

The action skill $\sigma_{ac}$ is a data-driven motion skill. This skill is based on an example of motions that perform a certain action. Examples of action motions include motions such as pointing, pouring, kicking, etc. This skill is implemented similarly to work of Huang et al. [HMK11]. The skill requires a set of blendable examples which can be blended together in order to generate any new in-between motion. Similarly to the transfer skill $\sigma_{ik}$, the action skill $\sigma_{ac}$ accepts a parameter list $\pi_{ac} = (f, t)$ where the frame $f$ is the last frame of the previous locomotion and $t$ is the target position and orientation where the action needs to be performed. It returns a motion which is a weighted blend of the set of examples motions within the action skill $\sigma_{ac}$.

## 5.5 Multi-Modal Search

We now explain our multi-modal search. As mentioned earlier, our global planner takes a list of skills, a mode transition function and the constraints that need to be satisfied. The planner maintains a priorities frontier akin to that of graph search. Although the search uses motion primitives to generate motions, our planner generates a search tree of *motion* clips where motion primitives add or modify

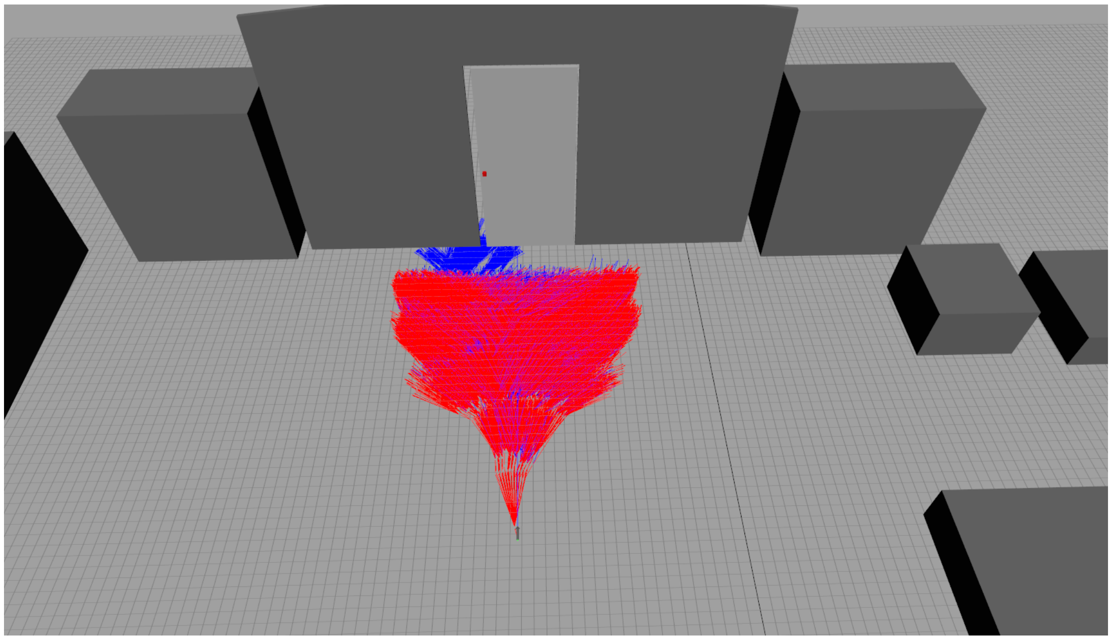

Figure 5.2: An example of a multi-modal search tree. Red segments represent unmodified examples from the motion capture database and blue segments represent the motions modified by the motion primitive skills.

the given motions. The user generally specifies the start and goal location for the character. The search usually starts in the locomotion mode, although it could start from any mode. The locomotion search expands nodes in similar fashion as in Chapter 1 and Chapter 2 and tries to expand towards the goal. During each expansion, depending on the constraints of the problem, such as the position of the handle of the door or position of the book on the shelf, the locomotion skill actives the transfer skill and checks whether a transition may occur to any other mode. In the door opening example, the character would switch from the locomotion mode to the manipulation mode. The transfer skill is a fast operation, which quickly evaluated whether a mode switch could occur. If the character is close to the constraint, and the transfer skill is successful, then the mode is switched and the character is within the next mode as defined by the mode transition function. This mode switch only occurs for the branches of the search tree which stem from the current motion node where the mode switched occurred. Other branches may be in different modes. In the door example, there will be other motion nodes still expanding within the locomotion and generating other motions that place the character in different locations for another mode switch. Once a switch is made, usually a manipulation skill takes over and given the parameters it needs to solve tries to solve the subtask.

For example, for the door opening case, this is the reaching skill. This skill is continuous and randomized and takes parameters: the current posture of the character prior to switching to the manipulation skill and the position of the handle. Since the reaching skill is a manipulation motion primitive, it is a continuous planner and therefore can solve a wide range of possible motions which is essential for the successful execution of our multi-modal planner. The resulting motion of the reaching skill is an arm reaching motion which is superimposed on top of the locomotion, leading to a reaching motion while the character is still moving towards the goal. If the mode is successful, it switches to the next mode and activates all the skills of the next mode. In the door opening example, there is only one mode following the reaching mode and that is the door opening mode. By using a strict function that defines mode transitions and a list of skills that may be generated within a mode, the multi-modal planner ensures that it never generates motions which violate the mode transition function. Therefore we do not see motions where the character walks, reaches for the door handle and resumes walking without trying to open the door. The correct mode switches make sure that allowed sequence of motion primitive get activated. The planner then proceeds to the next mode and activates all the motion primitives within that mode. In the door opening case, after the manipulation task, there is only one mode and that is the door mode where the character opens the door as fully as possible. The next modes include release the handle and going back to the locomotion until the goal is successfully reached.

It is important to mention that the planner is building a search tree of different possible candidates each attempting to solve the general problem. All of the candidates are sorted in A*-like manner. Occasionally, some of the branches of the search tree will fail or collide with the environment; however, the planner will continuously generate new instances of skill activations until a suitable sequence successfully solves the problem. Thus, the planner will activate many parametrized variations by by the guidance of the mode transition function and the list of motion primitive skills that are designed to specifically solve the subtask within the mode.

In addition, some modes will use collision avoidance deformations which will further help the planner in achieving the goal. For instance, the door opening skill rotates the torso of the character as it attempts to open the door. This not only gives the character a change to open the door fully but it also helps in generating a more human-like motion. Moreover, the locomotion skill may use other forms of collision avoidance deformations such as moving their arms to avoid colliding with the door. These deformations help to increase the solution space of the generated results.

### 5.5.1 Deformation

In addition to the motion primitives which control the character at configuration level, we also deform the motion examples of the motion graphs in order to precisely meet the constrains of the problem. The deformation explained in this section provides for a simple deformation model which can be easily quantifiable and also provides an easy way for increasing the solution space of the possible solutions.

Given a motion $m$ which is associated with a node $n$ of the motion graph, the deformation procedure deforms $m$ is this manner: first, the user specifies the amount of the desired deformations. This is done by providing a list of $\alpha_i \in A$ in angle degrees. For each $\alpha_i$ a new node $n_d$ is generated which is associated with the deformed motion $m_d$ and then the node $n_d$ is inserted back into the motion graph. For a motion $m$ and an angle degree $\alpha$, each frame

$$f_i(p_i, o_i, v_i) \in m \tag{5.4}$$

where, $p_i$ and $o_i$ is the root position and orientation of the frame $f_i$, and $v$ is a vector such that $v_i = p_{i+1} - p_i$ or with other words $v_i$ is a positional differential between frame $f_{i+1}$ and $f$, we create a quaternion $q(p, \alpha)$, which represents a rotation about the positive Y axis of $\alpha$ degrees centered at $p$. Then for each frame $f_i$ we apply the following deformation:

$$o'_i = q(p, \alpha) \cdot o_i \tag{5.5}$$

$$v'_i = q(p, \alpha) \cdot v_i \tag{5.6}$$

The deformation is applied iteratively. In this first iteration all the frames of $m$ are deformed. In the next iteration, only frames between the second and last frame are deformed. Generally, at the $i$th iteration, only frames between $m_i$ and $m_n$ are deformed. See Figure 5.3 for an example.

In order to control the desired deformation we provide a method to quantify the deformation. The most undesirable artifact of deformation is the feet sliding. Therefore, we observe the toe and heel joints of the skeleton and measure the deformation as the averaged squared distance of the toe and heal joint position between the original motion and the deformed motion. The quantification is done as follows: first we detect all frames where the heel or toe joint is fixed on the floor. Then we group these frames into distinct sets such that all frames of the set belong to the same stepping motion. Then, we translate and rotate all the joint positions in the set by the root position and orientation of the first frame of the set. This creates a reference frame where we can measure how much have the heel and toe joints differed from the original position. Figure 5.4 shows an example. We see that for the original motion (red points) the heel or toe joint is fixed. As the deformation becomes more aggressive (green and blue) we see that the heel and toe joint start drifting further away from their desired fixed position. The deformation is then defined as weighted sum of square distance averages. All deformation above a certain predefined threshold are not considered. In our experimentation, we deformed each node of the motion graph by $\alpha = \{-0.4, -0.2, 0.2, 0.4\}$. At

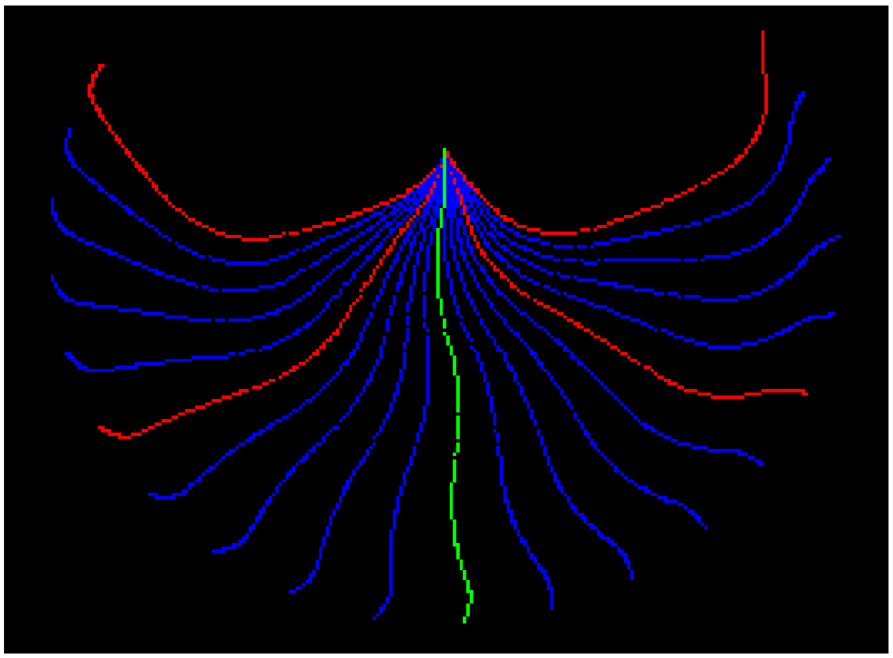

Figure 5.3: Motion deformation at various angles. The green projection is the original motion, and the blue and red deformations show the range of the motion when deformed between -1 and 1 degrees.

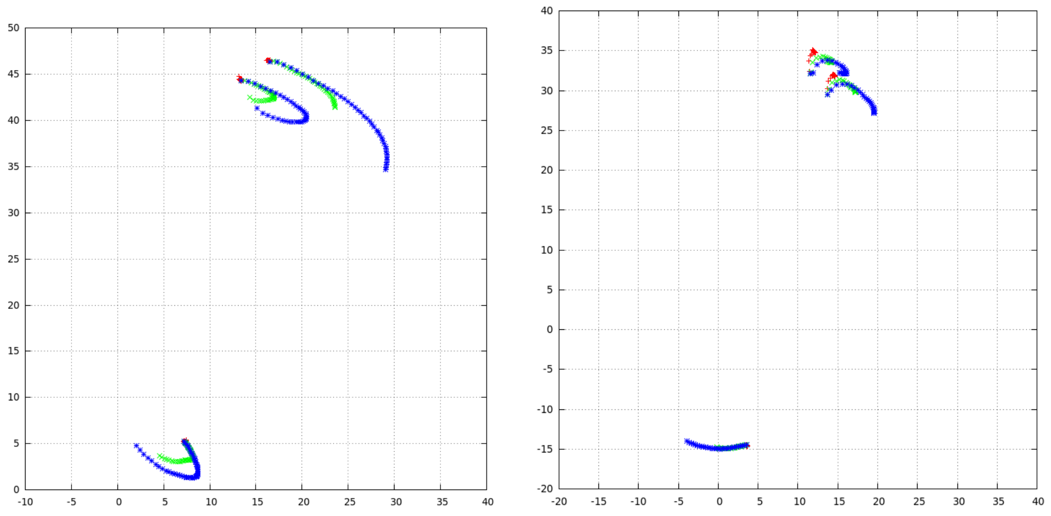

Figure 5.4: Left: Heel joint position projection for three stepping motions. Red points represent the original motion, the green points represent a deformed motion with $\alpha = 0.5°$ and blue represents a deformed motion with $\alpha = 1.0°$. Right: Same comparison as in the left image, however, in this instance for the toe joint.

these levels of deformation, the feet joints did not appear sliding.

### 5.5.2 Collision Avoidance

Beside the deformation described in Section 5.5.1, our method uses another type of deformation for the purpose of collision avoidance. These forms of deformation are usually applied only on a single joint to the motion of a single motion graph. For example, during the door opening mode while the character is opening the door using the door skill, the collision avoidance is rotating the root joint by $1^\circ$ at each attempt to further open the door. This not only looks more natural but at the same time makes it possible for the character to more easily open the door.

Another collision avoidance deformation is deployed during the door walk through mode, where the left arm is moved gradually towards the torso in order to avoid colliding with the wall. The deformation is spread out gradually along the motion and follows a Gaussian curve such that the arm has human-like velocity profile.

### 5.5.3 Coordination

In order to maintain the naturalness of the generated motion, a coordination model is employed. As the local planners try to generate various motions the coordination models assures that other joint are coordinated with the joints which are being changed. The parameters of the coordinations model are tuned from experiments run by human subjects. The coordination model maintains a gaze model so that the head of the character always stares at the object that is being manipulated. The coordination model always takes into an account the coordination between the upper body and lower body motions so that they look coordinated. In addition to maintain the joint positions within the character the coordination model also determines some constrained concerning the task that is being considered. For example, for the door opening task and book picking task, the coordination model specifies the approach angle to the door or book shelf. This is implemented by culling the branches of the search tree that do not satisfy these constraints. For extensive searches this discrimination speeds up the search and does not expand nodes which are known to know yield results.

## 5.6 Results

To show the effectiveness of our algorithm, in this section we present three various scenarios where our framework makes use of the inherent information within the problem task and solves it efficiently.

### 5.6.1 Door Opening

As mentioned in Section 5.1 the problem of door opening is a particularly challenging problem for data-driven methods, primarily because data-driven approaches such as motion graphs can not easily account for slight variations in the problem definition. For example, if we character needs to reach for the handle to open the door, the motion graphs needs many such examples to account for a wide range of possibilities. A wide range of motion does not easily scale with motion graphs as the computational requirements during the search procedure are exponential to the number of frames in the motion capture database. Therefore, multi-modal approaches are more efficient in this regards as they do not fully rely on a discrete rearrangement of motion examples but use localized planner that solve the subtasks as the character moves within modes.

The problem of door opening is separated in four different modes: locomotion mode, door reaching mode, door opening mode, door release mode and back to the locomotion mode. The user specifies the starting and goal position for the character and also inputs the environment. The search begins in the locomotion mode and must not necessarily pass through these nodes in order to reach the goal. With other words, if there is another way of reaching the goal, without opening a door then that alternative will be competing as well in the general search.

The skill set $\Sigma$ for the door opening problem contains the following parametrized motion skills:

$$\Sigma = \{\sigma_{loc}, \sigma_{tr}, \sigma_{door}, \sigma_{reach}, \sigma_{release}\} \tag{5.7}$$

the $\sigma_{loc}$ locomotion motion skill is based on a motion graphs and is the main skill which move the character around. The parameter list $\pi_{loc} = (s, g)$ for $\sigma_{loc}$ takes the start location $s$ and a goal location $g$ and expands a motion graph search tree starting from $s$ and leading towards $g$ by expanding nodes akin to an A*-like search mechanism.

The transfer skill $\sigma_{tr}$ takes two parameters $\pi_{tr} = (c, p)$ such that $c \in \mathbb{C}$ and $p \in \mathbb{R}^3$, where parameter $c$ is the last character posture as defined by the motion graph node from which the transition to this mode was made. The parameter $p$ is the location of the door handle. As the door is opened the handle position is updated accordingly. This motion skill primitive implements an analytical Inverse Kinematics (IK) for a 7 DOF arm. If a solution exists, the posture $c$ is updated such that when applied the character would be holding the door handle at $p$.

The door opening motion skill $\sigma_{door}$ applies the $\sigma_{tr}$ repeatedly until the door can not be further opened due to a failure of $\sigma_{tr}$ or due to a collision between the character and the environment, particularly the door. The parameter list $\pi_{door} = (p, q, path)$ takes into consideration the position $p$ and orientation $q$ of the door and also the *path* which when parametrized defines the parameter list for the subsequent $\sigma_{tr}$ invocations. With other words, the *path* parameter defines

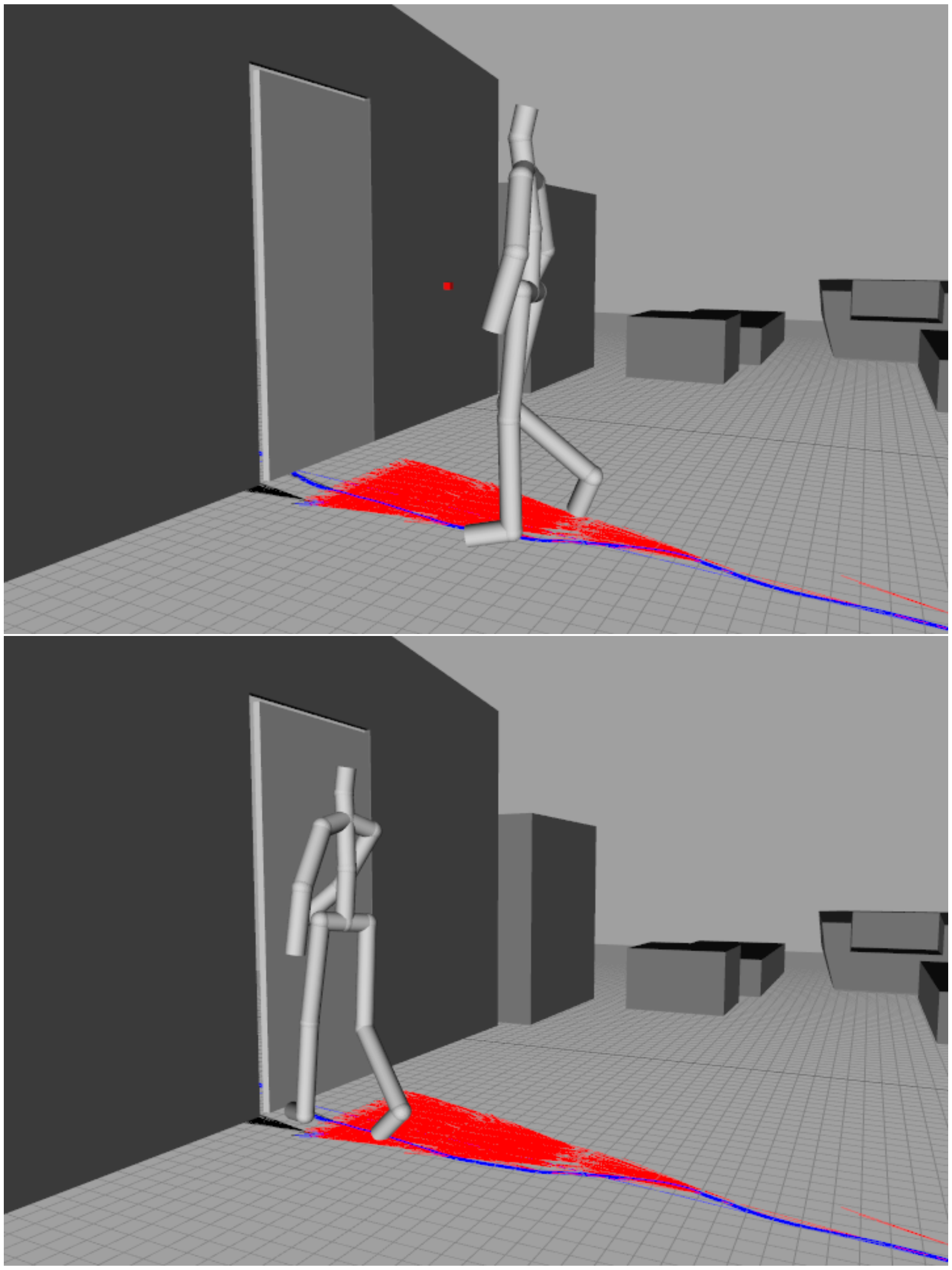

Figure 5.5: An example of door opening. Top image shows the character while still within locomotion skill. The bottom image shows the character in the last frame of the reach skill while holding the handle of the door.

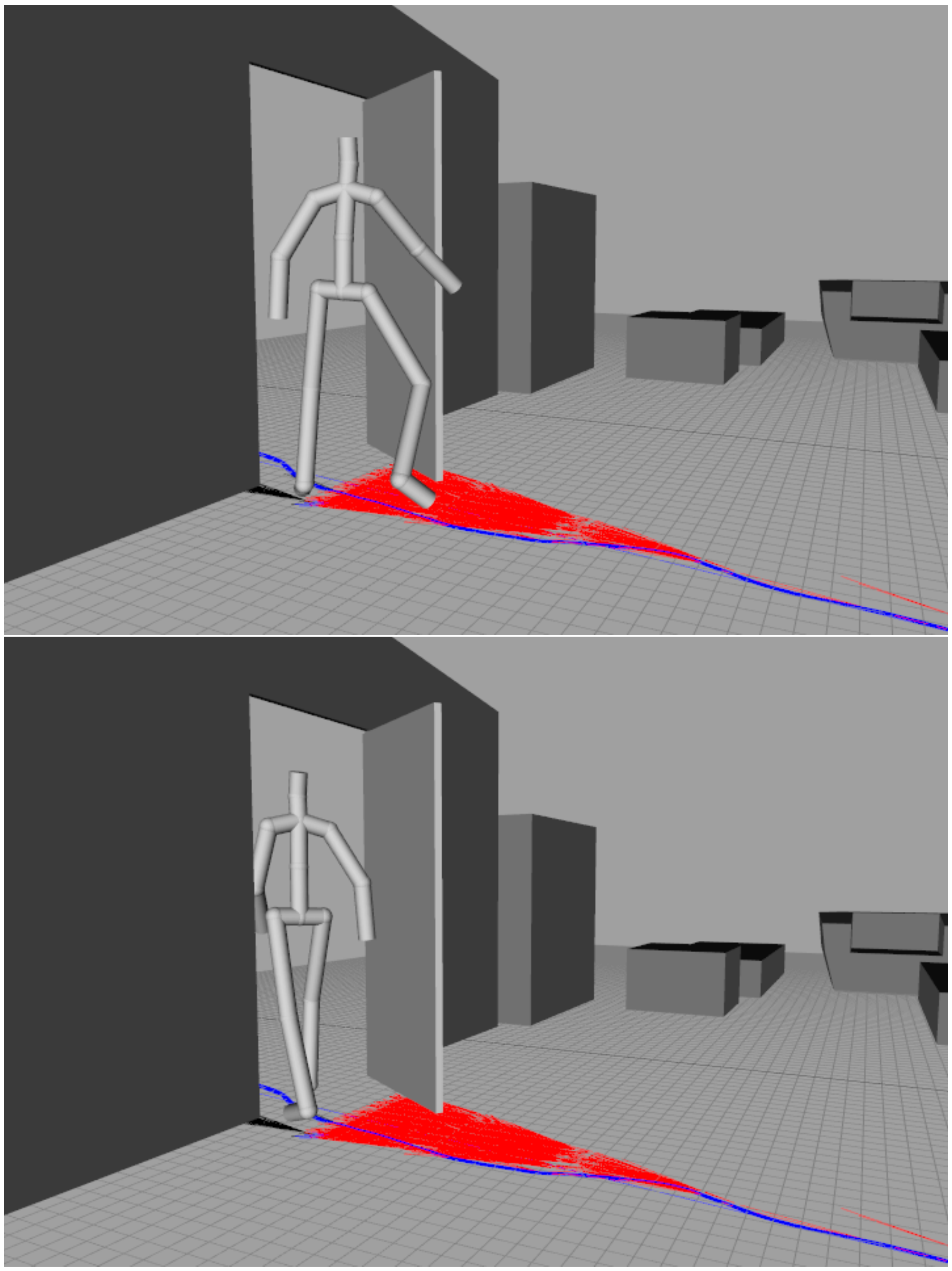

Figure 5.6: Continuation of the door opening example from Figure 5.5. The top image shows the character within the door skill trying to open the door as fully as possible. The bottom image shows the character going back to the locomotion skill while collision avoidance deformation (left arm) activated in this image.

the path that the character hand must follow in order to open the door. In our implementations we used circle and meridian arcs (based on an ellipsoid) with the later yielding better results due the way humans open doors. This motion skill primitive does terminate when the door is fully opened. The goal of the skill is to open the door as much as possible without colliding with the environment. Whether the character can successfully walk through the door is determined by the rest of the search with many different alternatives racing to search for a solution with the door opened at different angles.

The door motion primitive also deploys collision avoidance deformation by rotating the root position of the character and the door opening skill is trying to open the door. At each step, the character's root joint is rotated in order to give the character a better opportunity to further open the door. In order to assure that feet sliding does not occur, the door opening skill also re-parents the feet by using a simple IK module for the feet.

The reach motion skill $\sigma_{reach}$ and the release motion skill $\sigma_{release}$ are similar in nature but operate in different modes. The $\sigma_{reach}$ skill which allows the character to simultaneously play the motion by $\sigma_{loc}$ and at the same time plan the motion of the arm of the character. The parameter list $\pi_{reach} = (c, c')$ takes the initial configuration of the character $c$ which is the first frame taken from the result of $\sigma_{loc}$ and $c'$ is the result of applying $\sigma_{tr}$ at the last frame of the results of $\sigma_{loc}$. The result of $\sigma_{reach}$ is a motion that takes the character from $c$ to $c'$. With other words, this skill will produce a motion which would transition the character from walking to a posture where the character can reach the handle of the door, preparing it to open the door.

The release motion primitive skill $\sigma_{release}$ similarly to $\sigma_{reach}$ solves for the arm of the character; however, in the release mode the goal is to move the arm from a posture where it is holding the handle of an open door back to the last frame of the result generated by the last invocation of $\sigma_{loc}$. This brings back the character to the locomotion mode where it could attempt to pass through the door. The parameter list $\pi_{release} = (c_d, c)$ of the motion skill $\sigma_{release}$ takes the two above-mentioned postures and solves for the arm by using the manipulation skills. Unlike $\sigma_{reach}$, this motion primitive skill does is not time-parametrized and as such the character's other body parts remains unchanged.

Next, we specify the modes of the door opening problem. For this task we have the following modes:

$$\Theta = \{\theta_{loc}, \theta_{manip}, \theta_{tr}, \theta_{door}\} \tag{5.8}$$

and the following rules indicate which motion primitive skills may be invoked in these modes:

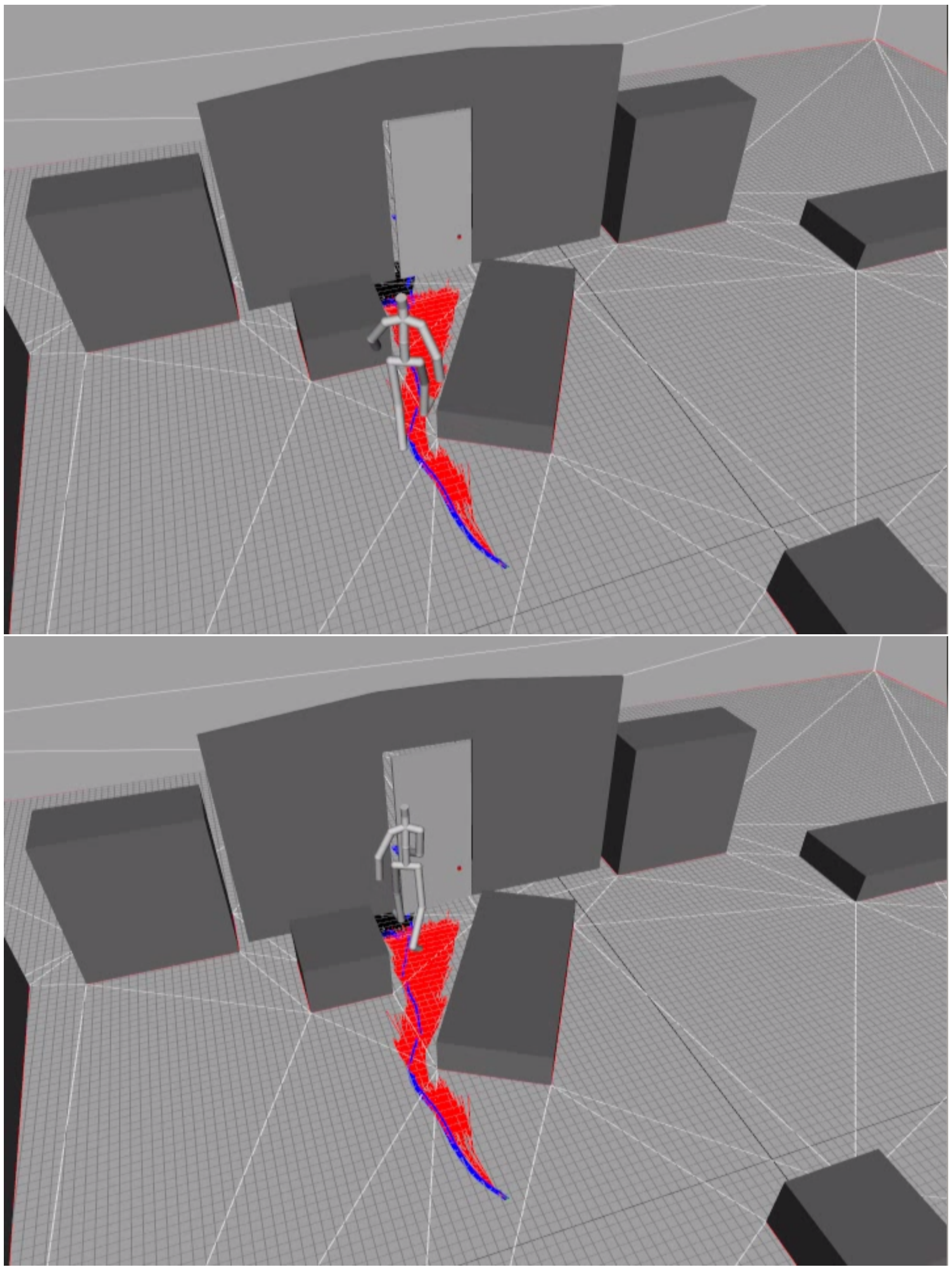

Figure 5.7: Similar door opening example as in Figure 5.5 but with obstacles in front of the door.

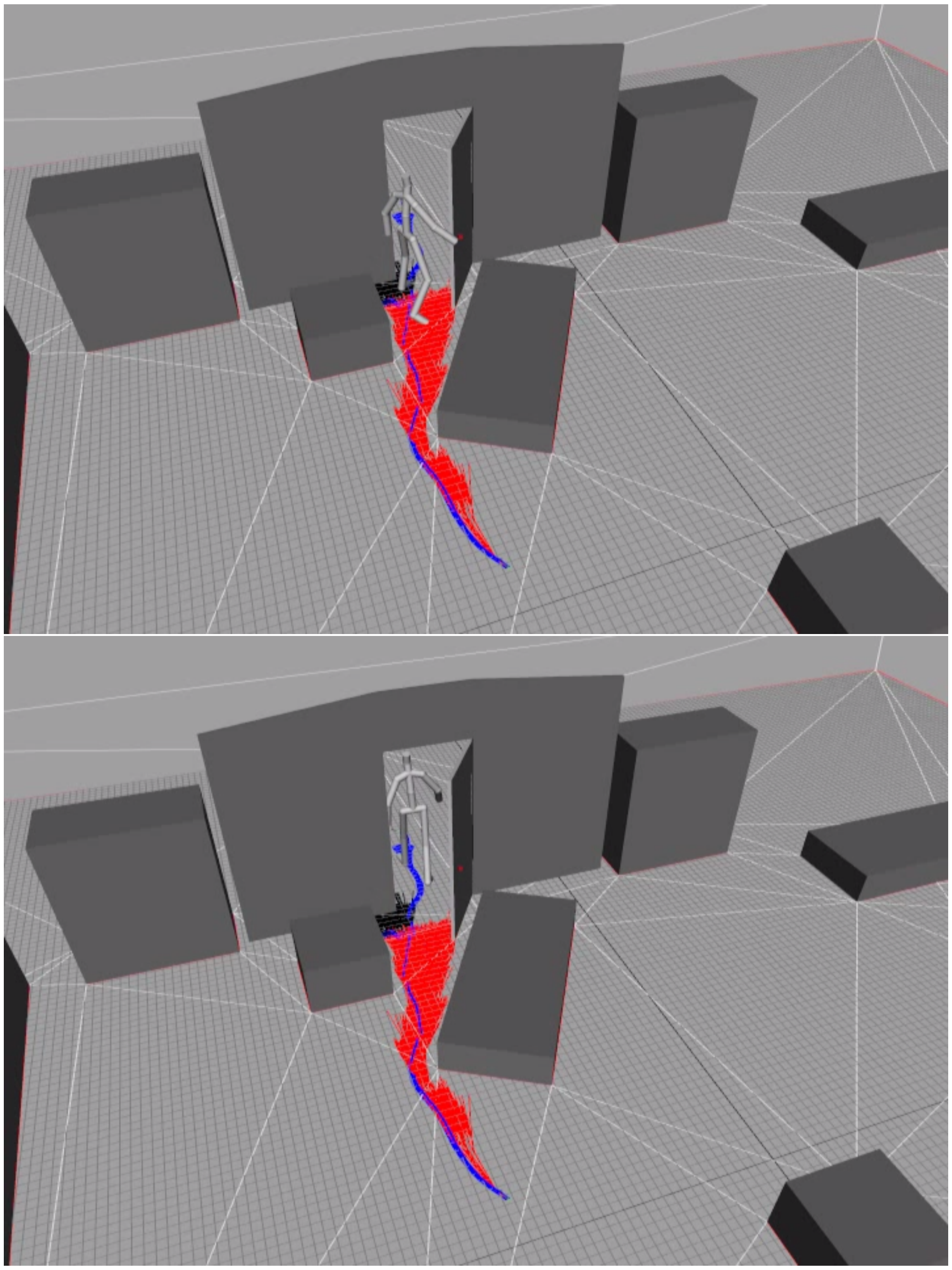

Figure 5.8: Continuation of the example on Figure 5.7.

$$\begin{aligned}
\lambda(\theta_{loc}) &= \{\sigma_{loc}, \sigma_{tr}\} \\
\lambda(\theta_{tr}) &= \{\sigma_{tr}\} \\
\lambda(\theta_{manip}) &= \{\sigma_{reach}, \sigma_{release}\} \\
\lambda(\theta_{door}) &= \{\sigma_{door}\}
\end{aligned} \tag{5.9}$$

and function $\tau$ specifies the mode transitions:

$$\begin{aligned}
\tau(\theta_{loc}) &= \{\theta_{loc}, \theta_{tr}\} \\
\tau(\theta_{tr}) &= \{\theta_{manip}\} \\
\tau(\theta_{manip}) &= \{\theta_{door}, \theta_{loc}\} \\
\tau(\theta_{door}) &= \{\theta_{manip}\}
\end{aligned} \tag{5.10}$$

Given the functions $\tau$ and $\lambda$, the set of modes $\Theta$ and a set of motion primitives $\Sigma$ the door opening may be used within the framework of multi-modal planning to search for a solution. Each invocation of a skill creates a new node in the search tree and gets inserted to the sorted frontier associated with a cost for this new motion. In this view, the multi-modal planning may be seen as an extension of motion graphs where motion primitive skills may take an motion from one of the motion graph node, modify the motion to solve a particular subtasks and reinsert this new motion back into the search tree as if this modification was already within the motion capture database.

### 5.6.2 Book Picking

In this section we show another example of multi-modal data-driven planning. In this scenario the character walks to a book shelf and tries to pick up a book and replaces it to another place on the shelf. The motion skill used in this examples are similar the ones explained for the door opening motion which shows that motion primitives skill may be used for various different tasks. Unlike the previous example, we will not be utilizing the door opening skill $\sigma_{door}$ because we do not have to open a door in this particular case. However, we will define a new motion primitive skill $\sigma_{displace}$ which is based on an manipulation skill whose tasks is to displace the book from its initial position to new position on the shelf. The motion of the character include that of the book has to follow a collision free trajectory without colliding with the character or other objects on the shelves.

The skill set $\Sigma$ for the book picking example contains the following motion primitive skills:

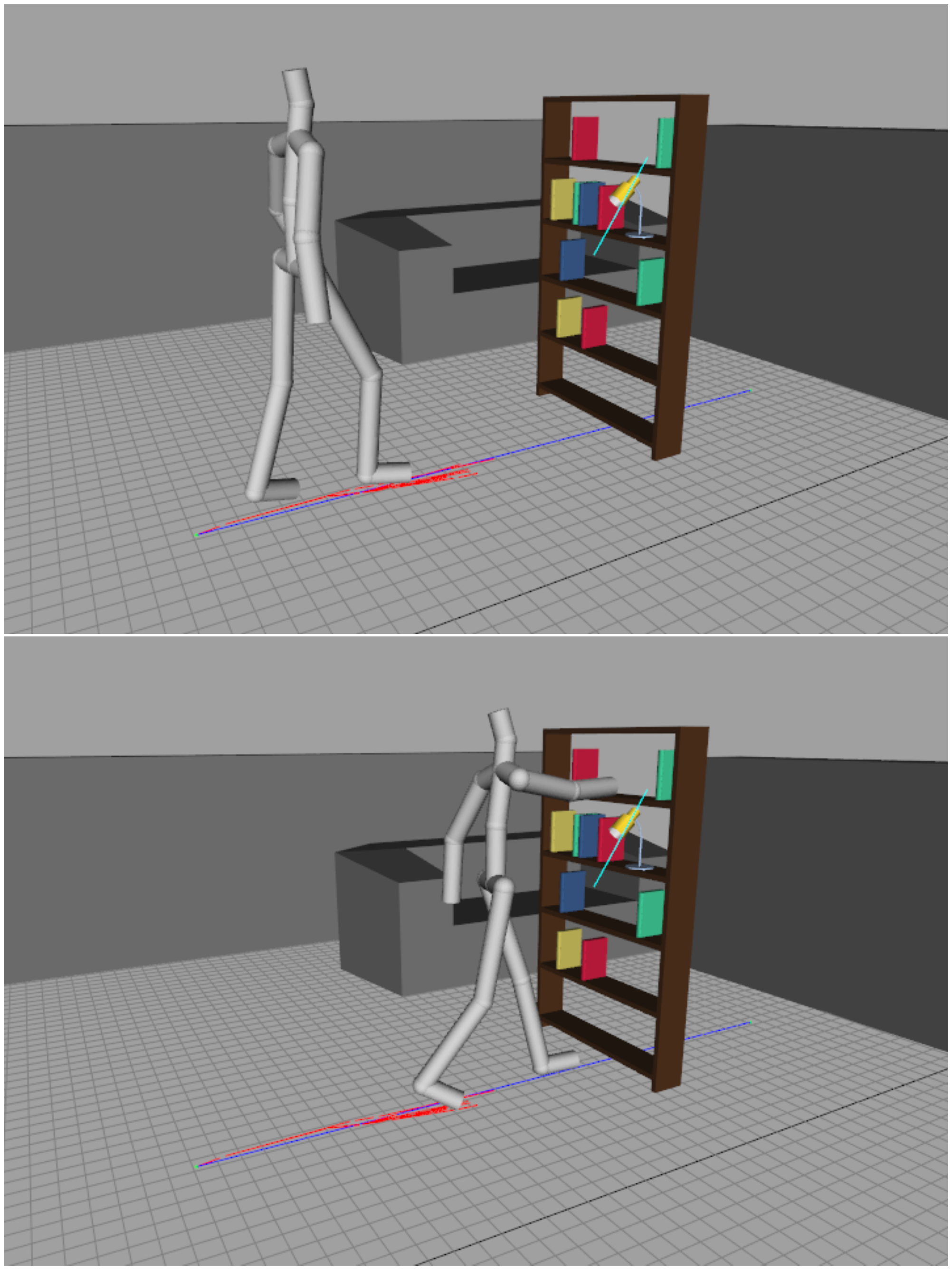

Figure 5.9: A book relocation example: the character begins by walking to a shelf (top) while using the locomotion skill, and then moving into the reaching skill to reach a book in order to pick it up from the shelf (bottom).

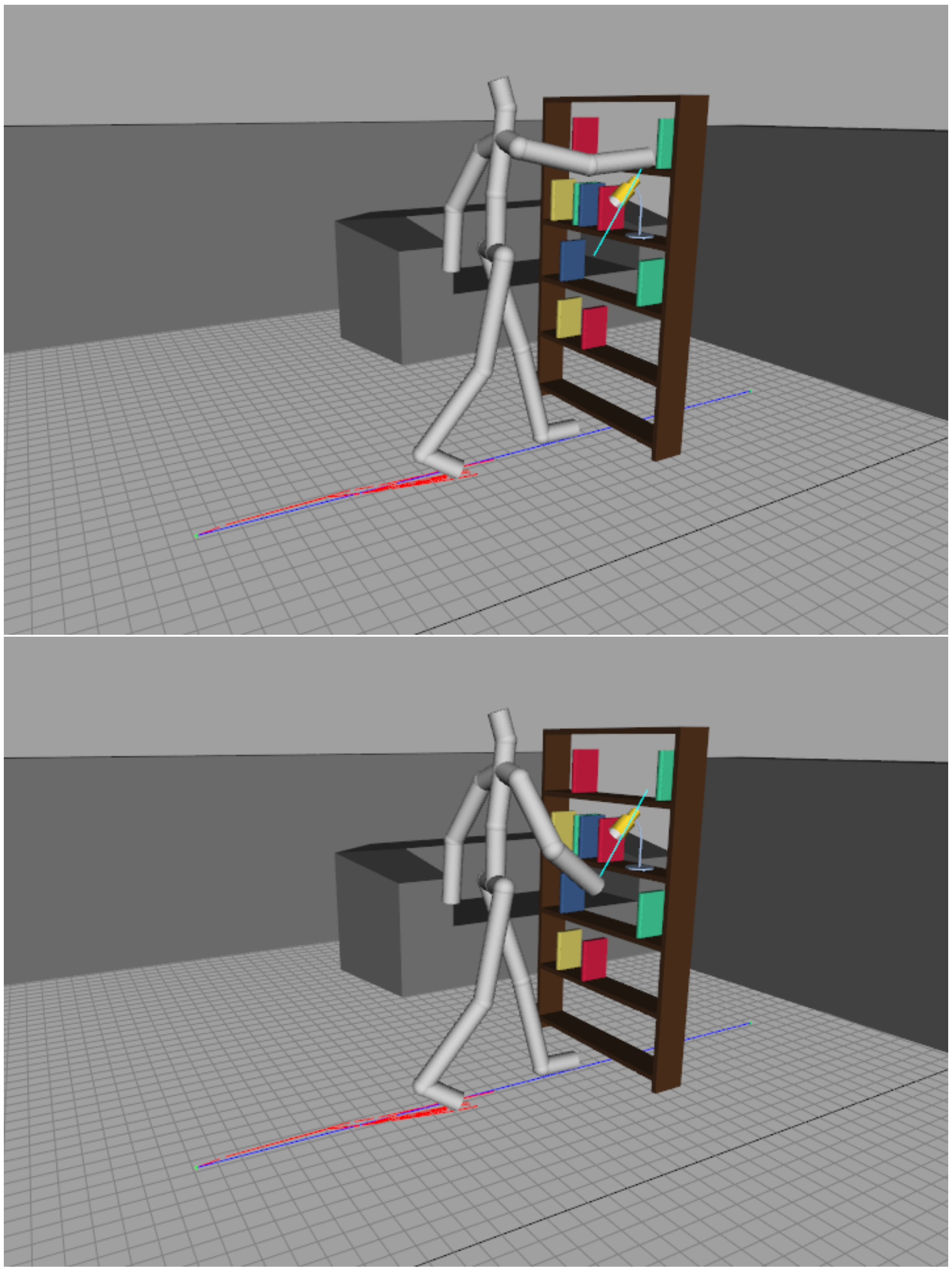

Figure 5.10: Continuation of the book example from Figure 5.9. The character is displacing the picked book (top) using the manipulation skill to a new location on the shelf and releasing his hand so that it can go back to locomotion mode (bottom).

$$\Sigma = \{\sigma_{loc}, \sigma_{tr}, \sigma_{displace}, \sigma_{reach}, \sigma_{release}\} \tag{5.11}$$

As in the door opening example the task of the character is to first approach the book shelf to a distance where it could reach a book, then pick a book from the shelf, move it to a new place on the shelf while avoiding other books and items on the shelf and finally, put the hand down similar to the release motion primitive. The character then may continue going back to the locomotion and perform different tasks.

The modes of this examples are the following:

$$\Theta = \{\theta_{loc}, \theta_{manip}, \theta_{tr}\} \tag{5.12}$$

Please note that the door mode $\theta_{door}$ is not in this list, and that the $\sigma_{displace}$ is within the manipulation mode $\theta_{manip}$. Next we list which motion primitive skills can be activated in the above mentioned modes.

$$\begin{aligned} \lambda(\theta_{loc}) &= \{\sigma_{loc}, \sigma_{tr}\} \\ \lambda(\theta_{tr}) &= \{\sigma_{tr}\} \\ \lambda(\theta_{manip}) &= \{\sigma_{reach}, \sigma_{release}, \sigma_{displace}\} \end{aligned} \tag{5.13}$$

and the function $\tau$ to define the mode transitions:

$$\begin{aligned} \tau(\theta_{loc}) &= \{\theta_{loc}, \theta_{tr}\} \\ \tau(\theta_{tr}) &= \{\theta_{manip}\} \\ \tau(\theta_{manip}) &= \{\theta_{manip}, \theta_{loc}\} \end{aligned} \tag{5.14}$$

With this configuration we could solve any book picking scenarios for various book locations or other obstacles that may be on the way of the character. Two examples are shown on Figure 5.10 and Figure 5.11.

### 5.6.3 Water Pouring

As we noticed in this first two scenarios, motion skills are very versatile and may be reused to solve different tasks. In this section we introduction the action skills which generate a new motion by blending a set of similar parametrized motions. This is particularly useful, for example where the character has to pour some liquid, point toward a target, dial a phone, kicking a ball or punch another character.

The action skill is a generic motion primitive skill which contains a database

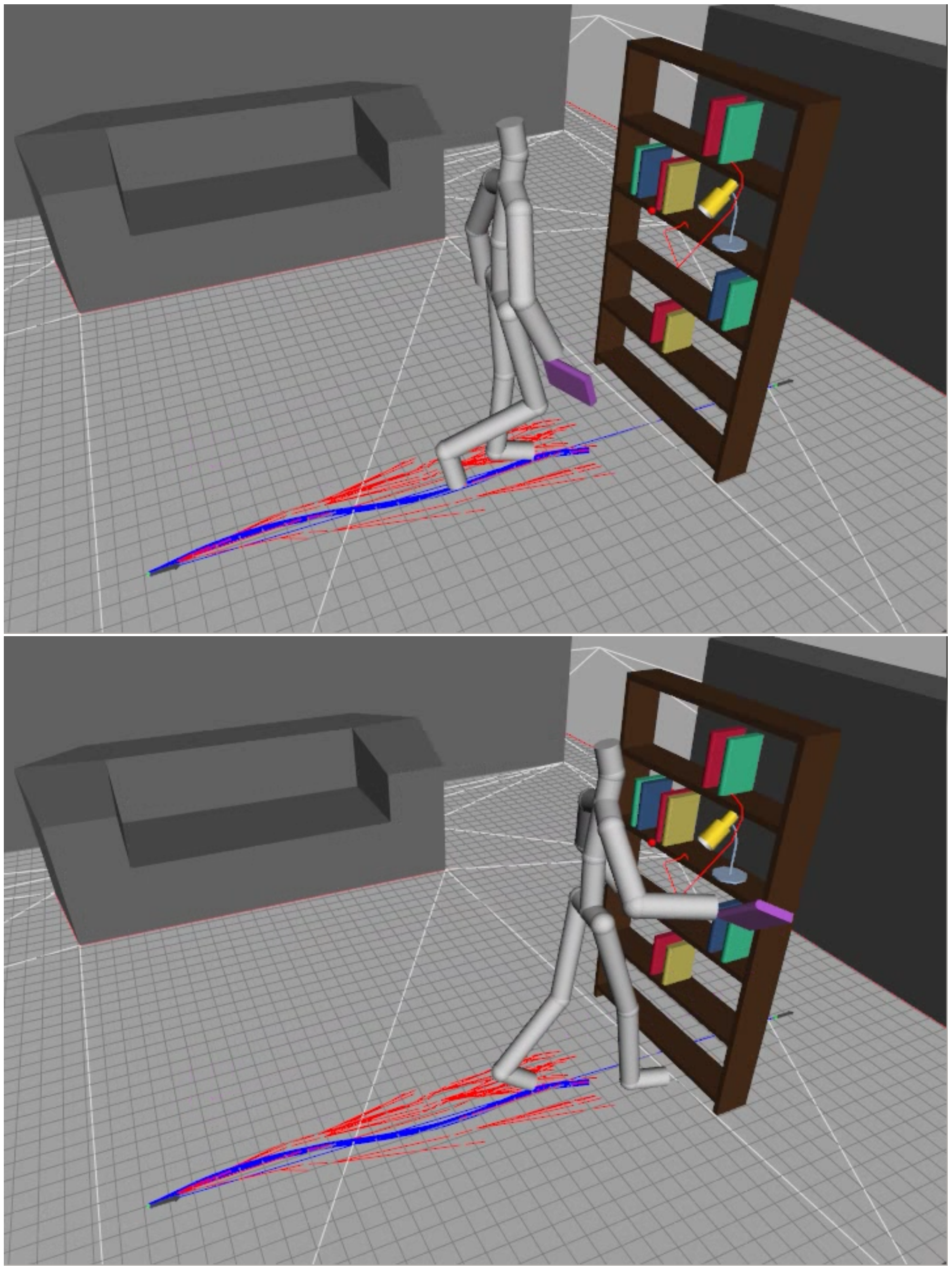

Figure 5.11: A book relocation example: the character begins by walking to a shelf (top) while holding a book using the locomotion skill, and then moves into the reaching skill to place the book on the shelf (bottom).

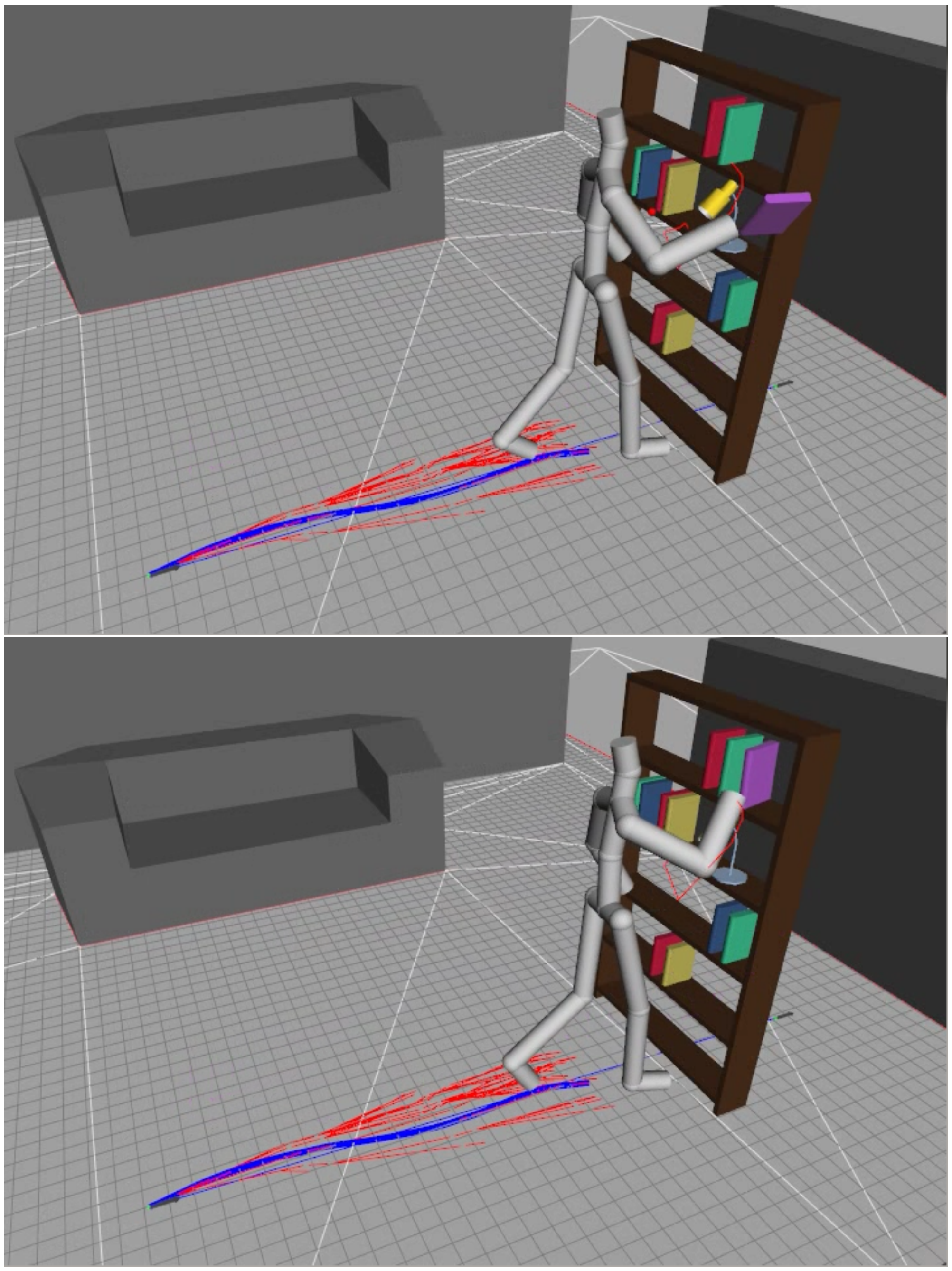

Figure 5.12: Continuation of the book example from Figure 5.11. The character is still within the reaching skill and continues to place the book on the shelf (top). The book is placed on the shelf and a switch to a manipulation skill is made to move the book to a new location (bottom).

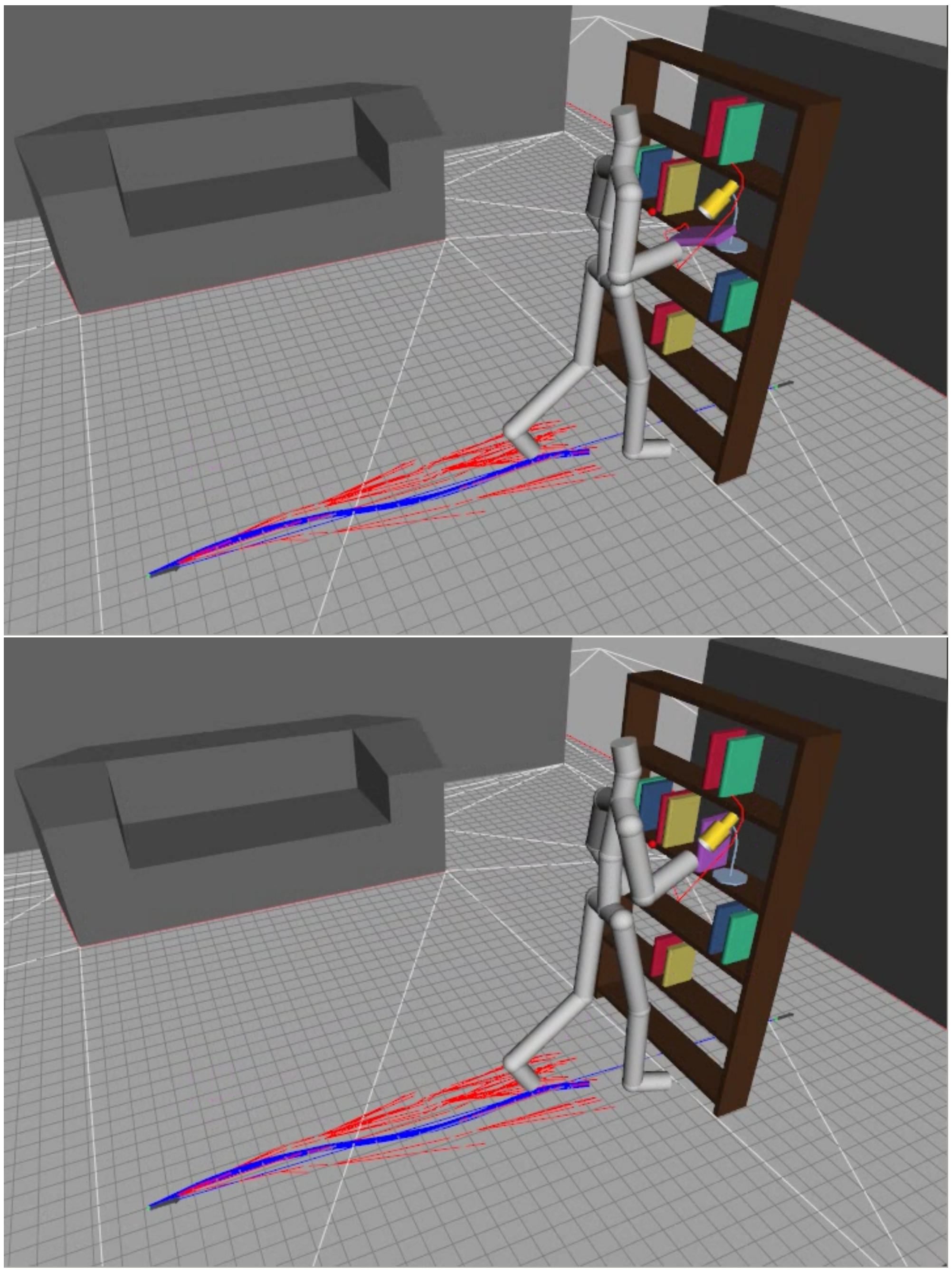

Figure 5.13: Continuation of the book example from Figure 5.12. The character is now replacing the book by carefully planning a motion such that it does not collide with other objects (top), the character safely puts the book on a new position on the shelf (bottom).

of similar actions. Given, this example set and a goal target position and orientation, this motion skill generates a motion which performs the given action where the characters end effector ends up at the given target constraints. The general planner may request motion which may be outside the expected cover space of the action skill, and in those events, the action skill reports a failure and does not generate a solution. In such a case, the planner would continue expanding various other locomotion skills until the action skill is well position to perform the given action.

We will not list the list of modes and motion skills and their corresponding functions as they are almost identical to the case of the book picking, where the $\sigma_{displace}$ motion primitive skill is replaced with the $\sigma_{water}$ which is an action motion primitive skill and contains a list of pouring examples. This skill implements and inverse blending mechanism, similar to that of [KG04] and [HMK11] in order to synthesize a desired motion which falls within the boundaries set by the examples. The action skill $\sigma_{water}$ controls the entire posture of the character and generates motions for the feet as well, however, the locomotion generated by the action skill, although parametrized, by design does not have a lot of variation.

Examples of this problem have the character walk with water glass approaching a table and pour some water to an empty cup on the table. After pouring the water, the character would release his hand and go back to a resting posture where it could go back to a locomotion motion primitives. The table may contain objects and the action skill makes sure that the generated motion does not collide with other objects. An example can be seen on Figure 5.14 and Figure 5.15.

## 5.7 Conclusion

We presented a multi-modal planner which combines parametrized motion primitives skill to achieve high quality motions in environments with many obstacles. The multi-modal planner used a locomotion skill based on our feature-based motion graph as its main locomotion skill. In addition, manipulation and action skills were used. These skills implemented randomized continuous algorithms or methods based on blending example motions. We showed that the by defining modes and assigning localized planners, we can search for complex motions which interact with the environment. Examples included opening and walking through doors, relocating a book on a shelf, and pouring water to a teapot on a table. The generated examples showed that allowing instances of motion primitives compete within a global multi-modal framework, well addresses the problem of generating high quality motions for human characters in complex environments.

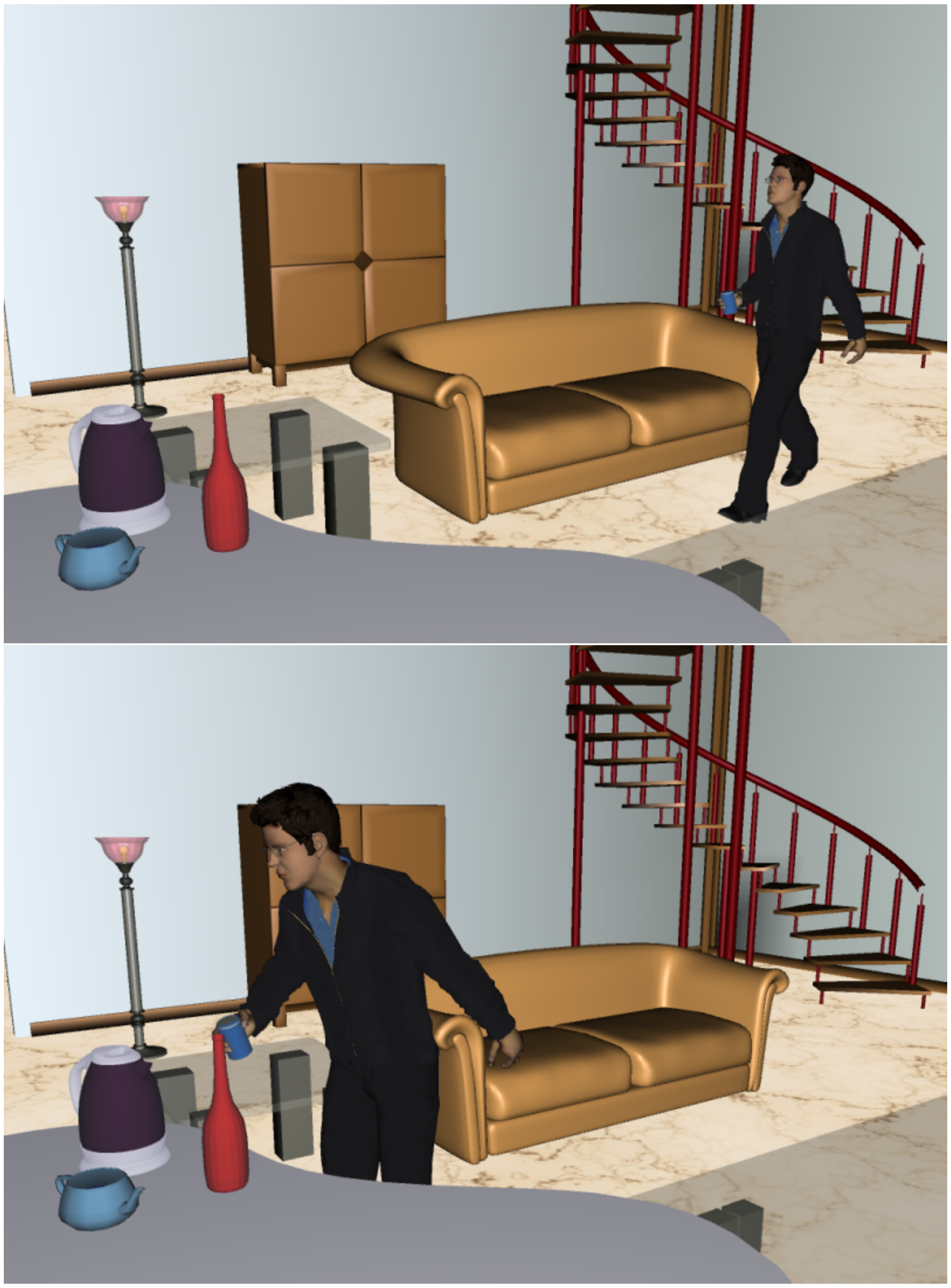

Figure 5.14: Water pouring example: *Top:* A frame from the locomotion skill bringing the character to an appropriate location to perform the action. *Bottom:* Character switches to the action mode and begins to pour the water. See Figure 5.15 for more examples.

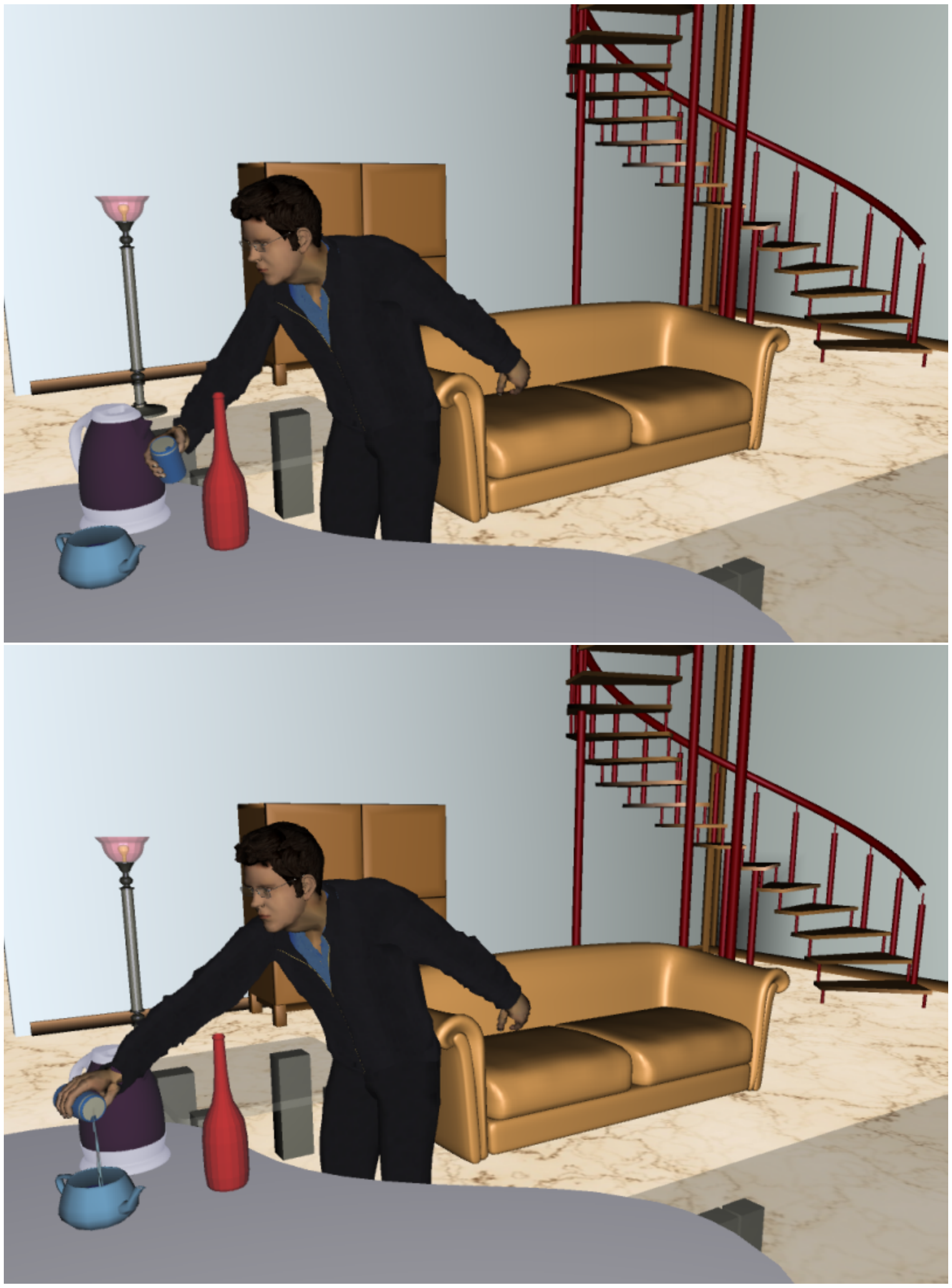

Figure 5.15: Continuing water pouring example from Figure 5.14. *Top:* The action skill ensures that the action motion does not collide with other objects. *Bottom:* The character pours water to inside the teapot on the table.

# CHAPTER 6

# Conclusion

Planning the motions of a virtual character with high fidelity and control is a difficult challenge. Striking a balance between these two competing properties makes the problem particularly complex to solve efficiently. Data-driven approaches have produced high quality results because of the inherent realism of human motion capture data, whereas planning algorithms have successfully solved general continuous problems with high degree of control. A great deal of research work has been dedicated to combining these approaches to increase the independent joint-level control and retain the realism and human-likeness of the motion capture data.

In this dissertation, we proposed a multi-modal planner, which takes a data-driven approach and combines it with local primitive skill planners to generate high quality motions for a virtual character. Our framework was designed to allow several parametrized motion primitive skills to compete with each other in contributing to the final solution. The local planners use inherent knowledge about the modes of the character and use methods suited to solve the task in these modes. This decoupling of the problem effectively reduces the dimensionality of the human character motion planning and allows synthesizing of complex motions in an environment with obstacles. Our framework generates motions for example scenarios such as: opening doors, relocating books in shelves and pouring water.

The proposed framework is dependent on the locomotion skill, which synthesized the most important part of human motion. This was done by using a feature-based motion graph which uses geometric features to automatically introduce structure into the motion graph. Such semantical structure not only improves the search procedure, but also reduces the time spent on the motion graph construction phase and avoids the feet sliding corrections. The locomotion skill uses a geometric feature which segments a walking motion into small walking cycles. This feature was shown to be quite versatile at segmenting different styles and variations of locomotion. The same walking feature segments both a straight or turning motions, as well as other motion types such as basketball or ballet. Features work with different moods as well; examples of happy, sad and tired motions were segmented successfully.

Besides automatically building a feature-based motion graph with semantical structure, we employed a triangulation-based search procedure for guiding our path following technique within the environment, making sure that the search does not get stuck in local minima. This was achieved by first triangulating the environment and finding a 2D path with clearance to the obstacles. This allowed for disabling of collision checking, which is usually a very time consuming component. Moreover our feature-based motion graph used a IK-based deformation model to continuously adjust the transitions of a search branch in order to precisely reach target goals. This proved essential for performing manipulation tasks.

Precomputation was also used with feature-based motion graphs. Due to the added structure of these graphs, precomputation of nodes of the graph allowed for real-time motion generation for human characters in an environment with many obstacles. We achieved this by precomputing motion maps for each nodes of the graph and employing a novel search algorithm, which used our motion maps to rapidly find a solution. The motion maps needed be only precomputed once, may be used for all subsequent queries, and could be shared by all human characters in the environment. Prior works used precomputation to improve the speed of manually-built motion graphs, primarily because unstructured motion graphs were not able to evenly cover the environment. In this dissertation, we showed that precomputing motion maps for feature-based motion graphs did not suffer from this problem. Our path following method made use of this fact and deployed motion maps to solve partial paths until the goal was reached. We showed that our method can search for a solution much faster than the state of the art methods and did not require any manual graph construction. Further, time complexity improvements where shown by the use of a parallelized version of our method that was able to play and search simultaneously. This made it possible for real-time motion generation of many human characters in a complex environment.

Our multi-modal data-driven planner made use of a locomotion skill, manipulation skill and action skill to generate human realistic and human-like motion. These skills implemented different local planners. The locomotion skill was based on a feature-based motion graph, whilst the manipulation and action skills used a randomized planner or a data-driven example-based planner. The skills were parametrized to achieve a wide range of motions; using our multi-modal planner, we were able to solve complex motions which involved interaction with the environment. Usually solving such examples requires a lot of motion preparation, however, with our method we were able to open doors and relocate books using a small set of motion capture data which only contained locomotion and a set of manipulation tasks. We showed that the parametrized nature of our framework allowed for many varieties of possible solutions, which was essential to solving tasks in an environment with obstacles. The locomotion skill was based on a feature-based motion graph. In this way, we were able to provide a wide range of possible placement options for the character. This was important for tasks such as opening doors or other manipulation tasks, because very often the availability of many placements for the character to perform manipulation is decisive in determining whether a problem may be solved. Moreover, we showed that by decoupling the problems into different modes, we introduce domain-specific knowledge about the problem. This constrains the sequence mode in which planners gets activated in order to synthesize the final solution. We showed that this is not only important to speed up the search, but it also allows for realistic motion synthesis. Our framework allowed different planning implementations for motion primitives within a mode, because different local planners might succeed in different situations. The mode decoupling was showed to help generate realistic and human-like results and effectively reduce the dimensionality of human motion planning.

The one important contribution of this dissertation was to show that multi-modal motion planning could be used with data-driven methods to synthesize high quality results confirming that multi-modal planning is an effective way of solving complex motions for human characters. In addition, we showed that feature-based motion graphs and pre-computation were important for a successful multi-modal planning based on parametrized motion skills. The work presented in this dissertation therefore indicates the benefits of decomposing a complex problem into subtasks that are individually solved by local planners. Most notably, this is a first framework which *automatically* generates motions from an unstructured database of human motion capture in real time among obstacles. Our results provide compelling evidence that a multi-modal approach can further advance the research on human motion planning for realistic and human-like characters. However, some limitations are worth noting, and we review some of our limitations in the next section.

## 6.1 Limitations

Although our feature-based motion graph was capable of automatically introducing semantical information about the motion database, synthesizing good results remain sensitive to the choice of the features used. If the wrong features are used, the motion might not be segmented properly, and the resulting graph might have poor connectivity. Additionally, the combined effect of large numbers of geometric features was not studied. A feature-based motion graph was used for the purpose of building a locomotion planner, and therefore experimentations with many geometric features were not done, because few features were sufficient to segment the motion for an efficient locomotion planner.

The main limitation of our precomputation method is the size of the motion maps. Because we computed motions maps for all the nodes of the motion graph, memory space usage was of a limiting factor. Although we showed that precomputation does not improve beyond a certain horizon, having a very large scale feature-based motion graph might prohibit the precomputation of all motion maps. In regards to the multi-modal planner, the main drawback involves coming up with good set of modes which lead to a good logical subdivision of the task. In addition, each mode needs a suitable local parametrized planner. If a problem does not have well-defined modes and a list of well represented parametrized local planners, a solution might become difficult to find. However modes transitions need only be defined once per set of problems. Animators can easily build libraries of such modes and their transitions and then use these to form motion primitive skills at higher levels.

## 6.2 Future Work

In this dissertation, we did elaborate analysis of locomotion with geometric features to see the effect of choosing transitions at points segmented by features. A future avenue in improving feature-based motion graph could involve learning and automatically devising geometric features which improve the coverage, connectivity and interactivity of the feature-based motion graph. This could be achieved by an optimization procedure where a set of features are optimized to yield the best feature-based graphs in terms of multi-modal data-driven motion planning for human characters in environments with many obstacles.

Precomputation, as presented in this dissertation, is only done within a single motion primitive skill. Future avenues may include precomputation within multiple modes involving many motion primitive skills. This would further speed up the planning and may also give rise to different motion primitives specific to exploiting this level of precomputation. Moreover precomputation could be improved by more research on coverage analysis of the motion capture database. Evaluating the coverage of the motion capture within the environment would be highly useful in the context of multi-modal data-driven planning. This would be similar to establishing the convergence rate of a sampling-based planning method. It would be highly beneficial to an animator to be given bounds on the probability of a problem being solved, given a set of motion capture examples and set of motion primitive skills. Such guarantees may help in deciding what set of motions and motion skills could be considered sufficient to solve a given problem. Other improvements might be achieved by finding a compression mechanism for motion maps since they are limited by the size of the available computer memory.

In this dissertation, we solved door-opening, book-relocation tasks by subdividing these problems into simpler tasks. It should be possible for a multi-modal planner, however, to use motion primitive skills at lower levels in order to solve tasks at higher levels. Examples of these include: walking down the corridor, entering a room by opening a door, picking up a book from the shelf, exiting the room, walking down the corridor, entering another room, and leaving the book on a table. At such a level, opening a door and picking up a book might be used as one single atomic parametrized motions skill. In this manner, a hierarchical multi-modal planner with finer motion primitives might be more similar to how humans solve real-life everyday tasks. This could be further advanced by using learning to improve results from past examples. New motion primitives and modes may be learned to better decompose a problem. Learning may also involve tuning the motion primitive skill planners but, in addition, it may also be used as way of returning motions which are already solved or are close enough that they might be deformed to fit the required constraints.

# References

[AF02] Okan Arikan and David A. Forsyth. "Synthesizing Constrained Motions from Examples." *Proceedings of SIGGRAPH*, **21**(3):483–490, 2002.

[AFO03] Okan Arikan, David A. Forsyth, and James F. O'Brien. "Motion synthesis from annotations." *Proceedings of SIGGRAPH*, **22**(3):402–408, 2003.

[ALH08] Gustavo Arechavaleta, Jean-Paul Laumond, H. Hicheur, and A. Berthoz. "An Optimality Principle Governing Human Walking." *IEEE Transactions on Robotics*, **24**(1):5–14, 2008.

[BB04] Paolo Baerlocher and Ronan Boulic. "An inverse kinematics architecture enforcing an arbitrary number of strict priority levels." *Visual Computer*, **20**(6):402–417, August 2004.

[BB05] B. Burns and O. Brock. "Single-Query Entropy-Guided Path Planning." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2124–2129, 2005.

[BC12] Benjamin Balaguer and Stefano Carpin. "Bimanual regrasping from unimanual machine learning." In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 3264–3270, 2012.

[BCP08] Philippe Beaudoin, Stelian Coros, Michiel van de Panne, and Pierre Poulin. "Motion-Motif Graphs." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 117–126, 2008.

[BDN07] Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. "Grasp Planning in Complex Scenes." In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, December 2007.

[BE09] Ben J.H. van Basten and Arjan Egges. "Path Abstraction for Combined Navigation and Animation." In *Motion in Games*, pp. 182–193, 2009.

[BE11] Ben J.H. van Basten and Arjan Egges. "Flexible Splicing of Upper-Body Motion Spaces on Locomotion." *Computer Graphics Forum*, **30**(7):1963–1971, 2011.

[BEG11] Ben J.H. van Basten, Arjan Egges, and Roland Geraerts. "Combinining Path Planners and Motion Graphs." *Computer Animation and Virtual Worlds*, **21**:1–22, 2011.

[BFK06] Jur van den Berg, Dave Ferguson, and James Kuffner. "Anytime path planning and replanning in dynamic environments." In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 2366–2371, 2006.

[BK00] R. Bohlin and Lydia Kavraki. "Path planning using lazy PRM." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pp. 521–528, 2000.

[BSL12] Yunfei Bai, Kristin Siu, and Karen Liu. "Synthesis of concurrent object manipulation tasks." *ACM Trans. Graph.*, **31**(6):156:1–156:9, November 2012.

[BSP04] Jernej Barbič, Alla Safonova, Jia-Yu Pan, Christos Faloutsos, Jessica K. Hodgins, and Nancy S. Pollard. "Segmenting motion capture data into distinct behaviors." In *Proceedings of Graphics Interface (GI)*, pp. 185–194, 2004.

[Can88] John F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, USA, 1988.

[CHP07] Seth Cooper, Aaron Hertzmann, and Zoran Popović. "Active learning for real-time motion controllers." In *Proceedings of SIGGRAPH*, 2007.

[CK04] Joel Chestnutt and James Kuffner. "A Tiered Planning Strategy for Biped Navigation." In *Proceedings of the IEEE - RAS/RSJ Conference on Humanoid Robots*, November 2004.

[CKH11] Myung Geol Choi, Manmyung Kim, Kyunglyul Hyun, and Jehee Lee. "Deformable Motion: Squeezing into Cluttered Environments." *Comput. Graph. Forum*, **30**(2):445–453, 2011.

[CLS02] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. "Planning Biped Locomotion using Motion Capture Data and Probabilistic Roadmaps." *Proceedings of SIGGRAPH*, **22**(2):182–203, 2002.

[CP02] Stefano Carpin and Enrico Pagello. "On Parallel RRTs for Multi-robot Systems." In *Proceedings of the 8th Conference on Italian Association for Artificial Intelligence*, pp. 834–841, 2002.

[EAP06] Claudia Esteves, Gustavo Arechavaleta, Julien Pettré, and Jean-Paul Laumond. "Animation Planning for Virtual Characters Cooperation." *ACM Transaction on Graphics*, **25**(2):319–339, 2006.

[Egg08a] Arjan Egges. "Analysis of Human Navigation and Manipulation Motions." In *Proceedings of Measuring Behavior*, August 2008.

[Egg08b] Arjan Egges. "Opening Doors in Motion Analysis Research." In *Motion in Games*, pp. 188–199, 2008.

[FPT01] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. "Composable controllers for physics-based character animation." In *Proceedings of SIGGRAPH*, pp. 251–260, 2001.

[FXG12] Rukun Fan, Songhua Xu, and Weidong Geng. "Example-Based Automatic Music-Driven Conventional Dance Motion Synthesis." *IEEE Transactions on Visualization and Computer Graphics*, **18**(3):501–515, March 2012.

[Ger10] Roland Geraerts. "Planning short paths with clearance using explicit corridors." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1997–2004, 2010.

[GO07] Roland Geraerts and Mark Overmars. "Reachability-based analysis for Probabilistic Roadmap Planners." *Robotics and Autonomous Systems*, **55**(11):824–836, November 2007.

[GSK03] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. "Snap-together motion: assembling run-time animations." In *Proceedings of the Symposium on Interactive 3D graphics and Games (I3D)*, pp. 181–188, 2003.

[HBH06] Kris Hauser, T. Bretl, K. Harada, and Jean-Claude Latombe. "Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots." In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, pp. 2641– 2648, July 2006.

[HG07] Rachel Heck and Michael Gleicher. "Parametric Motion Graphs." In *Proceedings of the Symposium on Interactive 3D Graphics and Games (I3D)*, pp. 129–136, 2007.

[HKG06] Rachel Heck, Lucas Kovar, and Michael Gleicher. "Splicing Upper-Body Actions with Locomotion." In *Proceedings of Eurographics*, September 2006.

[HLM99] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. "Path Planning in Expansive Configuration Spaces." *International Journal of Computational Geometry and Applications*, **9**(4/5):495–512, 1999.

[HMK11] Yazhou Huang, Mentar Mahmudi, and Marcelo Kallmann. "Planning Humanlike Actions in Blending Spaces." In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2653–2659, 2011.

[HN11] Kris Hauser and Victor Ng-Thow-Hing. "Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task." *International Journal on Robotic Research*, **30**(6):678–698, 2011.

[HNG07] Kris Hauser, Victor Ng-Thow-Hing, and Héctor H. González-Baños. "Multi-modal Motion Planning for a Humanoid Robot Manipulation Task." In *International Symposium on Robotics Research*, pp. 307–317, 2007.

[HWB95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. "Animating Human Athletics." In *Proceedings of SIGGRAPH*, pp. 71–78, 1995.

[IAF07] Leslie Ikemoto, Okan Arikan, and David Forsyth. "Quick transitions with cached multi-way blends." In *Proceedings of the Symposium on Interactive 3D Graphics and Games (I3D)*, pp. 145–151, 2007.

[ICD05] Y.P. Ivanenko, G. Cappellini, N. Dominici, R.E. Poppele, and F. Lacquaniti. "Coordination of locomotion with voluntary movements in humans." *Journal of Neuroscience*, **25**(31):7238–7253, 2005.

[KAA03] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. "Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping." In *Proceedings of Eurographics*, volume 22, pp. 313–322, 2003.

[Kal10] Marcelo Kallmann. "Shortest Paths with Arbitrary Clearance from Navigation Meshes." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2010.

[KBM04] Marcelo Kallmann, Robert Bargmann, and Maja J. Matarić. "Planning the Sequencing of Movement Primitives." In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAAB)*, pp. 193–200, July 2004.

[KG03] Lucas Kovar and Michael Gleicher. "Flexible automatic motion blending with registration curves." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 214–224, 2003.

[KG04] Lucas Kovar and Michael Gleicher. "Automated Extraction and Parameterization of Motions in Large Datasets." *Proceedings of SIGGRAPH*, **23**(3):559–568, 2004.

[KGP02] Lucas Kovar, Michael Gleicher, and Frederic H. Pighin. "Motion Graphs." *Proceedings of SIGGRAPH*, **21**(3):473–482, 2002.

[KHB10] Marcelo Kallmann, Yazhou Huang, and Robert Backman. "A Skill-Based Motion Planning Framework for Humanoids." In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010.

[KHK09] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. "Synchronized multi-character motion editing." *ACM Transactions on Graphics*, **28**(3):1–9, 2009.

[KKK94] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. "Planning Motions with Intentions." In *Proceedings of SIGGRAPH*, pp. 395–408, 1994.

[KL00a] James Kuffner and Jean-Claude Latombe. "Interactive manipulation planning for animated characters." In *Proceedings of Pacific Graphics*, October 2000. poster paper.

[KL00b] James Kuffner and Jean-Claude Latombe. "Interactive manipulation planning for animated characters." In *Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications*, pp. 417–418, 2000.

[KL00c] James Kuffner and Steven M. LaValle. "RRT-Connect: An Efficient Approach to Single-Query Path Planning." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, April 2000.

[KLY11] Oussama Kanoun, Jean-Paul Laumond, and Eiichi Yoshida. "Planning foot placements for a humanoid robot: A problem of inverse kinematics." *International Journal on Robotic Research*, **30**(4):476–485, 2011.

[KM04] Marcelo Kallmann and Maja Matarić. "Motion Planning Using Dynamic Roadmaps." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4399–4404, April 2004.

[KNK03] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. "Motion Planning for Humanoid Robots." In *Proceedings of the 11th International Symposium of Robotics Research (ISRR)*, November 2003.

[KS05] Taesoo Kwon and Sung Yong Shin. "Motion modeling for online locomotion synthesis." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 29–38, 2005.

[KSG02] Lucas Kovar, John Schreiner, and Michael Gleicher. "Footskate Cleanup for Motion Capture Editing." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 97–104, 2002.

[KSL96] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. "Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces." *IEEE Transactions on Robotics and Automation*, **12**:566–580, 1996.

[KTW10] Björn Krüger, Jochen Tautges, Andreas Weber, and Arno Zinke. "Fast Local and Global Similarity Searches in Large Motion Capture Databases." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 1–10, July 2010.

[Lam09] Fabrice Lamarche. "TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints." *Computer Graphics Forum*, **28**(2), March 2009.

[Las87] John Lasseter. "Principles of Traditional Animation applied to 3D computer animation." In *Proceedings of SIGGRAPH*, pp. 35–44, 1987.

[Lat90] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publisher, December 1990.

[Lav98] Steven M. Lavalle. "Rapidly-Exploring Random Trees: A New Tool for Path Planning." Technical Report 98-11, Iowa State University, October 1998.

[LaV06] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[LB11] Y. Li and Kostas Bekris. "Learning Approximate Cost-to-Go Metrics To Improve Sampling-based Motion Planning." In *Proceedings of International Conference on Robotics and Automation (ICRA)*, May 2011.

[LCL06] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. "Motion patches: building blocks for virtual environments annotated with motion data." *ACM Transactions on Graphics*, **25**(3):898–906, 2006.

[LCR02] Jehee Lee, Jinxiang Chai, Paul Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. "Interactive Control of Avatars Animated with Human Motion Data." *Proceedings of SIGGRAPH*, **21**(3):491–500, July 2002.

[LH02] Peter Leven and Seth Hutchinson. "A Framework for Real-time Path Planning in Changing Environments." *The International Journal of Robotics Research*, **21**(12):999–1030, 2002.

[Liu05] Karen Liu. *Towards a Generative Model of Natural Motion.* PhD thesis, University of Washington, 2005.

[Liu08] Karen Liu. "Synthesis of Interactive Hand Animation." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, July 2008.

[LK01] Steven M. LaValle and James Kuffner. "Randomized Kinodynamic Planning." *International Journal of Robotic Research*, **20**(5):378–400, 2001.

[LK05] Manfred Lau and James Kuffner. "Behavior Planning for Character Animation." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 271–280, August 2005.

[LK06] Manfred Lau and James Kuffner. "Precomputed search trees: planning for interactive goal-driven animation." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 299–308, 2006.

[LK10] Manfred Lau and James Kuffner. "Scalable Precomputed Search Trees." In *Motion in Games*, volume 6459, pp. 70–81, 2010.

[LKN09] Pengcheng Luo, Michael Kipp, and Michael Neff. "Augmenting Gesture Animation with Motion Capture Data to Provide Full-Body Engagement." In *Intelligent Virtual Agents (IVA)*, pp. 405–417, 2009.

[LL04] Jehee Lee and Kang Hoon Lee. "Precomputing avatar behavior from human motion data." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 79–87, 2004.

[LLK11] Sergey Levine, Yongjoon Lee, Vladlen Koltun, and Zoran Popović. "Space-time planning with parameterized locomotion controllers." *ACM Transactions on Graphics (TOG)*, **30**(3), May 2011.

[LP02] Karen Liu and Zoran Popović. "Synthesis of complex dynamic character motion from simple animations." In *Proceedings of SIGGRAPH*, pp. 408–416, 2002.

[LP10] Seong Jae Lee and Zoran Popović. "Learning behavior styles with inverse reinforcement learning." *ACM Transactions on Graphics*, **29**(4):122:1–122:7, July 2010.

[LS09] Sanjeev Khanna Liming Zhao, Aline Normoyle and Alla Safonova. "Automatic Construction of a Minimum Size Motion Graph." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2009.

[LWB10] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. "Motion fields for interactive character locomotion." *ACM Transactions on Graphics*, **29**(6):138:1–138:8, December 2010.

[LWS02] Yan Li, Tian-Shu Wang, and Heung-Yeung Shum. "Motion texture: a two-level statistical model for character motion synthesis." *Proceedings of SIGGRAPH*, **21**(3):465–472, 2002.

[MC12] Jianyuan Min and Jinxiang Chai. "Motion graphs++: a compact generative model for semantic motion analysis and synthesis." *ACM Trans. Graph.*, **31**(6):153:1–153:12, November 2012.

[MCC09] Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. "Interactive Generation of Human Animation with Deformable Motion Models." *ACM Transactions on Graphics*, **29**(1):9:1–9:12, December 2009.

[MJC01] M. Mizuguchi, Buchanan J., and T. Calvert. "Data driven motion transitions for interactive games." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2001.

[MK11] Mentar Mahmudi and Marcelo Kallmann. "Feature-Based locomotion with Inverse Branch Kinematics." In *Proceedings of the 4th international conference on Motion in Games (MIG)*, pp. 39–50, November 2011.

[MK12] Mentar Mahmudi and Marcelo Kallmann. "Precomputed Motion Maps for Unstructured Motion Capture." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 127–136, July 2012.

[MK13] Mentar Mahmudi and Marcelo Kallmann. "Analyzing Locomotion Synthesis with Feature-Based Motion Graphs." *IEEE Transcations on Visualization and Computer Graphics*, **19**(5):774–786, May 2013.

[MR06] Meinard Müller and Tido Röder. "Motion templates for automatic classification and retrieval of motion capture data." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 137–146, 2006.

[MRC05] Meinard Müller, Tido Röder, and Michael Clausen. "Efficient content-based retrieval of motion capture data." In *Proceedings of SIGGRAPH*, pp. 677–685, 2005.

[NK00] C.L. Nielsen and Lydia Kavraki. "A two level fuzzy PRM for manipulation planning." In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pp. 1716–1721, 2000.

[NK09] Michael Neff and Yejin Kim. "Interactive editing of motion style using drives and correlations." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 103–112, 2009.

[PB02] Katherine Pullen and Christoph Bregler. "Motion Capture Assisted Animation: Texturing and Synthesis." *Proceedings of SIGGRAPH*, pp. 501–508, 2002.

[PBC05] Erion Plaku, Kostas Bekris, B.Y. Chen, A.M. Ladd, and Lydia Kavraki. "Sampling-Based Roadmap of Trees for Parallel Motion Planning." *IEEE Transactions on Robotics*, **21**(4):597–608, 2005.

[PKM06] Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen. "Developing navigation behavior through self-organizing distinctive-state abstraction." *Connection Science*, **18**(2):159–172, 2006.

[PKV07] Erion Plaku, Lydia Kavraki, and M. Vardi. "Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning." In *Proceedings of Robotics: Science and Systems*, June 2007.

[PLT05] Julien Pettre, Jean-Paul Laumond, and Daniel Thalmann. "A Navigation Graph for Real-Time Crowd Animation on Multilayered and Uneven Terrain." In *Proceedings of the First International Workshop on Crowd Simulation (V-CROWDS)*, 2005.

[PSS02] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. "On-line locomotion generation based on motion blending." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 105–111, 2002.

[PZL10] Jia Pan, Liangjun Zhang, Ming Lin, and Dinesh Manocha. "A Hybrid Approach for Synthesizing Human Motion in Constrained Environments." In *Conference on Computer Animation and Social Agents (CASA)*, 2010.

[RBC98] Charles Rose, Bobby Bodenheimer, and Michael F. Cohen. "Verbs and Adverbs: Multidimensional Motion Interpolation." *IEEE Computer Graphics and Applications*, **18**:32–40, 1998.

[Rei79] John H. Reif. "Complexity of the movers problem and generalizations." *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pp. 421–427, 1979.

[RP07] Paul S. A. Reitsma and Nancy S. Pollard. "Evaluating motion graphs for character animation." *ACM Trans. Graph.*, **26**(4), October 2007.

[RZS10] Cheng Ren, Liming Zhao, and Alla Safonova. "Human Motion Synthesis with Optimization-based Graphs." *Computer Graphics Forum*, **29**(2):545–554, 2010.

[SB02] Cyrill Stachniss and Wolfram Burgard. "An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments." In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 508–513, 2002.

[SH07] Alla Safonova and Jessica K. Hodgins. "Construction and optimal search of interpolated motion graphs." *Proceedings of SIGGRAPH*, **26**(3), August 2007.

[SH08] Alla Safonova and Jessica K. Hodgins. "Synthesizing Human Motion from Intuitive Constraints." In *Artificial Intelligence Techniques for Computer Graphics*, pp. 15–39, 2008.

[SK04] Michael Stilman and James Kuffner. "Navigation Among Movable Obstacles: Real-time Reasoning in Complex Environments." In *Proceedings of the IEEE International Conference on Humanoid Robotics (Humanoids)*, volume 1, pp. 322–341, December 2004.

[SKF07] Ari Shapiro, Marcelo Kallmann, and Petros Faloutsos. "Interactive Motion Correction and Object Manipulation." In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, April 2007.

[SKG05] Makyu Sung, Lucas Kovar, and Michael Gleicher. "Fast and Accurate Goal-directed Motion Synthesis for Crowds." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, July 2005.

[SL01] G. Sanchez and Jean-Claude Latombe. "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking." In *International Symposium on Robotics Research (ISRR)*, November 2001.

[SMM05] Madhusudhanan Srinivasan, Ronald A. Metoyer, and Eric N. Mortensen. "Controllable real-time locomotion using mobility maps." In *Proceedings of Graphics Interface 2005*, pp. 51–59, 2005.

[SO06] Hyun Joon Shin and Hyun Seok Oh. "Fat graphs: constructing an interactive character with continuous controls." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 291–298, 2006.

[Tar72] Robert Endre Tarjan. "Depth-First Search and Linear Graph Algorithms." *Siam Journal on Computing*, **1**(2):146–160, 1972.

[TGB00] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. "Real-time inverse kinematics techniques for anthropomorphic limbs." *Graph. Models Image Process.*, **62**(5):353–388, September 2000.

[TLP07] A. Treuille, Y. Lee, and Z. Popović. "Near-optimal Character Animation with Continuous Control." In *Proceedings of SIGGRAPH*, 2007.

[TS00] Kurt A. Thoroughman and Reza Shadmehr. "Learning of action through adaptive combination of motor primitives." *Nature*, **407**:742–747, 2000.

[WB03] Jing Wang and Bobby Bodenheimer. "An evaluation of a cost metric for selecting transitions between motion segments." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 232–238, 2003.

[WB04] Jing Wang and Bobby Bodenheimer. "Computing the duration of motion transitions: an empirical approach." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 335–344, 2004.

[WBE10] H. van Welbergen, Ben J.H. van Basten, Arjan Egges, Zs. M. Ruttkay, and Mark Overmars. "Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control." *Computer Graphics Forum*, **29**(8):2530–2554, 2010.

[WC91] L.-C.T. Wang and C.C. Chen. "A combined optimization method for solving the inverse kinematics problem of mechanical manipulators." *IEEE Transactions on Robotics and Automation*, **7**(4):489–499, 1991.

[WH97] D. J. Wiley and J. K. Hahn. "Interpolation Synthesis of Articulated Figure Motion." *IEEE Computer Graphics and Applications*, **17**(6):39–45, 1997.

[WJM06] Pawel Wrotek, Odest Chadwicke Jenkins, and Morgan McGuire. "Dynamo: dynamic, data-driven character control with adjustable balance." In *Proceedings of the ACM SIGGRAPH Symposium on Videogames*, pp. 61–70, 2006.

[WK88] Andrew Witkin and Michael Kass. "Spacetime Constraints." In *Proceedings of SIGGRAPH*, pp. 159–168, 1988.

[WP95] Andrew Witkin and Zoran Popović. "Motion Warping." In *Proceedings of SIGGRAPH*, pp. 105–108, 1995.

[WP10] Jia-chi Wu and Zoran Popović. "Terrain-Adaptive Bipedal Locomotion Control." *ACM Transactions on Graphics*, **29**(4):72:1–72:10, July 2010.

[YKH04] Katsu Yamane, James Kuffner, and Jessica K. Hodgins. "Synthesizing Animations of Human Manipulation Tasks." *Proceedings of SIGGRAPH*, **23**(3):532–539, 2004.

[ZLX12] Dan Zong, Chunpeng Li, Shihong Xia, and Zhaoqi Wang. "Planning interactive task for intelligent characters." *Computer Animation and Virtual Worlds*, **23**(6):547–557, 2012.

[ZS08] Liming Zhao and Alla Safonova. "Achieving Good Connectivity in Motion Graphs." In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 127–136, July 2008.