INTERACTIVE TECHNIQUES FOR MOTION DEFORMATION OF ARTICULATED FIGURES USING PRIORITIZED CONSTRAINTS

THÈSE N^{O} 3459 (2006)

PRÉSENTÉE LE 10 MARS 2006

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
Laboratoire de réalité virutelle

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Benoît LE CALLENNEC

DEA en informatique, Université de Rennes I, France
et de nationalité française

acceptée sur proposition du jury:

Prof. C. Petitpierre, président du jury
Dr R. Boulic, directeur de thèse
Dr K. Aminian, rapporteur
Dr J.-P. Laumond, rapporteur
Prof. D. Thalmann, rapporteur
Dr J.-P. Verriest, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2006

CONTENTS

Remerciements

Cette thèse a été financée par le Fonds National Suisse de la Recherche Scientifique, et effectuée au laboratoire de Réalité Virtuelle (VRLab) de l'école polytechnique fédérale de Lausanne (EPFL). Je tiens tout d'abord à remercier sincèrement mon directeur de thèse, Ronan Boulic, pour m'avoir encadré avec autant de patience et de disponibilité. Je tiens aussi à le remercier pour avoir su me soutenir et m'encourager dans les moments difficiles. Je remercie aussi le professeur Daniel Thalmann, directeur du laboratoire, qui m'a permis de travailler principalement sur ce projet. Je remercie aussi les membres du jury de thèse, le professeur Jean-Paul Laumond, le professeur Jean-Pierre Verriest, le professeur Kamiar Aminian ainsi que le professeur Daniel Thalmann pour le soin qu'ils ont apporté à l'examen de cette thèse. Enfin, je remercie le professeur Claude Petitpierre pour avoir présidé ce jury. Je souhaite remercier Pascal Glardon, avec qui j'ai discuté, des heures durant, d'animation de personnages et en particulier de la façon dont un humain posait les pieds par terre perdant ainsi toute crédibilité aux yeux de nos collègues. J'aimerais aussi remercier Thierry Michelot, Nicolas Elsig et Mireille Clavien pour avoir modélisé les nombreux personnages que j'ai torturés. Je remercie Helena Grillon pour avoir entièrement relu cette thèse et pour m'avoir permis de la rendre beaucoup plus compréhensible. Je remercie Julien Pettre pour ses diverses critiques sur les versions préliminaires de ce manuscrit, Lorna Herda et Raquel Urtasun pour nos conversations techniques que je n'ai toujours pas comprises ainsi que les admins Jan Ciger et Etienne de Sevin pour m'avoir expliqué en particulier que ça ne s'appelait pas "mulot" mais "souris". Sur un plan plus personnel, je souhaite remercier Ronan pour avoir été quelqu'un de très accessible et patient avec moi, Pascal, Anderson, Pablo et Schubert pour nos soirées philosophie et poésie à Satellite, Nicolas, Sylvie, Thierry, Ali et Mehdi pour leur amitié ainsi que Helena, Barbichette et Jojo (les petits nouveaux) pour leur jovialité et leur sympathie. Je souhaite aussi remercier tous mes amis Bretons pour m'avoir hébergé lors de mes retours au pays et pour m'avoir accueilli à coups de galettes et de kouign amann. Je souhaiterais remercier ma mère ainsi que Jacquounet pour m'avoir toujours soutenu dans mes choix et m'avoir permis de mener à bien mes projets. Enfin, je souhaiterais remercier Benoît Le Callennec sans qui, rien de cela n'aurait été possible.

Résumé

Depuis déjà plusieurs années, animer des humains virtuels de manière convaincante est devenu extrêmement demandé dans plusieurs domaines de l'industrie. Dans les jeux vidéos par exemple, les humains virtuels sont souvent les personnages principaux. S'ils n'étaient pas animés de manière réaliste, alors tous les efforts faits pour donner aux joueurs un sentiment d'immersion seraient ruinés. Parallèlement, les films générés par ordinateur sont devenus très populaires et ont donc augmenté les exigences en terme de qualité d'animation. En effet, les humains virtuels sont maintenant devenus les nouvelles stars de films comme Final Fantasy ou Shrek et sont parfois même utilisés pour créer des effets spéciaux dans des films comme Matrix. Dans ce contexte, les animations des humains virtuels ne doivent plus uniquement être réalistes, comme dans le cas des jeux vidéos, mais doivent de plus être expressives comme pour de vrais acteurs. Bien que le fait de créer des animations à partir de zéro soit encore très répandu, cela exige des compétences artistiques ainsi que beaucoup d'heures de travail (si ce n'est des jours) pour créer quelques secondes d'animation. Pour toutes ces raisons, la capture de mouvements est devenue très intéressante pour générer des animations : au lieu de les *créer*, l'idée est tout simplement de *reproduire* les mouvements d'un acteur. Cependant, la capture de mouvements n'est pas parfaite et nécessite plusieurs améliorations. En effet, la capture de mouvements est un processus qui implique des techniques et des équipements complexes. Cela se traduit souvent par des animations bruitées qui doivent être éditées. De plus, il est très difficile d'exactement prévoir ce que doivent être les animations finales. Par exemple, il arrive souvent que le réalisateur d'un film décide de changer le script. Dans ce cas, les animateurs doivent modifier certaines partie d'un mouvement voire sa totalité. **Le but de cette thèse est de fournir aux animateurs des outils interactifs pour les aider à facilement et rapidement modifier des animations préexistantes**. Dans un premier temps, nous présentons notre moteur de cinématique inverse utilisé pour appliquer des contraintes cinématiques à chaque temps d'une animation. Nous proposons ensuite un système de déformation de mouvements permettant à l'utilisateur de spécifier des contraintes et d'éditer une animation initiale afin qu'elle puisse être exploitées dans un nouveau contexte (personnages, décors, etc). Finalement, nous présentons un algorithme semi-automatique pour extraire les caractéristiques importantes d'une animation obtenue par capture de mouvements. Ces caractéristiques peuvent par exemple aider les animateurs à spécifier les contraintes importantes qu'une animation devra respecter.

Abstract

Convincingly animating virtual humans has become of great interest in many fields since recent years. In computer games for example, virtual humans often are the main characters. Failing to realistically animate them may wreck all previous efforts made to provide the player with an immersion feeling. At the same time, computer generated movies have become very popular and thus have increased the demand for animation realism. Indeed, virtual humans are now the new stars in movies like Final Fantasy or Shrek, or are even used for special effects in movies like Matrix. In this context, the virtual humans animations not only need to be realistic as for computer games, but really need to be expressive as for real actors. While creating animations from scratch is still widespread, it demands artistics skills and hours if not days to produce few seconds of animation. For these reasons, there has been a growing interest for motion capture: instead of creating a motion, the idea is to *reproduce* the movements of a live performer. However, motion capture is not perfect and still needs improvements. Indeed, the motion capture process involves complex techniques and equipments. This often results in noisy animations which must be edited. Moreover, it is hard to exactly foresee the final motion. For example, it often happens that the director of a movie decides to change the script. The animators then have to change part or the whole animation. **The aim of this thesis is then to provide animators with interactive tools helping them to easily and rapidly modify preexisting animations**. We first present our Inverse Kinematics solver used to enforce kinematic constraints at each time of an animation. Afterward, we propose a motion deformation framework offering the user a way to specify prioritized constraints and to edit an initial animation so that it may be used in a new context (characters, environment,etc). Finally, we introduce a semi-automatic algorithm to extract important motion features from motion capture animation which may serve as a first guess for the animators when specifying important characteristics an initial animation should respect.

Keywords: Inverse Kinematics, Motion Editing, Motion Capture, Constraint Detection

CHAPTER 1

Introduction

In this thesis, we propose interactive tools capable of helping animators to easily and rapidly edit motion capture animations in order to produce high-end animations.

1.1 Context

Convincingly animating virtual humans has become of great interest in many fields in recent years. In computer games for example, virtual humans are often the main characters. Failing to realistically animate them may annihilate all previous efforts made to provide the player with an immersion feeling. In parallel, computer generated movies have become very popular and thus have increased the demand for animation realism. Indeed, virtual humans are now the new stars in movies such as Final Fantasy or Shrek, or are even used for special effects in movies such as Matrix. In this context, not only do virtual human animations need to be realistic as for computer games, but also really need to be expressive as for real actors.

This task is inherently difficult for two reasons. First of all, humans are naturally experts when it comes to looking at other humans. Hence, any subtle inaccuracy in a virtual human animation leads to a strange feeling for human observers: we do not precisely know what is wrong, but we know that something is wrong. Secondly, the human body is a complex architecture containing a lot of DoFs. As it is not acceptable to change each articulation parameters by hand, complex algorithms must be developed to ease the animation of such structures.

The manual design of key postures, also known as *keyframing*, is still a widespread technique as it offers the animators great control over the results. Another technique is to procedurally generate human animations using predefined mathematical functions which we can tune using specific, low level parameters. Other algorithms use dynamics to produce human animations. However, these methods tend to produce motions which look mechanical or even passive. Indeed, human motion is not entirely dictated by mathematics or physical laws.

For these reasons, there has been a growing interest for motion capture: instead of creating a motion from scratch, the idea is to *reproduce* a motion. The motion of a live performer is recorded using mechanical armatures, magnetic sensors or cameras before being applied to virtual humans. This technique is widely used it allows to reproduce subtle details of human movements in a very short period of time.

However, motion capture is not perfect and still needs improvements. Indeed, the motion capture process involves complex techniques and equipments. This often results in noisy animations which must be edited. Moreover, it is hard to exactly foresee the final motion. For example, it often happens that the director of a movie decides to change the script. The animators then have to change part of or the whole animation.

One method consists in adjusting each posture of the animation by hand to correct unwanted artifacts. While it is still used in many computer-generated imagery (CGI) animation companies, this demands great efforts and artistic skills. For this reason, it is of interest to use high-level constraints which the animators may use to adjust a whole animation. For example, instead of specifying each point of the trajectory of a hand, an animator may define a high-level constraint specifying important locations this hand should pass through and its whole trajectory may then be automatically generated.

Afterward, given a set of high-level constraints specified by the user, we should be able to realistically deform an animation to satisfy these constraints. It may happen that these latter are not simultaneously achievable because they are conflicting. The animator must then decide which of them are very important and which may be only partly achieved by specifying priorities.

The first step of the editing process is to define important characteristics of the initial motion the animator needs to keep. In many animations however, these characteristics are numerous: specifying all of them by hand may be tedious and take several hours. It is then of interest to provide the animators with a first guess of these important constraints.

In this thesis, we propose a motion deformation framework offering the user a way to specify prioritized constraints to deform an initial motion capture animation. These prioritized constraints are useful as they directly reflect the user needs when conflicts between constraints occur. Moreover, we provide the user with a semi-automatic algorithm to detect important constraints in motion capture animations.

1.2 Overview of the Motion Deformation Algorithm

Our motion deformation algorithm belongs to the class of per-frame IK plus filtering methods. It is summarized in Figure 1.1. The main idea is that a motion may be decomposed into

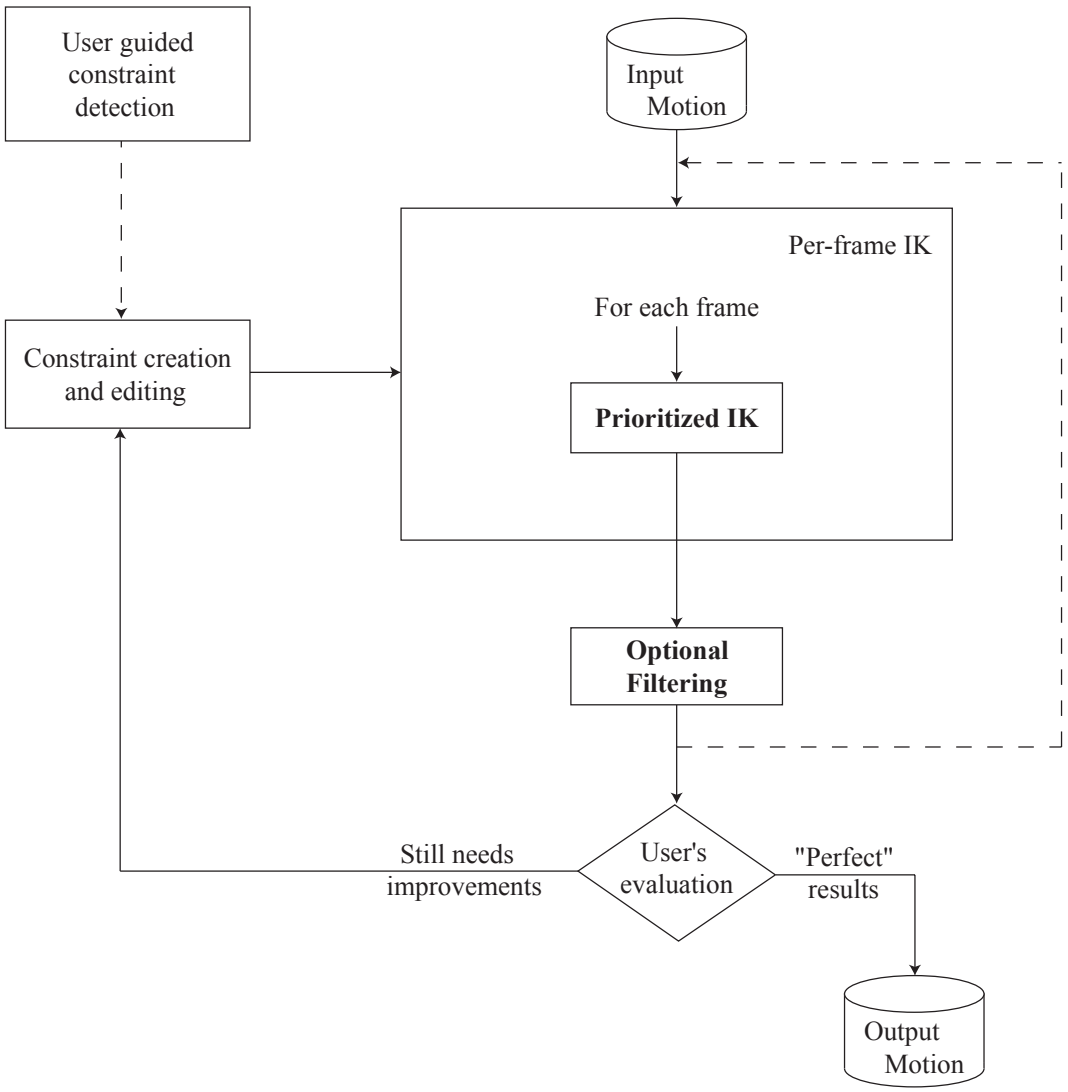

Figure 1.1: Overview of the motion deformation framework.

independent frames. Then, given a set of user-specified high-level constraints, each frame is adjusted so that it achieves the predefined constraints as much as possible. Finally, the resulting motion is filtered to remove potentially added discontinuities. This latter stage of the algorithm may destroy the results obtained from the previous one. As a consequence, a per-frame IK plus filtering algorithm is inherently iterative.

We additionally propose a semi-automatic algorithm to extract important motion features from a motion capture animation. This algorithm being particular enough in the context of motion editing, we develop it in Chapter 6.

1.3 Organization of this Document

This document is organized as follows:

- In Chapter 2, we review related work on human computer animation. We mainly focus our discussion on motion editing techniques.
- In Chapter 3, we present our numerical IK solver used to enforce user-specified constraints during the motion deformation process.
- In Chapter 4, we propose high-level constraints which the user can manipulate to edit an animation.
- In Chapter 5, we present our motion deformation algorithm. It is in charge of deforming an initial animation in order for the final one to respect a set of user-specified constraints.
- In Chapter 6, we propose a semi-automatic algorithm to detect geometric constraints in motion capture animations. This method is particularly useful in helping the animator specify the set of original constraints.
- In Chapter 7, we conclude this dissertation. In particular, we summarize the contributions of this thesis and suggest future work.

1.4 Preliminaries

1.4.1 Concepts and Definitions

In this document, we often refer to specific notions that may vary depending on the context. For example, the notion of goal is not the same whether we are considering postures or motions. We hereafter choose the following terminology for the remainder of this document:

Kinematic Chain: we call a *kinematic chain* any chain of links (or joints) with an *end-effector* at its free-end. The topmost joint is called the *root* of the kinematic chain.

Articulated Figure: we call an *articulated figure* any skeletal structure modeled as a hierarchy of joints. This representation is especially useful in an animation context. We use the HAnim standard to represent the articulated figures [HAnim]. Technical details about this representation and all associated issues (joint types, joint limits, etc) are explained in detail in [Baerlocher, 2001; Aubel, 2002]. It is important to note that an articulated figure may contains numerous kinematic chains. For example, a kinematic chain may be defined from the shoulder to the wrist, from the hip to the toe, etc.

Task: we call a *task* any specific goal to achieve in an *Inverse Kinematics* context. Hence, we are only interested in the final *posture* of the articulated figure which achieves (or not) the specified goal.

Constraint: we call a *constraint* any specific goal to achieve in a *motion editing* context. In this case, we do not only consider the final *posture* of the articulated figure achieving (or not) the specified goal but are also interested in the motion to achieve it (or not).

Motion Representation: we define a motion as a continuous function of time $\boldsymbol{m}(t) = (\boldsymbol{p_r}(t), \boldsymbol{q_1}(t), \cdots, \boldsymbol{q_{n_{joints}}}(t))$ where $\boldsymbol{p}(t)$ and $\boldsymbol{q}_0(t)$ represent the global position and orientation of the root node and $\boldsymbol{q}_i(t)$ is the local transformation of the i^{th} joint.

Posture Representation: Considering the previous notation, the posture of a virtual character at a given time t_α is defined as $\boldsymbol{m}(t_\alpha) = (\boldsymbol{p_r}(t_\alpha), \boldsymbol{q_1}(t_\alpha), \cdots, \boldsymbol{q_{n_{joints}}}(t_\alpha))$. However, this notation implicitly considers that this posture is part of an animation $\boldsymbol{m}(t)$. Hence we use this definition only when we are dealing with *motion editing* issues as it is only relevant in that context. We use the notation $\boldsymbol{\theta}$ instead when referring to the state vector of an articulated figure when dealing with *Inverse Kinematics* issues.

Abbreviations: In this thesis, we use the following abbreviations:

- **IK** for Inverse Kinematics,
- **DoF(s)** for Degree(s) of Freedom,
- **CoM** for Center of Mass.

1.4.2 Mathematical Notation and Conventions

In this document, we use the *column vector* convention and *right handed* coordinate frames. Scalars are denoted by small letters such as s. Vectors are denoted by small boldface letters such as $\boldsymbol{v}$. Matrices are denoted by capital boldface letters such as $\boldsymbol{M}$. Additional notations are introduced when necessary.

CHAPTER 2

Related Work

In this chapter, we present previous work on character animation. In the first section, we focus our attention on motion generation methods that is those which create new animations *from scratch*. We then review motion editing methods: those are used to modify *a single preexisting animation* to produce new ones. Finally, we present techniques using *motion databases*. It is important to note that some methods may belong to several categories at the same time. In these cases, we only consider their most important contribution to decide how they should be classified.

2.1 Motion Generation Methods

In this section, we consider a motion generation method any algorithm able to create an animation without any motion as input.

2.1.1 Keyframing

Keyframing borrows its name from the traditional hand animation technique [Lasseter, 1987]. It is composed of two steps. First, the animator specifies key postures for the character being animated: *the keyframes*. These keyframes are placed at specific times in the animation. Then, an algorithm computes inbetween frames by interpolating between these keyframes. While defining keyframes may be directly done using direct kinematics for simple objects, it becomes rapidly tedious and time consuming as soon as the total number of the animated

Figure 2.1: Wally B.'s zip off shows use of squash and stretch, anticipation, follow through, overlapping action, and secondary action (source: [Lasseter, 1987]).

object's DoFs increases. This is particularly noticeable for character animation as typical virtual humans may contain up to fifty DoFs without considering the fingers.

For that reason, specific algorithms have been developed to ease keyframes definition. Instead of defining each joint value of the virtual human, the animator only needs to specify the desired location of a body part. Then, each joint value directly or indirectly controlling this body part location is automatically computed. This technique is known as *IK*. As we dedicate an entire chapter to IK, a review of the most significant works in this field may be found in Chapter 3.

The interpolation algorithm is a crucial factor in the appearance of the final animation. Inbetween frames may be generated using *linear interpolation*. However, this technique is rarely employed as it usually generates unnatural motions. A more convenient method is to use cubic splines as they produce smooth and continuous curves.

The keyframing technique is still widespread as it offers the animators a total control over the final animation. Nevertheless, it requires that they deeply understand the underlying algorithms to be able to foresee the results. In addition, it demands a lot of experience and artistic skills to produce expressive animations. Finally, the animator must generally define a set of keyframes which is quite dense in order to obtain realistic motions.

2.1.2 Procedural Methods

Figure 2.2: Walking crowd with different gaits (source: [Boulic et al., 2004]).

Procedural methods rely on handcrafted algorithms to automatically generate motions. These algorithms may be thought of as time varying functions controlling low-level motion parameters. Perlin [Perlin, 1995] applied procedural texture synthesis techniques to create real-time responsive characters. The author managed to generate a wide variety of motions. Additionally, Perlin and Goldberg [Perlin and Goldberg, 1996] proposed the *Improv* system to create lifelike animated characters. Chin et al. [Chi et al., 2000] applied Laban Motion Analysis to retrieve the qualitative aspects of movements. In particular, they focused on the shape and effort properties. These parameters was then adjusted in real-time to alter

the "meaning and the dramatization enacted by the synthetic actor". Finally, Boulic et al. [Boulic et al., 2004] proposed a walking engine controlled by high-level parameters such as style, speed or target location. It is important to note that a virtual human is considered as an active system: it can move and react by itself without any external force. This means that procedural methods often use some kind of behavioral systems to enhance the believability of the generated character animations.

One *procedure* is in general dedicated to one class of animations. Hence, whenever the user needs some new motions, he has to redesign new algorithms. However, procedural methods are in general difficult to design. Indeed, it requires an in-depth understanding of the underlying parameters to effectively generate the expected motions. Finally, procedural methods rarely generate animations as realistic as motion capture ones because they fail to reproduce subtle human motion details.

2.1.3 Physics-Based Methods

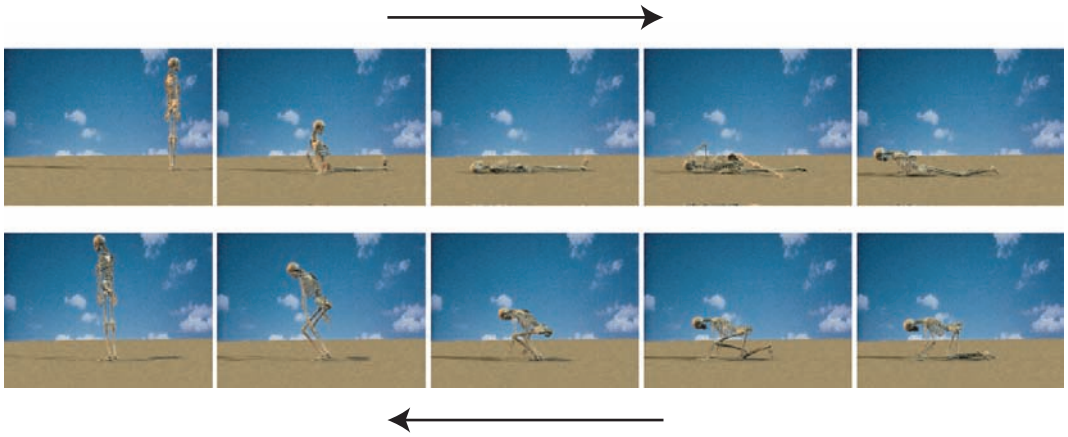

Figure 2.3: A dynamic *virtual stuntman* falls to the ground, rolls over, and rises to an erect position, balancing in gravity (source: [Faloutsos et al., 2001a]).

Physics-based methods could be also thought as procedural methods. However, instead of relying on hand-designed functions only, they also consider physical laws in order to compute new motions. The character is first considered as a structure of body parts for which we precisely know the mass distribution. Average mass distribution may be easily found in the biomechanical literature [Winter, 2004]. The problem is then to find the joint torques which produce the expected motion. Hodgins et al. [Hodgins et al., 1995] presented joint controllers producing a wide range of motions such as stand, run at various speeds, ride a bicycle and perform a gymnastic vault. They used finite state machines to control the different phases of the motion and proportional-derivative control laws for low-level joints control. This technique has been further extended by Wooten and Hodgins in [Wooten and Hodgins, 2000] to generate transitions between leaping, tumbling, landing, and balancing motions. However, a joint controller is specific to one class of motion and it is often difficult to compose them to obtain more complex behavior. To this end, Faloutsos et al. [Faloutsos et al., 2001a] [Faloutsos et al., 2001b] presented a framework allowing users to compose simple

joint controllers to create more complex ones. Individual controllers only need to determine if it is able to take control of the character dynamics depending on its current state: this is the *pre-conditions*. Given these information, a supervisor controller is build to resolve more complex tasks based on a combination of the simple ones provided by the individual controllers. Furthermore, they proposed a learning algorithm to semi-automatically teach the controllers appropriate pre-conditions. They demonstrated the validity of their framework using a virtual stuntman able to preserve balance and to recover from a fall. Ko and Badler [Ko and Badler, 1996] presented a two-step process for generating dynamically sound walking motion. A kinematics locomotion generator first produces walking motions which are dynamically modified to maintain balance and to respect stress constraints. Laszlo et al. [Laszlo et al., 1996] applied limit cycle control to cyclic motions such as walking to ensure that the underlying joint controllers still provide stable motions after small disturbances. While these methods can generate a wide variety of motions, these latter are limited to the virtual character they have been made for. This limitation severely reduces the potential reusability of joint controllers. Hodgins and Pollard [Hodgins and Pollard, 1997] then proposed a method to automatically adapt existing joint controllers to new characters. Their algorithm is applied in two stages. The joint controllers parameters are first approximately scaled with respect to the proportions of the new and the original character (sizes, masses and moments of inertia). Afterward, the new parameters are fine-tuned with simulated annealing. In particular, they demonstrated the efficiency of their method on a running child based on the joint controllers designed for an adult. Neff and Fiume [Neff and Fiume, 2002] proposed to enhanced the natural looking of physically generated motions by modeling the tension and the relaxation of the character. They demonstrated that these parameters could be introduced at joint level by taking antagonist muscles influence into account. Furthermore, they introduced in [Neff and Fiume, 2005] a prototype system allowing the animator to first explore the same space of solutions using high-level interfaces such as scripting and then to refine the animation using low-level parameters. While these methods are able to generate motions that are physically correct, they often produce mechanical animations. Moreover, it requires a deep understanding of the physical parameters to be able to foresee the output motions. Finally, the computational times are generally too prohibitive to use such techniques for interactive applications.

2.1.4 Motion Capture

Motion capture employs additional hardware to produce animation [Menache, 1999]. There exists several different types of motion capture systems:

- **Optical:** the system is composed of a computer controlling a set of infrared cameras capturing the three-dimensional position of reflective markers. The entire motion of the performer is then reconstructed based on the trajectory of these markers. Chai and hodgins [Chai and Hodgin, 2005] introduced a technique employing video cameras and a small set of retro-reflective markers to create a low-cost, easy-to-use optical motion capture system.

- **Electromagnetic:** the system is composed of electromagnetic sensors. These latter

Figure 2.4: Users wearing a few retro-reflective markers control the full-body motion of avatars by acting out the motion in front of two synchronized cameras. **From left to right:** walking, running, hopping, jumping, boxing, and Kendo (source: [Chai and Hodgin, 2005]).

measure their relative position and orientation to a magnetic reference. The motion of the performer is then immediately known that is why these systems are often used for real-time motion capture.

- **Electromechanical:** the system is composed of a suit similar to exoskeletons but equipped with potentiometers measuring the orientation of the performer articulations.

- **Video-based:** this class of techniques is becoming a very active field of research. The basic idea is to extract the motion of a performer given a video sequence. The main difficulty comes from the fact that often only one camera is used. Moreover, no additional hardware is used other than a simple camera.

Motion capture is now an intensively used technique to produce character animation. Indeed, it is able to capture very subtle details of human motions that makes the final results often more realistic than other methods. However, motion capture needs extensive post-processing to obtain usable animations. A motion capture animation is never perfect and always needs additional adaptation to take into account varying parameters:

1. The proportions of the virtual humans are often different than those of the performer,

2. The environment is different: the animation needs additional adjustments to ensure that the virtual character is accurately interacting with the surrounding environment,

3. The final animations are often noisy. The noise may come directly from the raw motion capture data or may be added during the post-processing steps.

In this thesis, we propose in particular an interactive framework to edit such animations.

2.2 Motion Editing Methods

In this section, we review methods used for editing animations. We consider an algorithm as being a motion editing method as soon as one if its main goal is to preserve the important characteristics of the initial motion.

2.2.1 Signal Processing

Figure 2.5: Emotion-based running examples with step-constraints (source: [Unuma et al., 1995]).

Several previous works considered each animation curve independently as a time-varying signal. They then applied signal processing techniques to modify an input motion. In particular, Bruderlin and Williams [Bruderlin and Williams, 1995] adapted multiresolution filtering, multitarget interpolation, waveshaping and displacement mapping to character animation. Simultaneously, Witkin and popović [Witkin and Popovic, 1995] described a technique called *motion warping* combining time warping and displacement mapping. The user conveniently places keyframes for editing an input animation which serves as a set of spatial constraints. Afterward, the displacement maps are computed for each animation curve as these latter are warped independently. Finally, the interpolation between keyframes is based on the changes (displacement maps) instead of being based on the absolute motion curves values. In [Unuma et al., 1995], Unuma et al. described a simple method to represent periodic motions using the so-called rescaled Fourier functional model. Then, the motions was easily interpolated, extrapolated, or subtracted with other motions.

These methods are in general difficult to use because the manipulation of a time-varying signal is fastidious and non-intuitive when the animator desires to modify a motion with some precise requests. Furthermore, they do not ensure that original kinematic constraints are preserved. When using motion warping for example, if the keyframes are not correctly placed, then the final motion may violate important geometric constraints such as the feet penetrating the ground or sliding.

Amaya et al. [Amaya et al., 1996] presented a technique extracting the *emotional component* of a motion and applying it to other motions to add emotions. Their method is divided into three main steps. First, the same motion is captured with different emotions: neutral, sad, angry, etc. Then, the emotional component is extracted by subtracting the neutral mo-

tion to the emotional one. Finally, this component is applied to another neutral motion to add similar emotional content.

2.2.2 Physics-Based Methods

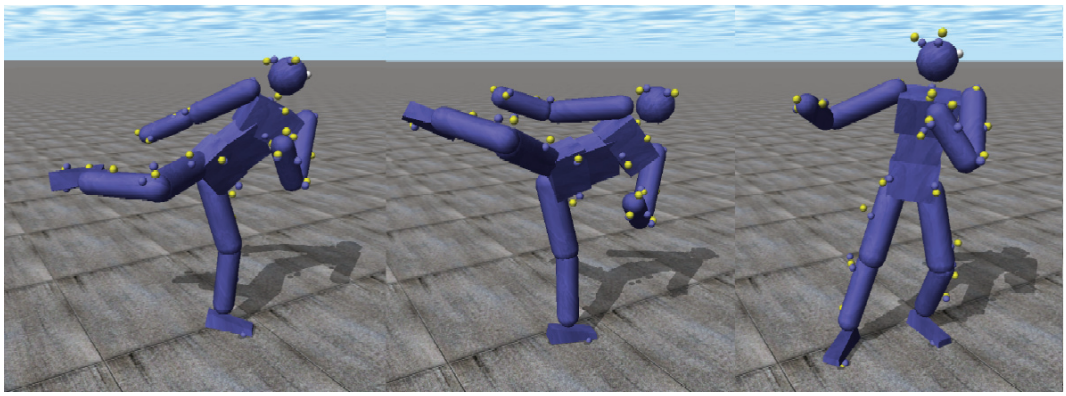

Figure 2.6: Close up stills showing marker placement (lighter spheres show motion capture, darker are virtual markers) (source: [Zordan and Horst, 2003]).

While it has been shown that a motion does not necessarily have to be physically correct to be visually appealing [Reitsma and Pollard, 2003; O'Sullivan et al., 2003], physical laws have been widely used to improve animations' realism. Some methods constrained the zero moment point of the character to remain inside its support polygon [Tak et al., 2002][Dasgupta and Nakamura, 1999][Ko and Badler, 1996] which is a requirement for physical correctness. In [Shin et al., 2003], the authors estimated the character's mass distribution using reference motions known to be physically correct. Then, they *touched up* the animation using different physical laws. During flight stages, they ensured that its total angular momentum was conserved. During ground stages, they used a simplified formulation of the Zero Moment Point to ensure that the character was dynamically balanced. [Pollard and Reitsma, 2001] and [Yamane and Nakamura, 2003] used a dynamics filter to track a reference motion while enforcing dynamic constraints. In [Zordan and Hodgins, 2002], the authors controlled a physical simulation using motion capture data. The virtual human was then able to dynamically react to external forces such as a punch. Zordan and Van Der Horst [Zordan and Horst, 2003] proposed a novel approach to convert 3D markers position to joint angles using dynamics. The method uses an inverse dynamics approach (instead of IK) to compute each frame. Each marker is controlled by a spring force attracting it to a correct location. Using internal forces, some consistency is added to avoid joint limits violation. Moreover, they enforced footplants by using friction external forces. Zordan et al. [Zordan et al., 2005] presented a technique mixing physical simulation and motion capture animations to simulate unexpected impacts onto virtual characters. When an impact occurs, the physical simulation drives the animation while a specialized search routine determines the best plausible re-entry into the motion database following the impact. Neff and Fiume [Neff and Fiume, 2003] introduced the concept of *aesthetic edits*. Edits are procedural operations which directly influence the animation expressiveness. They in particular presented three edits: the

succession (how the motion spread to the body), the amplitude (similar to a scale operator) and the extent (the spatial range of arms motion).

2.2.3 Spacetime Constraints

Figure 2.7: Top: Simple input animation depicting hopscotch (a popular child game consisting of hops, broad jumps and a spin jump). **Bottom:** Synthesized realistic hopscotch animation (source: [Liu and Popovic, 2002]).

The spacetime constraints method may be used to synthesis new motions or to edit existing ones using either kinematics or dynamics constraints. This resolution method is sufficiently different from previous ones that we prefer to dedicate a whole section to this technique. Furthermore, the spacetime constraints formulation originated numerous works in character animation.

Spacetime constraints refers to techniques computing a whole motion at once instead of computing each frame independently. As it considers a motion as a single entity, it is then possible to define constraints for the entire duration of the animation. Witkin and Kass [Witkin and Kass, 1988] first introduced this formulation to the graphics community. They considered the problem of synthesizing and/or editing an animation as a constrained optimization problem. The animators specifies *how* the final character should perform the motion, for example by minimizing the energy expenditure. This defines the objective function to optimize. Additionally, the users defines *what* the character should do. For example, a starting and ending postures for the character. This specifies the constraints for the constrained optimization problem. The authors used this method in particular to generate a hurdle jump for Luxo. They also included constraints enforcing the laws of physics and created an objective function minimizing the amount of energy the character must expend with its muscles. As this method is governed by the laws of physics, the results are physically correct. To reduce computational cost, Cohen [Cohen, 1992] used spacetime windows to focus on specific parts of the current animation and to provide the animators with interactive feedbacks. Gleicher and Litwinowicz [Gleicher, 1997] [Gleicher and Litwinowicz, 1998] proposed to apply the spacetime constraints technique to edit preexisting animations. As they started with an example motions, the problem was simpler because they did not have to specify motion details but instead had to ensure that the final solution stayed as close as possible to the initial one. Moreover, Gleicher [Gleicher, 1998] applied the spacetime constraints technique to perform motion retargetting. The user defines a set of constraints that the retargetted motion should preserve. Based on this set of constraints, the spacetime solver tries to find a motion that satisfies the previous set of constraints while minimizing

an objective function. In his algorithm, Gleicher used an objective function minimizing the distance to the initial posture (in a joint value sense). In [Popovic and Witkin, 1999], Newton's laws were applied on a simplified character to minimize computational costs. Rose et al. [Rose et al., 1996] minimized energy consumption to obtain realistic transitions between motions. Liu and Popović [Liu and Popovic, 2002] introduced a method for rapid prototyping of realistic motions. Starting from a simple animation generated using keyframing, physical laws are enforced to produce a more realistic one, with physics. Given an input animation, they first detect position and sliding constraints in order to separate the animation into constrained and unconstrained (flight) stages. Afterward, they generate transitions between these stages by suggesting the user a set of previously learned transition poses. Finally, they compute the final animation by minimizing the mass displacement, the velocity of the DoFs and by ensuring static balance. This optimization is subject to constraints on the linear and angular momentum that directly depend on whether the character is on the ground or airborne. Generating physically realistic animations using optimization often requires to compute first derivative of joint torques which is of quadratic complexity. This inevitably leads to scale problems. Indeed, when the number of DoFs increases, the computation times become rapidly prohibitive. Fang and Pollard [Fang and Pollard, 2003] then demonstrated that Newton-Euler equations of motion are rewritable to allow first derivatives of aggregate forces and torques to be computed in linear time. Abe et al [Abe et al., 2004] used a framework similar to the one presented in [Liu and Popovic, 2002] to generate a variety of motions given an input one. They then generated a variety of motions in real-time by using simple interpolation. Finally, Safonova et al. [Safonova et al., 2004] introduced a technique to generate motions using an optimization in a low-dimensional space. First, a set of reference motions is defined by the animator to build the low-dimensional space using Principal Component Analysis. The constraints are expressed in the world frame and then projected onto the low-dimensional space previously built. Finally, the motion is generated by optimizing its "representation" in the low-dimensional space (i.e. the PCA coefficient over time). To enforce kinematics constraints, they also used a simple IK solver. The optimization uses an objective function which tries to minimize the torques, the jerkiness and the deviation to original motions. The final result are represented using B-Splines representing the values of the PCA coefficients over time.

The main drawbacks are directly related to this formulation which uses a constrained optimization problem. Indeed, the constrains and the objective functions have to be mathematically defined. This is particularly difficult when the animator is mostly interested in the *style* of the final animation. For example, it is difficult (if not impossible) to mathematically define a "walk sadly" constraint. Moreover, this approach requires solving a single mathematical problem for the entire motion. This leads to very large constrained optimization problems that are usually very difficult to solve.

2.2.4 Per-Frame IK Plus Filtering

Our framework belongs to this class of motion editing techniques. We already gave its overview in Figure 1.1. We give a more in-depth state of the art on these class of motion editing techniques in Chapter 5 for the interested reader. As far as our knowledge goes, the

first work applying IK on a per-frame basis was introduced by Boulic in using the *Coach-trainee* metaphor. In [Lee and Shin, 1999], Lee and Shin introduced the *Per-Frame Plus Filtering* class of motion editing techniques. Choi and Ko [Choi and Ko, 2000] used an Inverse Kinematic solver to enforce constraints in an online manner. A similar technique was described in [Shin et al., 2001] where a concept of importance was introduced to choose, at each frame, whether a constraint is relevant or not. Monzani et al. [Monzani et al., 2000] proposed to use an intermediate skeleton to solve the problem of motion retargetting. Kovar et al. [Kovar et al., 2002b] used a specialized IK solver to solve the specific problem of foot-sliding. Kulpa et al. [Kulpa et al., 2005] used a Cyclic Coordinate Descent to enforce spatial constraints. Moreover, they proposed a method so that the motion adaptation is morphology-independent. Finally, Gleicher [Gleicher, 2001] proposed a taxonomy of constraint-based techniques. Our techniques provides several improvements over existing methods. In particular, we propose to use prioritized constraints to solve conflicts. We also propose a simple control of the trajectory of the CoM so that we can add significant deformation to an initial animation and still ensure that the final character is balanced. Finally, we also provide new classes of constraints to help the animators specify their needs.

2.3 Motion Database-Based Methods

Even though techniques relying on motion databases to produce new ones are beyond the scope of this thesis, we give an overview of the most significant works in this field. We first present generic methods to retrieve data from a database. In particular, we present methods focusing on motion retrieval. We present afterward methods combining motions such as blending and motion graphs techniques.

2.3.1 Data Retrieval

The vast majority of the following techniques are based on the same idea:

1. perform a dimension-reduction transform to project the sequence-space S onto the feature-space F,
2. define a correct similarity metrics in F,
3. find good potential matches in F (without discarding good matches),
4. post-process the results in the S to keep only good matches.

Agrawal et al. [Agrawal et al., 1993] proposed a method to search for whole sequences of data in a database. Given N sequences of length n, they first apply n-point Discrete Fourier Transform (DFT) and keep the first f coefficients only. Thus, a sequence is projected onto a f-dimensional point. Finally, all these f-points are organized for fast searching (using $R^* - Trees$ [Beckmann et al., 1990], $R - Trees$ [Jagadish, 1990] [Sellis et al., 1987] [Guttman, 1984], linear quadtrees [Orenstein, 1986] or grid-files [Nievergelt et al., 1984]). Given a query Q with a tolerance ϵ, they then apply the n-point DFT. The f-dimensional points that

Figure 2.8: Retrieval result of walk-forward (source: [Liu et al., 2003]).

are at a distance less than ϵ are then retrieved. To ensure that all potential positive matches are found, the distance in feature-space must underestimate (or even match) the distance in the sequence-space. Faloutsos et al. [Faloutsos et al., 1994] extended this method to *subsequences matching*. They compute the corresponding trail of each sequence in feature-space by performing the DFT on each subsequence using a sliding window of length w. Storing all the resulting trails in a $R-tree$ is not efficient. So, they first divide trails into sub-trails. Then, each Minimum Bounding Rectangle (MBR) is computed and stored. Finally, they use a hierarchical MBR representation. To search a subsequence, they project it into the feature-space and retrieve all the subsequences whose MBR contains the previous projected point. To search queries longer than w, they divide the query into p sub-queries of length w. Then, each sub-queries is performed with a tolerance of $\frac{\epsilon}{\sqrt{p}}$. Finally, the results are merged to remove false matches. Chan and Pu [Chan and Fu, 1999] proposed a similar method using wavelet for features extraction. Moreover, they introduce the *v-shift similarity*: the average of the sequences is subtracted before computing the Euclidean distance. Rafiei and Mendelzon [Rafiei and Mendelzon, 1997] proposed a set of linear transformations applied on the index allowing a wider range of possible queries. Keogh et al. introduced in [Keogh et al., 2001] a new simple technique for dimension reduction called Adaptive Piecewise Constant Approximation (APCA). Each time serie is compressed by:

- Approximating it with a set of segments (of varying length),
- Replacing each segment by its mean.

Finally, the time-serie is associated to the set of the mean values. The authors also introduced 2 new metrics:

- An approximate Euclidean measure that is very close to the corresponding Euclidean metrics but may overestimate the errors in feature-space (compared to the metrics in the original space) leading to false dismissals.
- An exact Euclidean measure that does underestimate the errors but is less accurate.

In [Vlachos et al., 2003], Vlachos et al. used a method which can accommodate two different metrics: The Longest Common SubSequence (LCSS) and the Dynamic Time Warping (DTW).

One restriction is that these methods directly rely on numeric metrics. Doing so completely discard semantics information. For example, two sequences could be quite different in shape but represent the same logical event.

A common strategy to easily retrieve motions from a database is to first annotate these motions and then, depending on these annotations, retrieve the corresponding animations. Moreover, special editing operations are often required such as cropping the frames of interest. However, this technique is limited to only short animations (clips). Indeed, the longer an animation is, the less intuitive the annotation process becomes. Moreover, if an animation is too long, the final annotations tend to become useless. Liu et al. [Liu et al., 2003] proposed a method to retrieve motions from example queries. First, the hierarchy is divided considering 5 sets of joints from parent to children joints (root, LHip, RHip, Chest, etc). Then, all the motions of each set of joints (one after the other from the parents to the children) are classified using dynamic clustering. Similar motions are associated to the node and a sample set representing the clusters is extracted and put in the node. This process is repeated until level 5 is reached. To retrieve a motion, each set of joints of the query (from top to bottom) is considered independently. They choose the k-nearest motions (contained in few nodes) and repeat the process until a leaf is reached. Instead of computing similarity between two entire motions, they extract keyframes representing the motions and compute the distance based on these keyframes two extract the matches. Emering et al. [Emering et al., 1998] used a similar approach to hierarchically recognize live motion captured animations. A motion is divided into 5 layers:

1. CoM velocity,
2. end-effectors velocities,
3. CoM position,
4. end-effectors positions,
5. whole body posture (joints values).

Then the motion retrieval is performed starting with the first layer. The potential candidates are then checked using the second one and so on. This approach builds a coarse-to-fine motion retrieval (starting with a low-cost similarity metrics with low discrimination and ending with a high-cost similarity metrics with a high discrimination). This technique achieves real-time rates which is its main contribution. However, it is difficult to prove that it does not end with false dismissals, which is an important property of common data retrieval algorithms.

Kovar and Gleicher [Kovar and Gleicher, 2004] used motion clips as requests. First, a match web is constructed for each pair of motions in the database. Then, given a query (which already belongs to the database), all the segments that satisfy a specific distance criteria are retrieved. These results are then used as intermediate queries to initiate new searches. Finally, the retrieval stops when no new clips are added. This method is quite powerful and gives good results. However, using a motion clip as a request is not suited to retrieve a motion from scratch as the user needs to first browse the database to find an initial guess. Thus, instead of using motion clips and corresponding annotations to query a motions database, Sakamoto et al. [Sakamoto et al., 2004] proposed a method based on key-postures to retrieve adequate clips from a data set. The whole database is first projected onto a two-dimensional space using Self-Organizing Map (SOM) while keeping topological relations of metrics in the higher space. Key-nodes are then kept by first clustering the nodes in the Motion Map and for each resulting region, choosing the closest node to its center to represent the whole cluster. Finally, clips are retrieved based on key-nodes chosen by the user on the Motion Map. Jenkins and Matarić [Jenkins and Mataric, 2002] used a motion capture database, perform dimension reduction using spatio-temporal isomap. Afterward, the result is clustered to extract behaviors. Finally, this process is repeated to extract behaviors of higher level. Cardle et al. [Cardle et al., 2003] applied the common data retrieval method to motions. In [Keogh et al., 2004], Keogh et al. proposed a method using uniform scaling for their similarity criterion instead of using the Euclidean distance or dynamic time warping. To evaluate the distance between a query Q and a candidate C, all the possibilities are computed (i.e all the possible scaling factors). To prune the search, they introduced a lower-bound distance using Maximum Bounding Envelope. Finally, they proposed a simple but effective way to index the data (high-dimensional as it is motions) to speedup the search. These methods do not take the environment into account. In particular, while retrieving motions interacting with objects in the scene, the post-processing (motion editing) due to the enforcement of interaction constraints tends to annihilate all the benefits of these methods. For example, if one wants to retrieve all the "sit down" motions to apply them onto a chair, the time spent to first define the constraints, and then to enforce them so that the character effectively sits down correctly would be prohibitive. Hence, these methods still require motion editing to ensure that the retrieved motion will correctly interact with the surrounding environment.

2.3.2 Motion Combination

A common method to produce new motions given example ones is to blend them according to weights. However, directly interpolating between *weighted motions* is not practical. Indeed, manually determining the weights for each motion is not acceptable for animators. Hence, motion blending techniques often provide high-level parameters to control this interpolation. Rose et al. pioneered this field by introducing in [Rose et al., 1998] a method to parametrized the motions contained in a database using *verbs* and *adverbs*. The verbs represent the type of motion and the adverbs the interpolation parameters (the weights). The main problem is then to construct a continuous application that, given an adverb, returns the interpolation parameters. This is commonly called *multivariate interpolation*. The authors solved this problem first using a low order linear polynomial to roughly approximate the

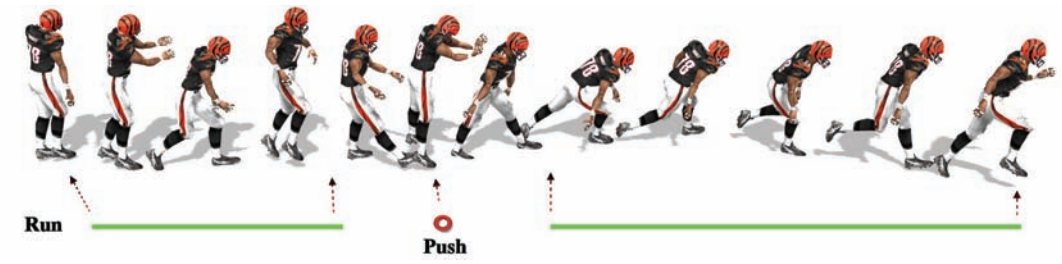

Figure 2.9: In addition to matching the annotations, a specific frame or motion can be forced to be used at a specific time. Here, the person is forced to pass through a pushing frame in the middle of the motion while running before and after the pushing (source: [Arikan et al., 2003]).

adverbs space. Then, Radial Basis Functions were used to locally adjust the polynomial. They additionally constructed a *verb graph*. As a result, given a set of verbs and adverbs, they were able to compute associated weights for computing the needed motions. Park et al. [Park et al., 2002] used a similar technique to provide a continuous control for real-time locomotion. Kovar and Gleicher [Kovar and Gleicher, 2003] then introduced *registration curves* to automatically compute allowed transitions between motions instead of doing it manually. Finally, Glardon et al. [Glardon et al., 2004] proposed a hierarchical space reduction based on Principal Components Analysis (PCA). They then linearly interpolated the weights in the PCA space to generate new walking and running motions. Mukay and Kuriyama [Mukai and Kuriyama, 2005] improved the RBF-based interpolation using geostatistics. Finally, Park et al. proposed in [Park et al., 2004] the *memory-based motion simulation*. Given a task to perform, their algorithm first extracts relevant preexisting motions from a motion database: the *root motions*. Each root motion is then adapted by correcting each animation curve independently. These latter are first segmented in strictly increasing parts, strictly decreasing parts and stationary parts. Each segment is then adapted to preserve the overall shape of the animation curves and to satisfy the desired goal.

Other methods focused on "attaching" motion clips one after each other to construct potentially infinite animations. These methods are known as *motion graphs* [Kovar et al., 2002a; Arikan and Forsyth, 2002; Lee et al., 2002]. The idea is to precompute a graph of motion clips to encode potentially acceptable transitions. These latter are automatically constructed by identifying frames where motions are similar. Then, they used path search algorithms to extract new motions satisfying user-defined constraints such as a posture at a specific time or a path to follow. Arikan et al. [Arikan et al., 2003] further extended this technique to take motion annotations into account. The user first needs to annotate few motions. Then, based on a Support Vector Machine classifier, new annotations are automatically added to the remainder of the database. They were able to generate motions that first wave, then run and finally walk. Finally, Reitsma and Pollard [Reitsma and Pollard, 2004] empirically evaluated the ability of a particular motion graph to reach different portions of the environment.

CHAPTER 3

Designing Postures Using Inverse Kinematics

Motion capture has recently become an appealing method to animate virtual characters. Indeed, one of its great advantages is its ability to rapidly produce very convincing motions with a limited amount of effort. However, the process leading from data acquisition to animation retargetting often introduces artifacts which the animators have to correct. Hence, animators still need important control over the results to correct and adjust any undesirable character's posture. For example, foot sliding is a common problem when dealing with motion captured animation. In fact, if the performer and the virtual character do not have the same limb size, such artifacts may occur and must be manually corrected.

One way to correct the posture of a virtual character is to use *forward kinematics*: the animator manually specifies the state vector (i.e. the posture) $\boldsymbol{\theta} = (\boldsymbol{p_r}, \boldsymbol{q_1}, \cdots, \boldsymbol{q_{n_{joints}}})$. The position of the end-effectors e we are interested in is then easily computed using the following kinematic equations:

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\theta}) \tag{3.1}$$

where $\boldsymbol{x}$ is the current position and/or orientation of the end-effectors and $\boldsymbol{f}$ the function which computes $\boldsymbol{x}$ given the state vector $\boldsymbol{\theta}$.

The computation of the solution is very fast as the function $\boldsymbol{f}$ is well-defined: it is the composition of all the transformations from the end-effector up to the root of the hierarchy. Unfortunately, forward kinematics inherently is a trial-and-error process which is tedious and

time consuming if the number of DoFs present in the virtual character is important (typically more than fifty for human models without considering the fingers). For example, if the animator needs to position the foot of a virtual character in order for it to be on the ground, he has to specify the joint values for the root of the hierarchy, the hip, the knee and finally the ankle. Then, if the foot cannot be correctly positioned because the joint value of the hip is not adequate for example, the animator needs to change all the joint values for all the joints from the hip down to the foot. Indeed, moving the hip also moves the knee, the ankle and the foot at the same time. Hence, the animator has to perform subsequent adjustments to obtain the expected results.

For that reason, tools to help animators have been extensively developed during the last years. In particular, *IK* provides capabilities to easily and rapidly define postures. Instead of specifying the value of each joint of the hierarchy by hand, the animator provides the desired position $\boldsymbol{x}$ of the end-effectors. The IK is then in charge of automatically determining a posture $\boldsymbol{\theta}$ that satisfies the specified tasks. We then need to solve:

$$\boldsymbol{\theta} = \boldsymbol{f}^{-1}(\boldsymbol{x}) \tag{3.2}$$

where $\boldsymbol{x}$ is the desired position of the end-effectors (i.e. the tasks) and $\boldsymbol{f}^{-1}$ the function that computes the state vector $\boldsymbol{\theta}$ given $\boldsymbol{x}$.

If there are less than seven DoFs in the hierarchy, then $\boldsymbol{f}^{-1}$ is well-defined and can be *analytically* computed. *Analytical IK* has the advantage of being extremely fast to compute. Furthermore, it leads to a finite number of solutions. As a result, all the solutions may be computed and the optimal solution can be chosen afterward depending on some predefined criteria (collision avoidance, energy consumption, closest one, etc).

However, if the hierarchy contains too many DoFs (typically more than seven) then $\boldsymbol{f}^{-1}$ is very difficult to compute. In this case, we need to use specific *numerical* methods to iteratively converge toward a solution. These methods work for any kind of hierarchy but are often more difficult to control than analytical ones. In particular, if there are few tasks with respect to the number of DoFs in the hierarchy, then the problem may have an infinite number of solutions: it is said to be *underconstrained* or *redundant*. We then need to choose the best solution among all the possibilities.

The control of complex articulated figures using *IK* often requires that we simultaneously apply multiple tasks. For example, a task may control the position of a hand to simulate a reaching action while another task controls the balance of the virtual human. These configurations may lead to conflicts between tasks because some are not achievable at the same time, whilst they can be separately. Conflicts may arise when one or more joints are shared by several tasks. Several strategies have been proposed to resolve those conflicts. A first possibility is to find a trade-off solution, where no task is achieved exactly, but each residual error is minimized. This solution consists in assigning a weight to each task to control the distribution of the residual error: this is known as the *weighting strategy*. A second possibility is to sort the tasks by order of priority, in order to satisfy the most important tasks first. This solution is known as the *priority strategy*.

In this chapter we first present related work on IK. Then, we introduce our *priority-based numerical IK solver* in Section 3.2. In Section 3.3 we present experimental results and

improvements (different parameterizations of complex articulations, different convergence methods, IK parameters, etc). We also present benchmarks we have performed to emphasize its good convergence properties. Finally we conclude this chapter by discussing its advantages and limitations with respect to previous approaches.

3.1 State of the Art in *Inverse Kinematics*

This section presents a state of the art of the major contributions in IK. Due to the vastness of this field (in robotics as well as in computer graphics), this review is not intended to be exhaustive but proposes some guidelines to help deciding which method is best suited depending on the problem we are dealing with.

3.1.1 Analytical Methods

It has been shown that for articulated figures containing up to six DoFs, the nonlinear kinematic equations are analytically solvable if the kinematic chain is carefully designed. For example, an articulated figure made of six revolute joints has an analytical solution if and only if three neighboring joint axes have an intersection [Paul, 1981; Craig, 1986]. Analytical resolution of the IK problem has several advantages:

1. The solution is very fast to compute. This is especially important in computer graphics since real-time interaction with virtual humans is an important issue when designing animation tools for animators.

2. The set of solutions is finite and then can be entirely computed. In robotics for example, it may be important to test all the possible solutions (that is all the postures of the robot manipulator satisfying the task) and then choose the best one to deal with external considerations such as collisions with the environment.

3. The solutions are repeatable: the set of solutions is always the same given a specific task (no matter what the initial posture is). The animators may then redesign the same posture even after numerous manipulations of the virtual human.

Korein presented in [Korein, 1985] a method to analytically solve the IK problem for anthropomorphic arms. The exposition deals with two problems: the control of the position only using four DoFs and the control of the position/orientation using seven DoFs. The algorithm may be decomposed as follows:

1. Compute the elbow joint angle respecting its joint limits.

2. Determine the circle to which the elbow is constrained to lie. This determine the interval for the *swivel angle*.

3. Compute the arcs of this circle respecting the joint limits of the shoulder (and the wrist while controlling the orientation). This gives acceptable intervals for the swivel angle.

4. Choose an adequate value for the swivel angle.

5. Finally solve the IK problem knowing the elbow position and angle.

Tolani presented a similar technique in [Tolani et al., 2000] to solve a part of the IK problem and used a numerical method to take into account joint limits (see Section 3.1.4).

3.1.2 Numerical Methods

Newton-Raphson Methods

The Newton-Raphson method is a technique to iteratively compute the solution of systems of nonlinear equations [Ortega and Rheinboldt, 2000]. In the IK context, the problem is then to find the root $\boldsymbol{\theta}^*$ of the following set of nonlinear equations given a good approximation of the solution $\boldsymbol{\theta}^0$:

$$\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{\theta}) - \boldsymbol{x} = 0 \tag{3.3}$$

Using a first-order approximation of $\boldsymbol{f}$ about $\boldsymbol{\theta}^0$ leads to a set of linear equations characterized by a Jacobian matrix relating differential changes of $\boldsymbol{\theta}$ to differential changes of $\boldsymbol{x}$. The system is then solved to find an approximation of the solution. It is important to note that due to the nonlinearity of $\boldsymbol{g}$, this solution is accurate only when a good estimation of the solution is used as the starting point. In other words $\boldsymbol{f}(\boldsymbol{\theta}^0) - \boldsymbol{x}$ must be sufficiently small to ensure that this method converges toward a good solution. As it is rarely the case, we have to solve sub-problems by converging toward intermediate goals, each intermediate solution being the starting point for the next sub-problem.

Resolved Motion Rate Control Methods

Whitney introduced in [Whitney, 1969] the *resolved motion rate control* method. Given a velocity task (speed and direction) for an end-effector, his approach computes the rates (angular speed) of the joints satisfying the task. Such a task could be to move along a straight line or a constant axis rotation. By differentiating the kinematic equations, he obtains a set of equations relating the velocity of the end-effector to the angular velocities of the joints thanks to the Jacobian. Finally, given the velocity of the end-effector (the task) and using a weighted pseudoinverse to invert the Jacobian, he solves the problem in order to find the adequate angular velocities of the joints. Numerous methods have extended this technique to deal with position tasks by integration of the velocities previously computed. Though this is not strictly the case, these methods are often also referred to as Resolved Motion Rate Control Methods.

Liégeois then proposed in [Liégeois, 1977] an extension of the general solution of the linearized kinematic equations. By exploiting the null space of the Jacobian, this method is used to minimize an additional criterion such as the deviation of the joints from their neutral posture (the middle of their range of motion). Following this, Klein and Huang [Klein and Huang, 1983] investigated the main drawbacks of using the pseudoinverse for controlling redundant manipulators. In particular, they showed that if one constrains the end-effector

onto a cycling path (a square in their example), the state vector cannot be known in advance for a specific location of the end-effector on that square. Even worse, the configuration of the kinematic chain tends to drift after each cycle. They proposed to use the redundant space left by the primary task to minimize an additional criterion in order to avoid these kinds of behaviors. Hanafusa et al. [Hanafusa et al., 1981] proposed an analysis of the manipulability and the redundancy of a kinematic chain. They qualitatively expressed the redundancy as being the set of vectors mapped to 0 by the Jacobian $\boldsymbol{J}$ (i.e. $N(\boldsymbol{J})$, the null space of $\boldsymbol{J}$). They then used this redundancy to solve the kinematic problem for two different tasks with order of priority. Nakamura et al. [Nakamura et al., 1987] presented the idea of task-priority when dealing with IK in more detail. Similarly, Maciejewski and Klein proposed in [Maciejewski and Klein, 1985] a formulation of the task-priority scheme with a simplified formula. The highest priority task is used to constrain the end-effector to follow a specified trajectory. The lower one is created in order for the *obstacle avoidance point* (the point of the chain which is closest to an obstacle) to be repulsed from the closest obstacle. Siciliano and Slotine [Siciliano and Slotine, 1991] generalized this concept to handle an arbitrary number of priority-tasks using a recursive formulation. Finally Baerlocher [Baerlocher and Boulic, 1998] proposed an efficient and recursive solution speeding up the computation of the projectors onto $N(\boldsymbol{J})$.

Singularities is an unavoidable problem when controlling articulated figures. Nakamura and Hanafusa [Nakamura and Hanafusa, 1986] introduced the singularity-robust inverse (or damped least squares inverse) as an alternative to the classical pseudoinverse to overcome the problem of singularities. Instead of minimizing the residual error at all costs (hence producing huge variations of the joint values near singularities) they proposed to simultaneously minimize the residual error *and* the solution norm. These two components are scaled in order for the solution's norm to have a high weight while in the proximity of singularities and the residual error to become the most important criteria to minimize while in a singular-free area of the configuration space. Maciejewski and Klein proposed in [Maciejewski and Klein, 1988] more sophisticated methods to dynamically determine the damping factor depending on the smallest singular value. This method has the advantage not to perturb the solution for well-conditioned configurations (i.e. far from a singular configuration) while retaining the important characteristics of the damped least squares solution on (or in the neighborhood of) a singular configuration. Finally, Chiaverini proposed in [Chiaverini, 1997] a new formulation of the task-priority resolution scheme to overcome the problem of algorithmic singularities. It is worth noting that this last mentioned paper makes the frequent mistake of using the damping least squares inverse to compute the projectors. In fact, important properties of the pseudoinverse do not hold anymore. This leads to a violation of the priority hierarchy as demonstrated in [Baerlocher and Boulic, 1998].

Jacobian Transpose Method

The Jacobian transpose method is quite similar to the resolved motion rate control one but instead of using a pseudoinverse of the Jacobian, it directly relies on its transpose. It has been introduced by Wolovich and Elliot in [Wolovich and Elliot, 1984]. Welman [Welman, 1993] discussed this method for interactive manipulation. Sciavicco and Siciliano [Sciavicco and Siciliano, 1988] extended the Jacobian transpose method to redundant manipulators. Hence,

they can handle collision avoidance as well as joint limits using the problem's redundancy. Das et al. [Das et al., 1988] also extended this method in order to satisfy a second criterion. However, while in singular configurations, they suggested to use a singular value decomposition as an alternative method for the projection onto the null space of the Jacobian. Doing so, they lose the advantage of using the transpose instead of the pseudoinverse of the Jacobian.

Each iteration of the Jacobian transpose method is very fast to compute since it does not need any computation of a pseudoinverse. However, this method is known to have very poor convergence characteristics and, for most manipulator configurations, it takes many more iterations before reaching a goal. As a consequence, the overall computation time is often worse than those of pseudoinverse-based methods.

3.1.3 Cyclic-Coordinate-Descent Method

The Cyclic-Coordinate-Descent (CCD) method is an iterative heuristic technique which minimizes an objective function by considering one joint at a time. Each iteration traverses the kinematic chain from the most distal joint to the base. Blow proposed in [Blow, 2002] an extension to the CCD to handle joint limits for an arm-like kinematic chain. As for the Jacobian transpose method, whilst the cost of a single iteration is very low, it suffers from poor convergence properties. Moreover, this method considers one joint at a time beginning with the most distal one. As a result, if the goal is close to the end-effector, only a few joints participate to the achievement of the task, leading to unpleasant configurations. Welman [Welman, 1993] also proposed a comparison between the Jacobian transpose and the CCD methods. Finally Kulpa et al. proposed in [Kulpa et al., 2005] a hierarchical CCD adapted to human-like figures in order to improve the realism of resulting postures. The skeleton is firstly subdivided into six groups of joints. A hierarchical CCD is then applied to each independent subgroup to ensure that only joints which are necessary to the tasks's achievement are actually used. Their CCD-based IK solver additionally handles priorities as well as CoM positioning.

3.1.4 Hybrid Methods

Lee and Shin presented in [Lee and Shin, 1999] a specialized IK solver dealing with human-like figures. The overall method is an optimization-based method which computes the joint value of a given a set of constraints. In order to reduce the number of variables to optimize for (hence speeding up the algorithm), the hierarchy is chopped so that joint values for the limbs are specifically solved using an analytical IK solver.

Tolani et al. [Tolani et al., 2000] presented a specialized IK method for Human-Arm-Like chains. Their algorithm uses a combination of analytical and numerical methods to deal with anthropomorphic limbs with seven DoFs. The redundant DoF is set by specifying the *swivel angle*. In cases where the goal is not reachable (because of joint limits for example), they first divide the problem into an analytical and a numerical part. They then use an optimization method to solve the reduced numerical problem.

Similarly, Shin et al. in [Shin et al., 2001] proposed an hybrid method to handle human-like articulated figures with 4 end-effectors (one for each limb) potentially having different

weights. They decomposed the IK problem into three sub-problems. The root position is first adjusted in order to minimize the weighted distance to the goals. Afterward, they ensure that all the goals are reachable given the new root position. To do so, the goals define 3D balls corresponding to the range of the root to ensure that they are reachable. Those 3D balls then define an intersection area that must contain the new root position. If it is not the case, then it is projected onto this surface. The body posture of the character is then optimized by minimizing an objective function. Finally, the posture of each limb is computed so that the goals are achieved. These two last steps are quite similar to the technique used in [Lee and Shin, 1999].

3.1.5 Conclusion

In the previous sections, we have reviewed the most important works on IK. In the next section, we present our IK solver. It is based on the resolved-motion rate control and this, for several reasons. First of all, we need to deal with any kind of hierarchies. As a consequence, analytical methods are not satisfactory as they are limited in the number of DoFs they can handle. Secondly, we need to achieve interactive rates. Jacobian transpose-based methods are known to have very bad convergence properties. Hence, this is clearly not a method to investigate. While the Cyclic-Coordinate-Descent method has proved to be reliable in some specific contexts, we believe that it is hardly usable in general. In particular, it often relies on some heuristics to compute the final solution. These heuristics are often directly dependent on the hierarchy. Finally, our IK solver uses priorities to arbitrate conflicts. Whilst weights may be used in such situations, priorities lead to a more intuitive solution as the tasks are hierarchically sorted to compute the final solution.

3.2 An HAnim Inverse Kinematics Solver

In this section, we first present the importance of using a standard such as *HAnim*. Then, the IK problem is stated using a numerical approach. Finally, we review the strategy to resolve conflicts between tasks: the *priority* strategy. Note that the original algorithm is presented in more detail in [Baerlocher, 2001]. However, for clarity purposes, it is important to summarize it in order for the reader to clearly understand the issues we are dealing with.

3.2.1 The *HAnim* Standard

HAnim is a standard which provides an abstract representation for modeling three dimensional human figures [HAnim]. Hence it allows for instance to exchange data between different HAnim-compliant applications. As we want to handle any kind of articulated structure (either human-like or multi-legged robots) we have thus adopted this standard.

The architecture of our IK solver is organized around HAnim-compliant data structures. This choice makes the algorithm more generic.

One of its most important advantages is the extension of the family of joint types. We are

now able to parametrize complex three dimensional articulations with a single exponential map. In prior approaches, these complex articulations were decomposed in a succession of three revolute joints with a common center of rotation. Although it is theoretically valid, this parameterization is subject to the *gimbal-lock* singularity [Watt and Watt, 1992][1]. Adopting the new joint types with their exponential map parameterization remove this singularity provided that we choose an adequate initial configuration for the articulation.

3.2.2 Inverse Kinematics Problem Statement

The IK problem can be stated as follows: given a kinematic chain parametrized by a state vector $\boldsymbol{\theta}$ with n_{joints} joints and n DoFs (with $n \geq n_{joints}$) and given a task to satisfy $\boldsymbol{x} = (\boldsymbol{x_1}, \cdots, \boldsymbol{x_m})$ with m_{tasks} sub-tasks and m the dimension of the so-called *task-space* (with $m \geq m_{tasks}$), determine a solution to the following system of nonlinear equations:

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\theta}) \tag{3.4}$$

where $\boldsymbol{x}$ and $\boldsymbol{f}$ are known.

Consider for example a kinematic chain made of n_{joints} joints parametrized using unit quaternions and a single task expressed in $\mathbf{SE}(3)$, then the function $\boldsymbol{f}$ may be defined as:

$$\begin{aligned} \boldsymbol{f} : \mathbf{R}^3 \times \mathbf{S}^{3n_{joints}} &\longrightarrow \mathbf{SE}(3) \\ (\boldsymbol{p_r}, \boldsymbol{q_1}, \cdots, \boldsymbol{q_{n_{joints}}}) &\longmapsto \boldsymbol{M_t}(\boldsymbol{p_r}) \prod_{i=n_{joints}}^{1} \boldsymbol{M_r}(\boldsymbol{q_i}) \end{aligned}$$

where $\boldsymbol{M_t}(\boldsymbol{p_r})$ is the transformation matrix corresponding to the global position of the root of the hierarchy and $\boldsymbol{M_r}(\boldsymbol{q_i})$ is the transformation matrix corresponding to the unit quaternion $\boldsymbol{q_i}$.

So, the problem could be thought of as determining the state vector $\boldsymbol{\theta}$ providing the function $\boldsymbol{f}$ and a task to satisfy $\boldsymbol{x}$. So, we must solve:

$$\boldsymbol{\theta} = \boldsymbol{f}^{-1}(\boldsymbol{x}) \tag{3.5}$$

As previously stated, the approach to solve this equation is quite different depending on the number of DoFs of the kinematic chain. If the kinematic chain is simple enough (i.e., contains less than seven DoFs), then an analytical resolution is well-suited [Paul, 1981; Craig, 1986; Tolani et al., 2000]. Otherwise, a numerical resolution of the IK problem must be considered. In our context, flexibility and genericity are major issues: we need to deal with arbitrary complex articulated figures (such as a whole virtual human) without restriction. Hence, we chose to solve the IK problem using a numerical algorithm.

[1] This occurs when the y-axis rotation is $\pm\pi/2$. We then lose one DoF as both x-axis and z-axis become aligned.

3.2.3 Inverse Kinematics Numerical Resolution

Overview of the Resolution Method

Equation (3.4) cannot be solved by simple inversion of the function $\boldsymbol{f}$ since it is generally nonlinear. Hence, we need to use more sophisticated methods. Our algorithm is based on the Newton-Raphson method which is similar to resolved motion rate. Each step i of the iterative algorithm may be summarized as follows:

1. Linearize function $\boldsymbol{f}$ about state vector $\boldsymbol{\theta}^i$.
2. Compute the Jacobian matrix $\boldsymbol{J}$ obtained during the linearization of function $\boldsymbol{f}$.
3. Solve the resulting set of linear equations for a small joint values increment $\Delta\boldsymbol{\theta}$ by inverting the Jacobian matrix. We must take special care when dealing with singular Jacobian matrices.
4. Take the new configuration $\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \Delta\boldsymbol{\theta}$ as the new starting point for the next step. Note that $\boldsymbol{f}(\boldsymbol{\theta}^{i+1})$ must be closer to the solution than $\boldsymbol{f}(\boldsymbol{\theta}^i)$, otherwise, the algorithm may not converge toward a solution.

These steps are iteratively repeated in order for the algorithm to finally converge toward a solution which locally satisfies the residual error of equation (3.5).

Linearization of equation (3.5)

Let h be a nonlinear function of one variable x and x^* be a root of h. We use the Newton-Raphson method to solve the nonlinear equation $h(x) = 0$. It consists in replacing the nonlinear function h at the iteration i by a linear function about the current configuration x^i. Hence, each iteration of the algorithm is defined as:

$$x^{i+1} = x^i - h'(x^i)^{-1}h(x^i) \tag{3.6}$$

where $h'(x^i)$ is the first derivative of function h at point x^i with respect to parameter x. If we consider the set of nonlinear equations expressed in equation (3.4), we want to find a solution for the following equation:

$$\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{\theta}) - \boldsymbol{x} = 0 \tag{3.7}$$

Using equation (3.6) we can find a solution to the previous equation using the Newton-Raphson algorithm:

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \boldsymbol{J}^{-1}(\boldsymbol{\theta}^i)\boldsymbol{g}(\boldsymbol{\theta}^i) \tag{3.8}$$

where $\boldsymbol{J}(\boldsymbol{\theta}^i)$ is the $m \times n$ Jacobian matrix of all first-order partial derivatives of function $\boldsymbol{f}$ (as well as $\boldsymbol{g}$ since $\boldsymbol{f}$ and $\boldsymbol{g}$ only differ about a constant vector which vanishes when computing the partial derivatives). Hence, given the *unknown* joint values increment $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}^{i+1} - \boldsymbol{\theta}^i$ we have:

$$\Delta\boldsymbol{\theta} = -\boldsymbol{J}^{-1}(\boldsymbol{\theta}^i)\boldsymbol{g}(\boldsymbol{\theta}^i) \tag{3.9}$$

Replacing $\boldsymbol{g}$ results in:

$$\Delta\boldsymbol{\theta} = -\boldsymbol{J}^{-1}(\boldsymbol{\theta}^i)(\boldsymbol{f}(\boldsymbol{\theta}^i) - \boldsymbol{x}) \tag{3.10}$$

Finally, using the *known* task increment $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{f}(\boldsymbol{\theta}^i)$ leads to:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^{-1}(\boldsymbol{\theta}^i)\Delta\boldsymbol{x} \tag{3.11}$$

Computing the Jacobian Matrix

The Jacobian matrix $\boldsymbol{J}$ maps differential changes of the joint values $d\theta$ to differential changes of the task coordinates dx and is a function of the current configuration $\boldsymbol{\theta}$. The task Jacobian matrices for revolute and ball-and-socket joints are summarized in [Baerlocher, 2001].

Solving the Linear System

Equation (3.11) can be solved in order to obtain a joint values increment $\Delta\boldsymbol{\theta}$ after having computed the Jacobian matrix $\boldsymbol{J}$ and a desired task increment $\Delta\boldsymbol{x}$. However, this system is usually underconstrained as the dimension of the task space is often lower than the one of the joint space. As a result, the Jacobian is not directly invertible. Hence, instead of using the inverse $\boldsymbol{J}^{-1}$ of the Jacobian matrix $\boldsymbol{J}$, we use the pseudoinverse $\boldsymbol{J}^{\dagger}$.

Moreover, the solution space can be partitioned into two orthogonal subspaces: the null space of $\boldsymbol{J}$, $N(\boldsymbol{J})$, which provides the set of solutions that do not contribute to the satisfaction of the problem, and its orthogonal complement $N(\boldsymbol{J})^{\perp}$ which of course, contributes. Hence, a general solution is composed of two terms: a particular solution with minimum norm (belonging to $N(\boldsymbol{J})^{\perp}$) to satisfy equation (3.11) as well as possible, and a homogeneous solution (belonging to $N(\boldsymbol{J})$) which can be used to satisfy an additional criterion. Hence, the problem can be summarized to solving the linear equation:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}^i)\Delta\boldsymbol{x} + \boldsymbol{P}_{N(\boldsymbol{J})}\boldsymbol{z} \tag{3.12}$$

where $\boldsymbol{P}_{N(\boldsymbol{J})}$ is the orthogonal projection operator onto $N(\boldsymbol{J})$ and $\boldsymbol{z}$ an arbitrary vector. The vector $\boldsymbol{z}$ can be used to optimize a specific criterion such as minimizing the distance to a reference configuration [Liégeois, 1977].

Choice of Intermediate Goals

As stated before, the Newton-Raphson method must be initialized with a good starting point because of the nonlinearity of function $\boldsymbol{f}$. In other words the difference between the current state $\boldsymbol{f}(\boldsymbol{\theta})$ and the task to achieve $\boldsymbol{x}$ must be small. This is rarely the case as in common situations, the end-effector is far from the goal to be achieved. The *tracking error* defined as being $\Delta\boldsymbol{x} - (\boldsymbol{f}(\boldsymbol{\theta}^{i+1}) - \boldsymbol{f}(\boldsymbol{\theta}^i))$ is thus too large: the algorithm may not converge to a satisfactory solution. Hence, we need to divide the problem into smaller ones by choosing intermediate goals between the starting location of the end-effectors and the task to be achieved $\boldsymbol{x}$ in order for $\Delta\boldsymbol{x}$ to be sufficiently small. Press et al. [Press et al., 1992] proposed the *line search and backtracking* method which consists in scaling down the solution until the tracking error is below a given threshold. Watt and Watt described in [Watt and Watt,

1992] a trial and error method: if the tracking error is too important, the task increment $\Delta \boldsymbol{x}$ is subdivided and the solution recomputed. This process is repeated until the tracking error is acceptable. While these methods generally require less inversions of the Jacobian matrix, they need additional evaluations of the function $\boldsymbol{f}$. Hence, the selected method is to use empirically computed fixed-length steps instead. This method has the advantage of being simple [Baerlocher, 2001].

3.2.4 Damping the Solution

The problem becomes ill-conditioned in the proximity of singularities: the norm of the resulting solution $\Delta \boldsymbol{\theta}$ tends to infinity [Maciejewski, 1990]. This is unacceptable as it violates the small increments hypothesis of equation (3.11) and tends to result in undesired behaviors of the kinematic chain. For this reason we must use a *damping factor* λ to constrain the norm of the solution to remain under a specific threshold. We discuss the damping factor impact in Section 3.3.2.2.

Hence, the following weighted combination of tracking error and solution norm is minimized instead of minimizing the tracking error alone:

$$\|\boldsymbol{J}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} - \Delta\boldsymbol{x}\|^2 + \lambda^2 \|\Delta\boldsymbol{\theta}\|^2 \tag{3.13}$$

where λ weights the relative importance of tracking error versus norm of the solution. When $\lambda = 0$, only the tracking error is minimized and it is equivalent to the previous solution. As a consequence, when λ increases, the solution norm is forced to decrease.

To minimize the error of equation (3.13) we use the so-called *damped least squares inverse*. This technique has been proposed by Nakamura et al. [Nakamura and Hanafusa, 1986]. The damped least squares inverse of $\boldsymbol{J}$ is defined as follows when $\lambda > 0$:

$$\boldsymbol{J}^{\dagger^\lambda} = \boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{J}\boldsymbol{J}^{\boldsymbol{T}} + \lambda^2 \boldsymbol{I}_{\boldsymbol{m}})^{-1} \tag{3.14}$$

Finally, using damped least squares inverse, the IK problem can be stated as solving the following linear equation:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^{\dagger^\lambda}(\boldsymbol{\theta}^i)\Delta\boldsymbol{x} + \boldsymbol{P}_{N(\boldsymbol{J})}\boldsymbol{z} \tag{3.15}$$

The next section presents our algorithm to solve conflicts between tasks.

3.2.5 Dealing with Conflicting Tasks: the Priority Strategy

When manipulating multiple tasks, it may happen that some of them cannot be achieved simultaneously while they can separately: these tasks are said to be in conflict. Two main methods have been proposed to resolve these conflicts: the *weighting* and the *priority* strategies. The weighting strategy tries to minimize the distribution of the residual error among the tasks using weighted tasks. This method leads to a compromise where none of the tasks is precisely achieved. On the other hand, the priority strategy uses priority layers to reflect

relative importance between tasks. The tasks belonging to the highest priority layer are enforced first. Then, those of the next priority layer are satisfied as much as possible without disturbing the previous ones, and so on. Figure 3.1 conceptually explains the difference between weighting and priority strategies. Moreover, Figure 3.2 shows a practical example of conflicting tasks and the solution depending on the methods. Consider the conceptual example of Figure 3.1 using three tasks, T_1, T_2 and T_3 where priority T_1 < priority T_2 < priority T_3. The solution belongs to the solution set of T_3. All solutions on the green arc minimize the residual error to T_2. The resulting solution is then the point from the green arc minimizing the residual error to T_1. Conversely, the weighting strategy leads to a compromise solution (shown in red) where no task is achieved.

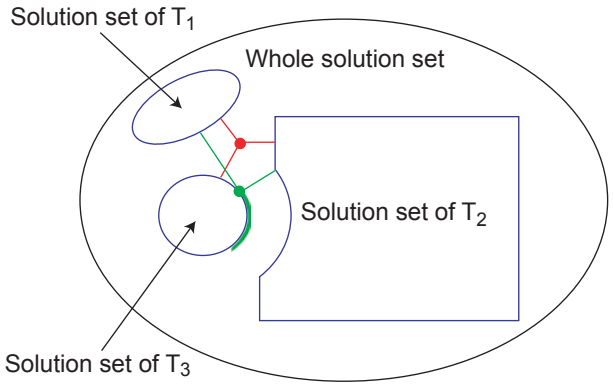

Figure 3.1: Conceptual illustration of the difference between weighting and priority strategies (priority T_1 < priority T_2 < priority T_3). **Green**: Solution using priority strategy. **Red:** Solution using weighting strategy.

In general, it is unpleasant for an animator to obtain a compromise between conflicting tasks. He should be able to specify which tasks should be achieved first and which ones are less important. For example, if we consider a collision avoidance task, it is obvious that it should have top priority and must be achieved regardless of the other tasks. Clearly, the weighting method is not always appropriate to handle multiple tasks: the animator should be able to impose relative priority between tasks.

The priority method is an extension of the numerical method presented in the previous

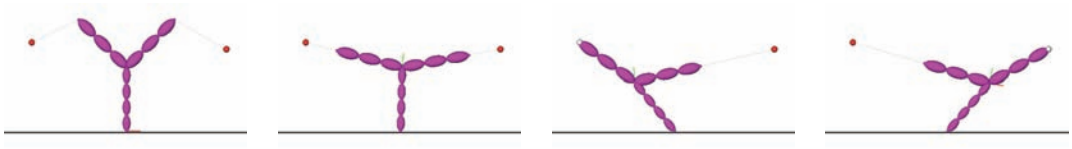

Figure 3.2: Comparison between weighting and priority strategies. **Left:** Initial configuration with conflicting tasks. **Middle left:** Solution using weighting strategy to solve conflicts. **Middle right:** Solution using priority strategy to solve conflicts (the left task has a *higher* priority than the right one). **Right:** Solution using priority strategy to solve conflicts (the left task has a *lower* priority than the right one).

sections. It recursively exploits the null space of the Jacobian of each priority layer.

We have shown in Section 3.2.3 that it is possible to exploit the redundancy of the problem to minimize a specified cost function represented by a (potentially arbitrary) vector z. Indeed, by projecting this vector onto the null space $N(\boldsymbol{J})$ of the Jacobian matrix $\boldsymbol{J}$, we can ensure that we do not disturb the specified tasks while minimizing the cost function as much as possible. It is also possible to exploit this vector to express a secondary task with a lower priority.

Suppose we have two tasks T_1 and T_2 to achieve, T_1 being more important than T_2. Then we must resolve two different equations (one for each task):

$$\boldsymbol{J_1}\Delta\boldsymbol{\theta} = \Delta\boldsymbol{x}_1 \quad \text{and} \quad \boldsymbol{J_2}\Delta\boldsymbol{\theta} = \Delta\boldsymbol{x}_2$$

As task T_2 is less important than task T_1, we have to project its solution onto the null space of $\boldsymbol{J_1}$ to ensure that it does not disturb the solution for task T_1. Moreover, the solution for the task T_1 participates (in a positive or negative way) to the achievement of task T_2. We must then compensate for this in the secondary task. Thus, more formally, using (3.15) we must solve:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1 + \boldsymbol{P}_{N(\boldsymbol{J_1})}(\boldsymbol{J_2}^{\dagger^{\lambda_2}}(\Delta\boldsymbol{x}_2 - \boldsymbol{J_2}\boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1))$$

where $\Delta\boldsymbol{x}_2 - \boldsymbol{J_2}\boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1$ is the new secondary task taking into account the displacement $\boldsymbol{J_2}\boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1$ due to the achievement of the first task. However, while this solution is mathematically correct, it badly tracks the secondary task. Indeed, this equation finds the best solution satisfying task T_2 and projects it onto the null space of $\boldsymbol{J_1}$ afterward. Hence, the final solution for task T_2 is not necessarily the best one after restriction to the null space of $\boldsymbol{J_1}$. A better solution is to directly compute the solution after restriction to the null space of $\boldsymbol{J_1}$. We then have to solve:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1 + (\boldsymbol{J_2}\boldsymbol{P}_{N(\boldsymbol{J_1})})^{\dagger^{\lambda_2}}(\Delta\boldsymbol{x}_2 - \boldsymbol{J_2}\boldsymbol{J_1}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1) \tag{3.16}$$

This approach can be generalized to an arbitrary number of tasks with multiple levels of priority. Consider t tasks T_p ordered from the highest priority ($p = 1$) to the lowest ($p = t$). Without loss of generality, we consider that they all have different priorities (since two tasks with the same priority level may be merged, to form a single *augmented* task). Siciliano and Slotine [Siciliano and Slotine, 1991] proposed an efficient and recursive scheme dealing with an arbitrary number of priorities. Baerlocher [Baerlocher, 2001] extended this method to speed up the projectors computation. Hence, the final algorithm is summarized as follows:

$$\begin{aligned}
\Delta\boldsymbol{\theta}_p &= \Delta\boldsymbol{\theta}_{p-1} + \widetilde{\boldsymbol{J}}_{\boldsymbol{p}}^{\dagger^{\lambda_p}}(\Delta\boldsymbol{x}_p - \boldsymbol{J_p}\Delta\boldsymbol{\theta}_{p-1}) \\
\Delta\boldsymbol{\theta}_1 &= \widetilde{\boldsymbol{J}}_{\boldsymbol{1}}^{\dagger^{\lambda_1}}\Delta\boldsymbol{x}_1
\end{aligned}$$

$$\begin{aligned}
&\text{with } \widetilde{\boldsymbol{J}}_{\boldsymbol{p}} = \boldsymbol{J_p}\boldsymbol{P}_{N(\boldsymbol{J}^{\boldsymbol{A}}_{\boldsymbol{p-1}})} \\
&\text{where } \boldsymbol{P}_{N(\boldsymbol{J}^{\boldsymbol{A}}_{\boldsymbol{p}})} = \boldsymbol{P}_{N(\boldsymbol{J}^{\boldsymbol{A}}_{\boldsymbol{p-1}})} - \widetilde{\boldsymbol{J}}^{\dagger}_{\boldsymbol{p}}\widetilde{\boldsymbol{J}}_{\boldsymbol{p}} \\
&\text{and } \boldsymbol{P}_{N(\boldsymbol{J}^{\boldsymbol{A}}_{\boldsymbol{0}})} = \boldsymbol{I_n}
\end{aligned}$$

$\boldsymbol{J}^{\boldsymbol{A}}_{\boldsymbol{p}}$ is the *augmented Jacobian* defined as:

$$\boldsymbol{J_p^A} = \begin{bmatrix} \boldsymbol{J_1} \\ \boldsymbol{J_2} \\ \vdots \\ \boldsymbol{J_p} \end{bmatrix} \tag{3.17}$$

Each step of this algorithm may be thought of as:

1. Adjust the desired task increment $\Delta \boldsymbol{x}$ in order to compensate for the displacement due to the partial solutions of higher priority levels.
2. Search the best partial solution for the adjusted task increment $\Delta \widehat{x_p}$. The search is restricted to the null space of $\boldsymbol{J_{p-1}^A}$ to ensure the enforcement of the hierarchy of priorities.
3. Update the projector by removing the range of $\widetilde{\boldsymbol{J_p^T}}$ (i.e by projecting the subsequent Jacobians onto its null space) in order for so that it to no longer be available for tasks of lower priority.

The last step consists in minimizing an additional criterion to take advantage of any remaining redundancy.

With this framework, we are now able to find a solution to the IK problem. The next section shows the application resulting from the implementation of this algorithm. This application provides tools to test and to enhance our IK solver. Moreover, some benchmarks have been performed on several key parameters of the IK solver. An implementation of the *transpose Jacobian* method is also available in order to perform comparisons with the numerical resolution we have implemented.

3.3 Experimental Results: *HBalance*

In this section, we present tests and benchmarks we have done using the test-platform we have implemented: *HBalance*. This framework was necessary for several reasons:

- To confirm the correctness of the established Jacobians implementation,
- To evaluate the influence of key parameters variations on the behavior of the IK solver,
- To compare robustness and computing performances with respect to prior approaches (Euler-angle based representation of complex joints, Jacobian transpose method,...).

HBalance provides the user with almost all the features of the IK. For example, we can control the position/orientation of different effectors, control the position of the CoM of the virtual human, weight the influence of each articulation during the convergence process, determine the number of joints recruited to satisfy a given task, etc.

In prior approaches, complex three dimensional articulations such as the shoulder were decomposed into three simple *revolute* joints (one DoF), each of them being parametrized

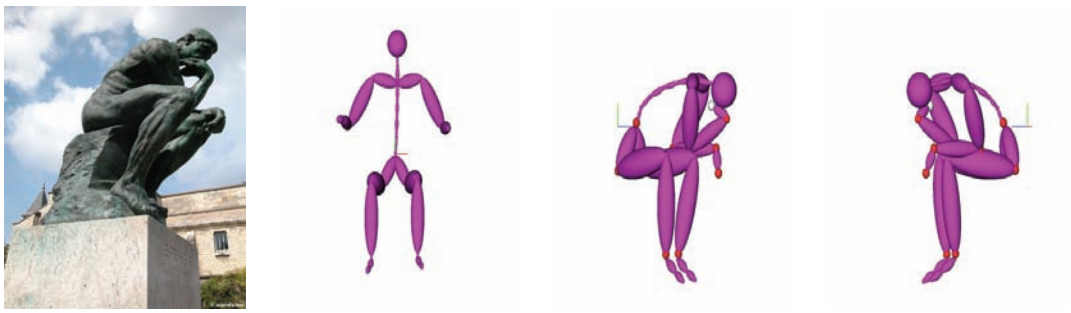

Figure 3.3: Using *HBalance* to design postures. **Left:** The Thinker by Auguste Rodin. **Middle left:** Initial configuration we started with. **Middle right and right:** Two views of the designed posture. **Remark:** The final designed posture is not exactly the same as the model due to the difference in proportions between "The Thinker" and our virtual character.

by a single axis of rotation. We then obtained the equivalent of the Euler angles parameterization.

The IK solver previously presented is now applied to joint hierarchies compliant with the *HAnim* standard. This standard provides in particular the possibility to represent complex three dimensional articulations with a unique joint parametrized by an exponential map. Hence, the *gimbal-lock* singularity can now be avoided. Furthermore, the exponential map parameterization is more intuitive to represent joint limits. Performing some comparisons between chains of joints containing exactly the same number of DoFs but using these two different parameterizations was essential to highlight the advantages of representing a complex three dimensional articulation with a single joint instead of a chain of *revolute* joints.

Our IK solver needs some *key parameters* (and more particularly an *integration step* and a *damping factor*) to converge toward a goal. Thus, it seems essential to be able to precisely tune these variables to well-understand their respective meaning and their relative influence. For this reason, this application also provides capabilities to modify these parameters in order to evaluate their respective influence during convergence process.

Finally, to stress the overall advantages of good convergence properties over low computational cost of our method, this application contains an implementation of the classical Jacobian transpose method.

3.3.1 Postures Design

HBalance is an application based on and made for the IK solver previously exposed. It provides fundamental features such as the control of effectors in position and/or orientation, CoM position control, joint limits, joint weighting, joint recruiting, etc.

3.3.1.1 End-Effectors Position and Orientation Control

One of the most important features of an IK solver is to precisely control end-effectors position and orientation. This is very useful in a number of applications such as:

- **Posture editing:** as previously stated, it is faster and easier for an animator to precisely design postures for virtual humans using IK. The posture editing may be used in animation to define *keyframes* or in the industry to check the reachability of different objects in a car or even to visualize the different postures obtained with the same task but for different virtual humans (and potentially different limb sizes).

- **Motion editing:** during the deformation process, motion editing algorithms induce artifacts (foot sliding, loss of environmental constraints, etc) to the initial motion. IK gives a great flexibility to such algorithms as it provides an easy way to impose specific constraints in order to ensure that important motion characteristics are preserved before and after the deformation. We discuss the use of IK for motion editing in chapter 5.

In this sense, it is extremely important for an IK solver to give the animators the possibility to robustly position and orient end-effectors.

Figure 3.3 shows an example of a posture designed using *HBalance*. Given a starting sitting posture, four tasks were required for the positioning of the legs, one to precisely position the pelvis, four to design the arms postures and two to constrain the orientation of the hands. The tasks controlling the legs were given a high priority. Those of the arms were given a medium one while those controlling the orientation were considered as the least important ones. See Appendix B.1 for a complete description of the set of tasks.

It is important to note that in our implementation, the hierarchy is subdivided into two main parts: the upper body (the spine, the arms and the head) and the lower body (the legs). Indeed, the root of the hierarchy is positioned at the pelvis in order for it to be the only joint which can translate. As a result, in this classical "reach while sitting" example, the root of the hierarchy (i.e. the pelvis) has to be recruited while the thighs are constrained to specific locations to accurately simulate the "bend forward" posture.

3.3.1.2 Center of Mass Position Control

To remain realistic, postures computed using IK should stay balanced as much as possible. As a consequence, the CoM position control has become of real interest with the development of virtual humans. Using the *inverse kinetics* method presented by Boulic et al. in [Boulic et al., 1996] it is now possible to precisely and realistically control the CoM position to ensure that the virtual character is balanced. Hence, given an articulated figure and a mass distribution, our algorithm is able to take into account a position constraint on the CoM. Figure 3.4 shows an example of postures with and without CoM control.

The starting configuration is the HAnim standard posture (upper left of Figure 3.4) and is also used as the reference configuration to force the character to remain as close as possible to its starting posture. The right foot is constrained to stay on the ground using two high-priority constraints. The right hand is constrained to reach a location far beyond the reachable space of the virtual character. See Appendix B.2 for a complete description of the set of tasks. As shown in the upper right of Figure 3.4, the final posture obtained without constraining the CoM position is not realistic. Hence, we use a task which constrains the CoM of the virtual character to project between its feet. As a consequence, the final posture is balanced and the results are more realistic (bottom left of Figure 3.4). Finally, an object

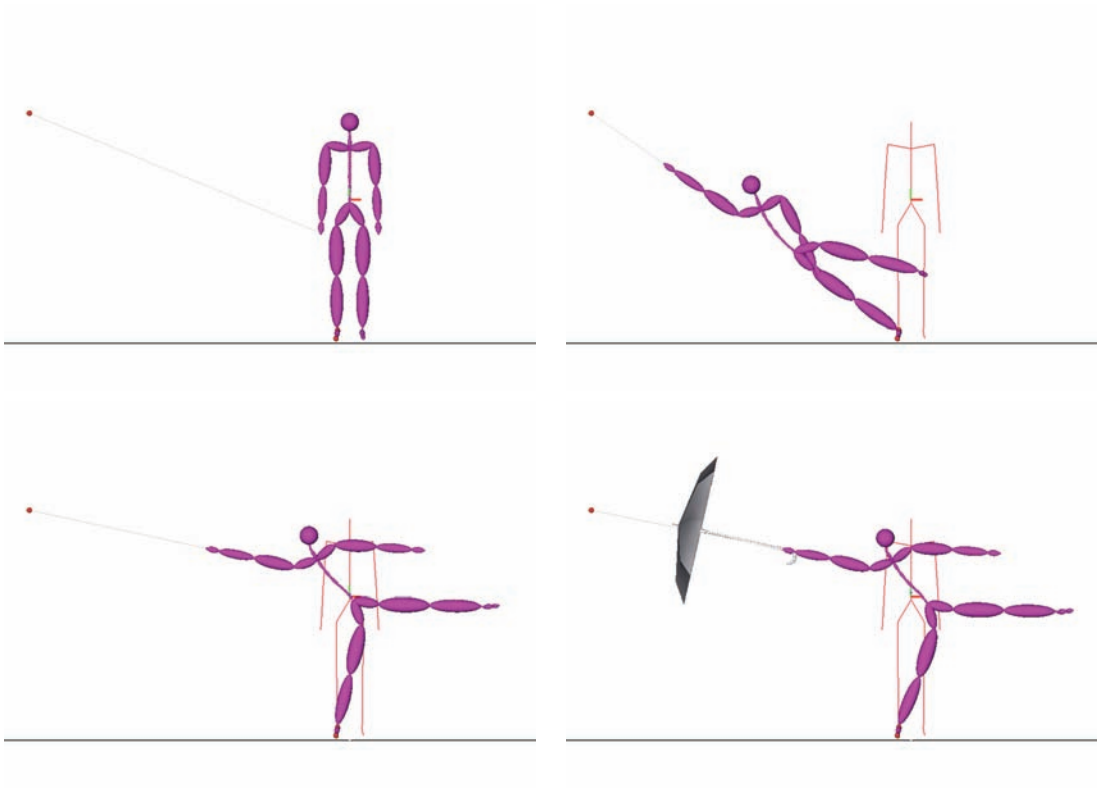

Figure 3.4: Example of a designed posture with *HBalance*: it immediately shows that a precise CoM position control results in more realistic postures. **Upper left:** Initial configuration. **Upper right:** Final posture without any CoM control. **Bottom left:** Final posture with CoM control. **Bottom right:** Final posture with CoM control. In addition, the virtual character is carrying an umbrella.

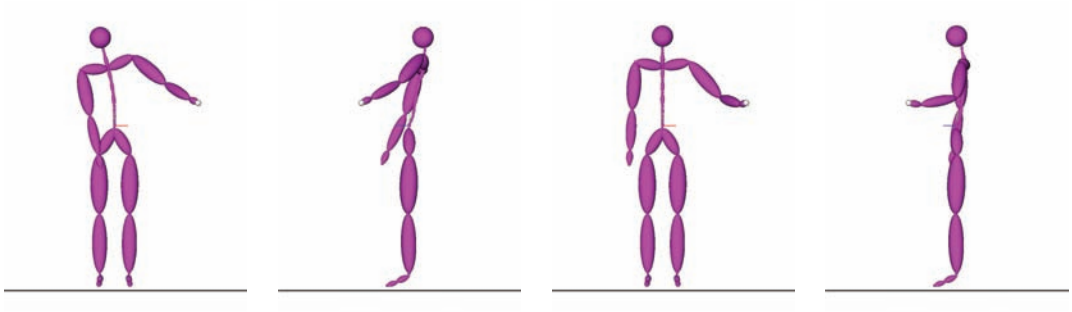

Figure 3.5: Examples of a reaching task with different joint recruiting levels. **Left:** The task uses the left arm as well as the spine to achieve the specified task (front and side views). **right:** The task uses the left arm only to achieve the specified task (front and side views).

is added to emphasize the influence of the CoM position onto the final posture (bottom right of Figure 3.4). Note the difference in posture to compensate for the additional weight of the umbrella. It is important to note that we do not consider any dynamic information during the computation of the virtual character's posture. As a consequence, our IK solver may sometimes generate solutions which contain excessive joint torques for common humans.

3.3.1.3 Joint Recruiting Level

A common approach in IK techniques is to directly consider the underlying structure. For example, human-like characters can be advantageously partitioned into smaller sub-hierarchies (limbs, spine,...). These latter are then controlled in a more efficient way by an IK solver dedicated to this problem. As we aim at providing a general framework to control any kind of hierarchy, this specialization is not satisfactory. Conversely, considering all the possible joints to control the position and/or orientation of an end-effector may prove to be counter-productive. While controlling the position of the wrist for example, it is important to choose whether we want the spine to participate or not. Thus, our IK solver allows a task to *recruit* all or part of the joints from its parent joint up to the root of the hierarchy. This is very useful for the animators as they can manually discard joints that should not participate to the achievement of the task. We can then easily define kinematic chains of varying lengths providing the user with more control over the final results. Figure 3.5 shows different postures obtained by changing the joint recruiting level.

The joint recruiting is submitted to one strict design rule which guarantees the enforcement of the hierarchy of priorities. It concerns those parts of the skeleton where multiple tasks may recruit part of their joints; in these regions, the rule requires joint sets associated to high-priority tasks to include any joint set associated to lower priority tasks. Failing to do so may lead to diverging solutions.

The problem of multiple overlapping kinematic chains was first addressed in [Badler et al., 1980]. However, instead of using an algorithmic scheme to solve this problem, we allow joints to be shared by multiple tasks as long as the following minimal recruiting rule is met: let T_i be a task of priority i, $Rec(T_i)$ its associated set of recruited joints and $Anc(T_i)$

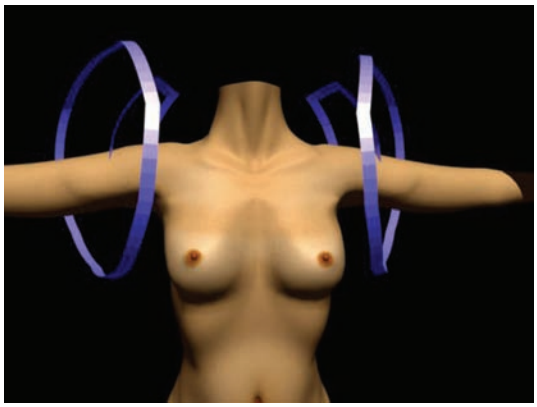

Figure 3.6: Shoulders joint limits using spherical polygons.

all the joints that may be recruited by T_i (its ancestors). Then, for priority levels $a > b$ we must have:

$$Rec(T_b) \cap Anc(T_a) \subset Rec(T_a) \tag{3.18}$$

This rule could be checked at task definition time and solutions could be proposed to the animator if it is not met.

3.3.2 Benchmarking

3.3.2.1 Exponential Map Versus Euler Angles

In computer graphics, virtual characters are often represented as hierarchies of joints linked together by segments. However, prior approaches decompose complex three dimensional articulations such as the shoulder into three simple *revolute* joints (one DoF), each of them being parametrized by a single axis of rotation. We then obtain the equivalent of the Euler angles parameterization. In this way, an articulation which was parametrized by a quaternion or an exponential map is transformed into a chain of simple articulations parametrized by Euler angles. However, although this method is valid in theory, it presents two major drawbacks. Indeed, it is difficult to intuitively and effectively represent the domain of possible configurations reachable for this kind of articulation. Moreover, it suffers from the *gimbal-lock* singularity.

As our IK solver is based on the *HAnim* standard, we are now able to parameterize complex three dimensional articulations directly with a single exponential map. This method is much more intuitive and efficient to represent the range of reachable configurations by an articulation using spherical polygons or elliptic cones as shown in Figure 3.6. The exponential map parameterization is also well-suited to avoid singularities if we choose an adequate initial position for the considered articulation beforehand in order for the anatomical constraints to prevent the articulation from being in the neighborhood of a singular configuration. Consider an exponential map $e_r = [s_x, s_y, s_z]$. As stated in [Baerlocher, 2001] and later in [Aubel, 2002], the exponential map has a singularity in the direction $d = [0, 0, -1]^T$ with $s_x^2 + s_y^2 = \pi^2$. This means that if we are to avoid singularities, we have to find a *zero position*

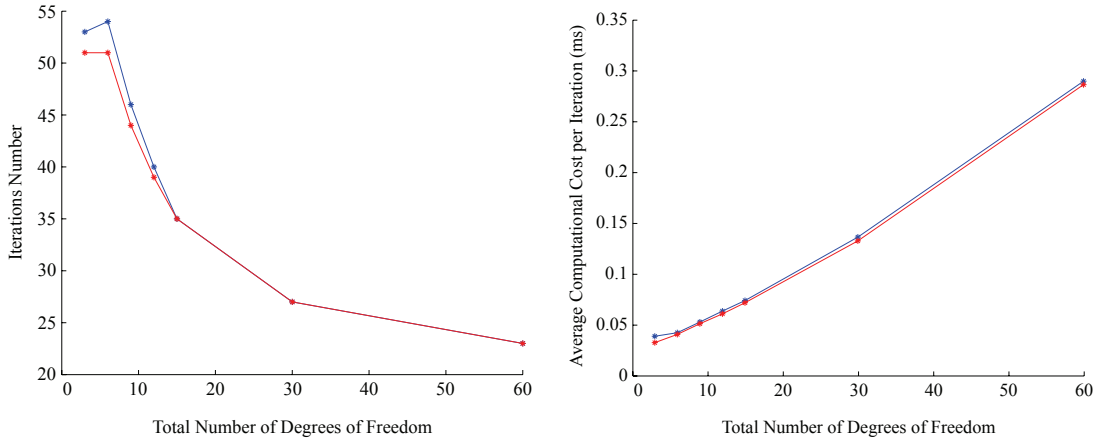

Figure 3.7: Benchmarks performed depending on the type of joints used. **Red Curve:** Revolute joints. **Blue Curve:** Ball-and-Socket joints. **Left:** iterations number comparison. **Right:** Computational cost comparison.

in order for joint limits to prevent the articulation from reaching this direction. For a "normally built" human, this is quite simple but has to be done on a joint basis. For example, the shoulder's zero position in our framework corresponds to an abduction of $\pi/2$ with respect to the HAnim standard [Aubel, 2002].

Performing some comparisons between chains of joints containing exactly the same number of DoFs but using these two different parameterizations was essential to highlight the advantages of representing complex three dimensional articulations with *ball-and-sockets* joints instead of chains of *revolute* joints. We must first notice that the expression of the Jacobian matrix for the exponential map parameterization is much more complex than the one for the Euler angles (see [Baerlocher, 2001] for details). However, this computation is performed only once for a three dimensional articulation parameterized with an exponential map while it is performed three times (one for each axis of rotation) for the same three dimensional articulation parameterized with Euler-angles. Furthermore, as the bottleneck of the algorithm is the inversion of the Jacobian matrix, its dimension is a critical issue. However, the Jacobian matrix has the same dimensions in both parameterizations as exponential map and Euler angles are both represented with three dimensional vectors.

The benchmarking has been performed on a Xeon, 3.2GHz, 1Go RAM. We have tested our IK solver using eight different goals for kinematic chains with varying numbers of DoFs. The results have then been averaged to estimate the mean of the iterations number to reach one goal and the computational cost for one iteration. Figure 3.7 shows that even if the exponential map parameterization is more complex than the Euler angles parameterization, and more particularly for the computation of the Jacobian matrix, the performances in terms of final computation time and iterations number are similar: the difference in terms of computational cost is essentially due to function call overhead. As a conclusion, this parameterization provides an intuitive way to represent joint limits, is singularity-free for well-chosen initial configurations and is similar to the Euler angles parameterization in terms of convergence and efficiency.

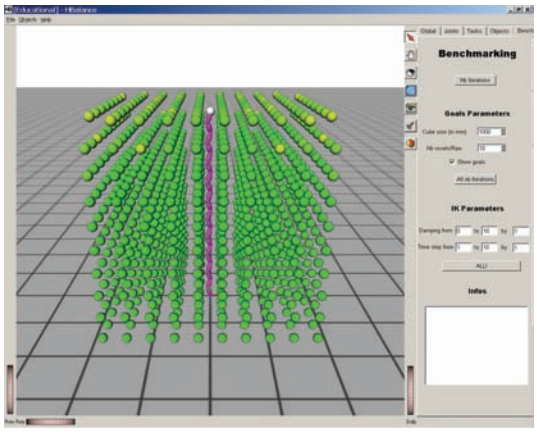

Figure 3.8: Test configuration to analyze the impact of *integration step* and *damping factor* in the IK solver.

3.3.2.2 Integration Step and Damping Factor

As already stated in Section 3.2.4, the problem becomes ill-conditioned in the proximity of singularities. As a result, the solution norm tends to infinity. To overcome this problem, we use a regularization technique called *damped least squares*. This technique introduces a parameter λ: the so-called *damping factor*. As λ increases, the solution norm is forced to decrease. Thus, the higher the damping factor, the more regular the convergence but the longer the process. Hence, it is important to find a good trade-off between regularity and computational cost.

Furthermore, the results we obtain after solving equation (3.5) represent joint velocities. As a consequence, we need to integrate the solution over time using an integration step to find the new joint values increments. Hence, another important variable of our IK solver is the *integration step*. If it is too high, it violates the condition of small increments and the process can not converge at all. On the other hand, if it is to small, it slows down the convergence at performances cost.

One important issue was therefore to compare the mutual influence of the integration step and the damping factor on our IK solver in terms of iterations number.

The test configuration we retained is shown in Figure 3.8. We chose this configuration for several reasons:

1. **A simple kinematic chain:** the kinematic chain is one meter long and contains twenty Ball-and-Socket joints for a total of sixty DoFs. We could have used a human limb instead. However, this is too specific and would have led to other issues biasing the results we are interested in. Indeed, the topology, the joint limits and/or the type of joints (i.e their DoFs) of the arm (or leg) directly influence the convergence of the IK solver. Consider the arm for example: the iterations numbers are very dissimilar from one task to another even when they are close to each other.

2. **A non-singular starting posture:** each joint of the kinematic chain is initialized to a non-null rotation so that we do not begin in a singular configuration.

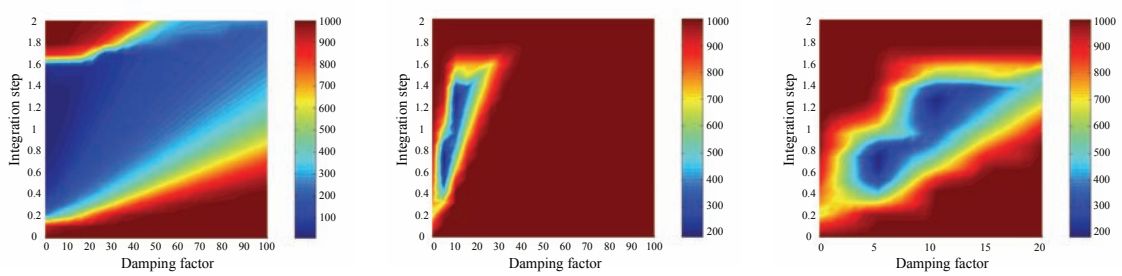

Figure 3.9: Influence of the *integration step* and the *damping factor* on the convergence process. The color scale indicates the iterations number. **Left:** Tests using reachable goals (no singular configuration). **Middle:** Tests using unreachable goals (singular configurations may occur). **Right:** Same diagram as the middle one with 0<damping factor<20.

3. **several tasks:** instead of choosing a single task to analyze the influence of the integration step and the damping factor, we discretize the cube surrounding the kinematic chain into one thousand voxels (its edge length is defined as the length of the kinematic chain). The center of each voxel then represents a task to satisfy.

   For each pair of parameters (integration step, damping factor), we then run the IK solver for all the tasks. We consider that the IK solver has converged if the kinematic chain is stable (i.e. the current configuration and the previous one are identical) *and* if the distance between the end-effector and the goal is minimal (0 if the goal is reachable). Moreover, we stop the IK solver after one thousand iterations.

   Finally the results are averaged between reachable goals (singularity-free configurations) and non-reachable goals (potentially leading to singular configurations).

Figures 3.9 and 3.10 summarize the results we obtained. Blue areas locate "*good convergence*" configurations while red ones emphasize *no convergence* configurations: the solver was unable to converge before the limited iterations number. We can notice that:

1. Very high values of the integration step (more than 1.6 in our examples) lead to instabilities preventing the kinematic chain to reach the specified goals before the maximum iterations number.

2. Very high values of the damping factor (more than 40 in the current examples) lead to poor convergence properties preventing the kinematic chain from reaching the specified goals before the maximum iterations number.

3. As expected, the damping factor has a real impact onto the convergence for singular configurations while it is less important in non-singular ones. This is particularly noticeable in cases where the damping factor is null. In singular configurations, the algorithm is not able to converge while it is for reachable goals.

As shown in Figures 3.9 and 3.10, the iterations number depending on the IK parameters is similar for both hierarchies. Indeed, they only differ by a scaling factor. However, different

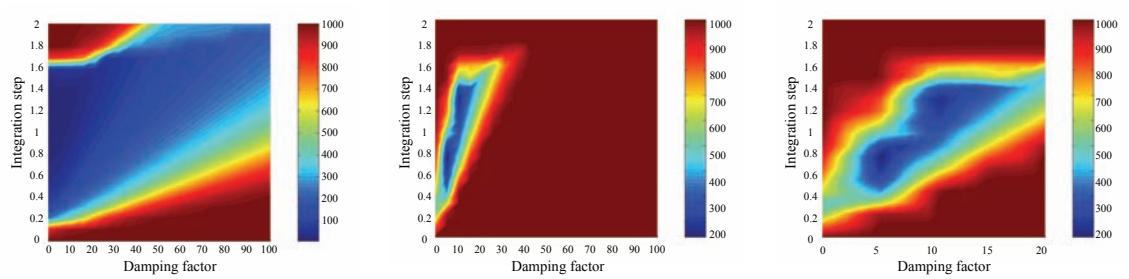

Figure 3.10: Same benchmarks as in Figure 3.9 but with a kinematic chain twice smaller (fifty centimeters long). **Left:** Tests using reachable goals (no singular configuration). **Middle:** Tests using unreachable goals (singular configurations may occur). **Right:** Same diagram as the middle one with 0<damping factor<20.

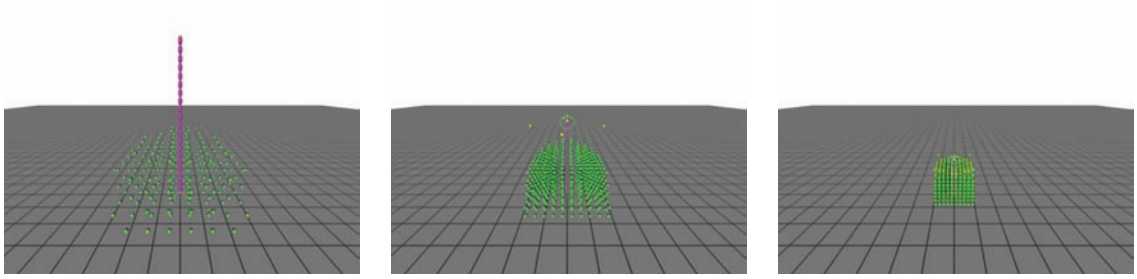

Figure 3.11: Reachable space for kinematic chains varying in size using the transpose method. The surrounding cube containing the goals to reach is scaled accordingly. The green balls indicate goals actually reached. **Left:** Two meters long kinematic chain. **Middle:** One meter long kinematic chain. **Right:** Fifty centimeters long kinematic chain.

kinematic chains (in topology, DoFs, joint limits...) generally lead to dissimilar results in term of iterations number.

It is difficult to find a general compromise between the *integration step* and the *damping factor*. On one hand, if the damping factor is too low, the process can not converge at all in case of singularity. On the other hand, if the damping factor is too high compared to the integration step, the convergence time is too prohibitive. However, using an integration step of one and a damping factor of ten works well for the majority of examples we have tested.

3.3.3 Damped Least Squares Inverse Versus Transpose

We use the *damped least squares* inverse of the Jacobian (DLS) to compute one iteration of the algorithm. Another technique is the *Jacobian transpose* method as discussed in Section 3.1. It is known to be fast to compute but it also has very poor convergence properties. The analysis of these two methods has provided us with results on the advantages and drawbacks of the DLS method.

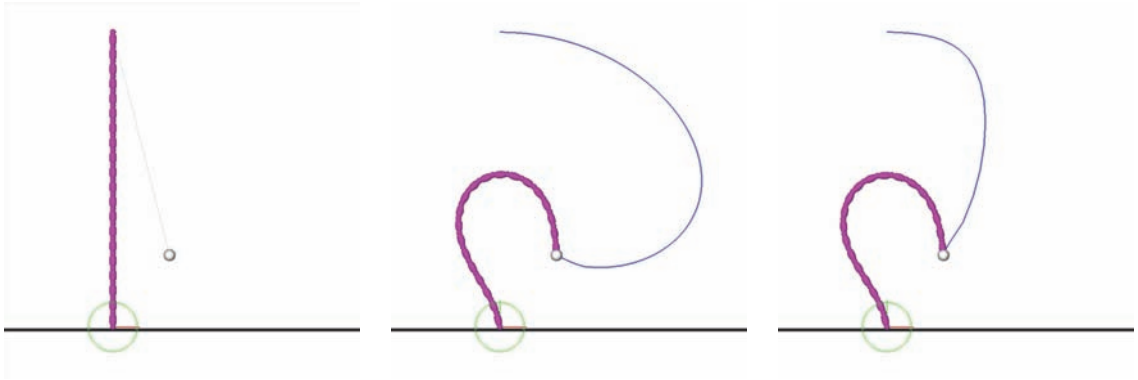

Figure 3.12: Paths followed by the end-effector depending on the IK method. **Left:** Initial configuration. **Middle:** Transpose of the Jacobian. **Right:** Damped least squares inverse of the Jacobian.

The test configuration is similar to the one used for the benchmarks in Section 3.3.2.2 (see Figure 3.8). Conversely to the DLS method which provides a solution of minimum norm, the transpose method may sometimes lead to diverging solutions. For example, when the joints are almost collinear, the final norm of the solution is too important, leading to instabilities of the kinematic chain. Figure 3.11 shows the goals for which the transpose method actually converges to a solution. We can notice that the length of the kinematic chain directly influences the number of reachable goals. Indeed, the longer the kinematic chain, the more important the task increment thus leading to instabilities. This could be overcome by multiplying the task by a gain factor. This factor may be computed depending on the length of the kinematic chain in order to reduce instabilities. This gain factor should depend on the current configuration as the actual end-effector displacement (obtained thanks to the previously computed joints increment solution) also directly depends on the current configuration [Craig, 1986].

Furthermore, Figure 3.12 shows the paths followed by the end-effector to converge toward an arbitrarily chosen goal (the white sphere). We can notice that the path followed by the end-effector for the DLS method is much more direct than the one using the transpose method.

Finally, as expected the transpose method has very poor convergence properties near the goal. As a result, although the transpose of the Jacobian is faster to compute than its DLS inverse, it needs much more steps to converge. In the example of Figure 3.12, the DLS method converges in 29 iterations and 8.2 ms (approximately 0.28 ms / iteration), while the transpose method converges in 82 iterations and 14.7 ms (approximately 0.18 ms / iteration), which is twice as long to achieve this simple task. Hence, while the transpose method is faster for computing one step, it takes much more iterations to converge toward a specific goal and as a result, it is less efficient than the DLS method.

3.4 Discussion and Conclusion

In this work we have first detailed, based on the work of Baerlocher [Baerlocher, 2001], a numerical resolution framework capable of solving multiple tasks simultaneously, and which supports both task-priority and weighting strategies for the resolution of conflicts.

We have then presented an *HAnim-compliant* application providing different kinds of features such as body parts position control, CoM position control, joint weighting, joint recruiting level, etc.

Prior techniques decompose complex three dimensional articulations such as the shoulder into three simple revolute joints. This parameterization is then equivalent to the Euler angles and suffers from several drawbacks: lack of intuitiveness and *gimbal-lock* singularity. We have shown that the exponential map parameterization is more appropriate than the previous one to represent complex articulations. While it is more complex than the Euler angles parameterization, its performances in terms of final computation time and iterations number are equivalent. Moreover, exponential maps provide an intuitive way to represent joint limits and is singularity-free for well-chosen initial configurations.

Thanks to this test-platform, we have performed benchmarks and comparisons on several key parameters of the IK solver. Among others, we have compared the mutual influence of the integration step and the damping factor. No general equation relating the integration step, the damping factor and the convergence behavior seems to be pertinent. Indeed, the convergence behavior is closely related to the considered hierarchy of joints and to the starting configuration. However, we have experienced that a damping factor of ten and an integration step of one work well for the vast majority of examples we have tested.

At this point, we have presented our IK solver and performed several tests to highlight its efficiency. This is particularly critical as we want to build a motion editing framework using a *per-frame IK plus filtering* scheme. Its efficiency is directly related to the underlying IK solver. So far, we have presented a way to design postures using our IK solver. However, our main interest is to edit motion capture animations using a constraint-based motion editing algorithm. The next chapter explicitly details the definition of a *constraint* in our framework. In particular, we present three classes of constraints dedicated to end-effector trajectory editing, footplant editing and balance control.

CHAPTER 4

Motion Deformation Constraints Definition and Design

Our main goal is to develop a constraint-based motion editing framework. That is, we want to modify an existing animation using *Motion Deformation Constraints* (for simplicity, they are called *constraints* in the remainder of this manuscript). These constraints are often explicitly created by the animator. As a result, any motion editing framework must provide the animator with a set of predefined tools to ease the constraint definition.

In the previous chapter, we have presented how to edit a virtual character's posture using tasks. It is important to note that tasks are quite different from constraints in our discussion: while tasks are used for designing postures, constraints are used to edit animations. They represent trajectories in space as well as in time.

In this chapter, we present three classes of constraints to help the animator editing animations. We first present a versatile class of constraints allowing the user to intuitively and precisely control the trajectory of end-effectors: the *shape-constraints*. Then, we present constraints entirely dedicated to footplants editing. Indeed, these constraints are often required when editing an animation. As a result, defining a specific class handling footplants editing is of great interest in our framework. Furthermore, Reitsma and Pollard [Reitsma and Pollard, 2003] and O'Sullivan et al. [O'Sullivan et al., 2003] have shown that even though a motion is not physically correct, it can still be believable if the deviation from physical correctness is not too important. Thus, to avoid producing animations containing unbalanced frames which lead to unpleasant results, we also introduce a constraint for the CoM position control so that the character seems to stay balanced during the entire animation.

4.1 State of the Art in *Constraints Formulation*

Energy Functions

As far as our knowledge goes, the most general formulation of constraints was introduced by Witkin et al. in [Witkin et al., 1987]. They presented a catalog of constraints expressed as energy functions. All the energy functions constraining a virtual character are first summed together: this results in a single energy function representing the whole set of individual constraints. This energy function is then optimized to find the solution which best satisfies the constraints. However, representing constraints as mathematical functions is not intuitive and sometimes difficult. For example, expressing constraints such as "grace" or "dance-like-Travolta" using a mathematical formulation is merely impossible. Moreover, this method suffers from all the problems due to optimization: occurrence of local minima, gradients sometimes complex to express, prohibitive computational cost, etc.

Physics-Based Constraints

Several previous works used physical laws to improve animations' realism. In [Popovic and Witkin, 1999], Newton's laws were applied on a simplified character to minimize computational costs. Rose et al. [Rose et al., 1996] generated realistic transitions between motions by constraining the motion to minimize the energy consumption. Liu and Popović [Liu and Popovic, 2002] used an optimization subject to constrains on the linear and angular momentum which directly depend on whether the character is on the ground or airborne. Some methods constrained the Zero Moment Point of the character to remain inside its support polygon to ensure that it was dynamically balanced [Tak et al., 2002][Dasgupta and Nakamura, 1999][Ko and Badler, 1996]. Enforcing the Zero Moment Point to remain inside its support polygon is an appealing constraint. However, it is well-defined for planar grounds but is difficult to generalize to uneven terrains [Sardain and Bessonnet, 2004]. Moreover, it leads to computationally expensive methods. In [Shin et al., 2003], the authors used a simplified formulation of the Zero Moment Point to achieve interactive rates. [Pollard and Reitsma, 2001] and [Yamane and Nakamura, 2003] used a dynamics filter to track a reference motion while enforcing dynamic constraints such as joint acceleration or contact forces. In [Zordan and Hodgins, 2002], maintained balance by constraining the CoM to remain as close as possible to the center of support, or by adjusting the desired reference motion by offsetting the joint angles of the legs similarly as in [Wooten and Hodgins, 2000]. Finally Pai and Patton demonstrated the relationship between velocity and position of the CoM and proposed a prediction model to decide whether the balance can be maintained or not [Pai and Patton, 1997].

Geometric Constraints

Geometric constraints are much more intuitive because they directly specify a goal for a specific body part: a point can be constrained to a specific position [Gleicher, 1997], [Gleicher, 1998], or can be constrained to move along a line [Witkin et al., 1987]. [Liu and

Popovic, 2002] and [Witkin and Popovic, 1995] used keyframes to specify motion features. Keyframes may be considered as a set of geometric constraints that determine the position and/or orientation of each joint.

Bindiganavale and Badler [Bindiganavale and Badler, 1998] introduced interaction constraints to map a motion from one character to another, each having a different morphology. Gleicher ([Gleicher, 1997], [Gleicher, 1998]) introduced a kind of interaction constraint: a point that should have the same motion as another point, a constant distance between two points and/or, a constant orientation between two points. [Witkin et al., 1987] used a parametric model and allows the specification of geometric constraints with interactions such as surface attachment or collision constraints.

4.2 The Shape-Constraints: a Versatile Representation of End-Effectors Trajectories

Constraint-based motion editing techniques require the user to specify the important features the final motion should achieve. Providing interactive tools to construct such constraints is a key point when designing motion editing methods. Systems failing to do so tend to be incomplete as it is not reasonable to ask the user to specify constraints in mathematical form.

In particular, constraints representing the trajectory of end-effectors are very useful. Unfortunately, it is not feasible to require the animator to precisely know the whole trajectory of a specific articulation or of a body part. Most of the time, he only knows a set of specific points the end-effector has to pass through. As a result, it is needed to find an efficient interpolation method so that given a set of points in space we can generate an entire trajectory for an end-effector.

A common way to represent trajectories in space is to use cubic interpolation splines such as Cardinal splines for example. However, this type of interpolation offers little flexibility as, by definition, the final spline is always smooth in space. As a result, it is not easy to have trajectories containing sharp corners when wished for. For example, consider the motion of the right wrist which first reaches a location, then stays stationary for a while and finally goes back to its initial position. In such a situation, we should be able to decide whether the desired end-effector's trajectory is smooth or not at locations where it stays stationary for a period of time.

Another important issue when editing an end-effector's trajectory is whether the original trajectory contains important features or not. In cases when the initial motion is entirely altered, the final end-effector's trajectory may be totally designed from scratch. On the other hand, when the original trajectory contains features the animator really wants to keep in the final animation, the final trajectory should be defined relatively to the initial one. For example, suppose the initial animation is a virtual character waving at someone. Suppose we only want to change the global location of the waving motion. Then, the animator only wants to smoothly add an offset to the initial motion instead of entirely redefining the waving motion from scratch.

In the next sections, we introduce a class of constraints representing end-effectors trajec-

tories that are continuous in space as well as in velocity and smooth *except at specific points*. A prior work similar to ours but applied to animation interpolation was presented in [Steketee and Badler, 1985]. Moreover, these constraints may be defined using different modes so that the user may decide whether he wants to take into account the initial end-effector's trajectory or not.

4.2.1 Overview

In this section we give an overview of the construction of a shape-constraint given an animator's specifications. We define a *shape-constraint* as a pair of curves $(\boldsymbol{S}(u), \boldsymbol{T}(t))$ with u the parameter of curve $\boldsymbol{S}$ and t the parameter of curve $\boldsymbol{T}$. $\boldsymbol{S}(u)$ defines the end-effector's *trajectory in space*. $T(t)$ is used to control the *timing* of $\boldsymbol{S}(u)$. The parameter t then represents the current time in the animation. These curves are automatically constructed given a set of *constraint points* specified by the animator.

Constraint Points

A constraint point for a shape-constraint is equivalent to a control point when dealing with splines. However, it also handle timing information to ensure that the end-effector passes through the specified locations at the requested times. More formally, a constraint point $P(P_{location}, P_{begin}, P_{end})$ is defined by:

1 - A three dimensional location $P_{location}$: this location is used to interpolate the final trajectory of the end-effector at each time. It is set by the animator in the world space to facilitate its manipulation. However, for flexibility purposes, the position of a constraint point is internally stored relatively to a reference point as a three dimensional displacement map (see Section 4.2.3 for further explanation). This three dimensional location is used as a control point for the trajectory curve $\boldsymbol{S}(u)$.

2 - A begin time P_{begin}: specifies the starting time at which the end-effector should reach $P_{location}$. This time is used as a control point for the time curve $\boldsymbol{T}(t)$.

3 - A end time P_{end}: specifies the ending time at which the end-effector should depart from $P_{location}$. This time is used as a control point for the time curve $\boldsymbol{T}(t)$.

The time interval $[P_{begin}, P_{end}]$ then defines a duration during which the end-effector should remain stationary at the specified location $P_{location}$. It is important to note that the constraint points are the only objects that the animator needs to explicitly specify to construct a shape-constraint.

Constructing a Shape-Constraint

For clarity purposes, we focus our discussion on shape-constraints containing a single constraint point only. However, as the method is straightforward, it is very easily extended to cases with several constraint points. The construction of a shape-constraint that holds

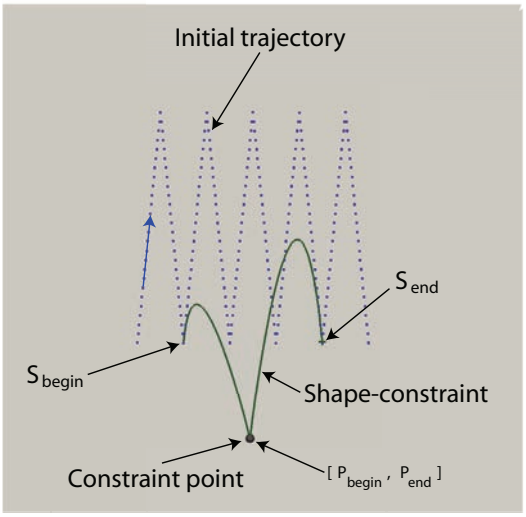

Figure 4.1: Example of a shape-constraint with a single constraint point.

over a period of time $[S_{begin}, S_{end}]$ and which contains a constraint point P defined by $(P_{location}, P_{begin}, P_{end})$ consists in the following steps (this example is conceptually illustrated in Figure 4.1):

1 - The first (resp. the last) control point of $\boldsymbol{S}(u)$ is created at the initial end-effector's position at time S_{begin} (resp S_{end})

2 - The tangent of the first (resp. the last) control point of $\boldsymbol{S}(u)$ is adjusted with respect to the velocity of the end-effector in the input motion.

3 - The location $P_{location}$ of the constraint point P is added to $\boldsymbol{S}(u)$ as a control point.

4 - If the end-effector must stay stationary for a period of time at location $P_{location}$ then $\boldsymbol{S}(u)$ is made sharp at the corresponding control point.

5 - Finally, $\boldsymbol{T}(t)$ is computed so that $\boldsymbol{S}(\boldsymbol{T}(t_\alpha)) = P_{location}$ for $P_{begin} \leq t_\alpha \leq P_{end}$.

The animator only needs to define the timing of the shape-constraint as well as the constraint points the shape-constraint has to pass through. The trajectory and time curves are then automatically computed.

4.2.2 Specification of a shape-constraint

In this section, we detail the construction of trajectory and time curves separately.

The Trajectory Curve

The trajectory $\boldsymbol{S}(u)$ of an end-effector is represented as a *Kochanek-Bartels* spline [Kochanek and Bartels, 1984]. This class of C^1 continuous interpolating cubic splines is based on Hermite interpolation basis functions. That is, each segment $[P_i, P_{i+1}]$ of the spline is defined as:

$$S(u) = P_i h_1(u) + P_{i+1} h_2(u) + D_i h_3(u) + D_{i+1} h_4(u) \tag{4.1}$$

where $0 \leq u \leq 1$, D_i (resp. D_{i+1}) is the tangent at control point P_i (resp. P_{i+1}) and the h_j are the *Hermite interpolation basis functions*.

In order to offer greater flexibility, each control point has two different tangents: the *source derivative* and the *destination derivative*. Each of these tangents is directly influenced by three parameters: *tension*, *continuity* and *bias*. These parameters are useful to explicitly change (or even break) the continuity of the curve at the control point. Given the tension t, the continuity c and the bias b, the source derivative DS_i and the destination derivative DD_i are computed as follows for a control point P_i (see [Kochanek and Bartels, 1984] for further details):

$$DS_i = \frac{(1-t)(1-c)(1+b)}{2}(P_i - P_{i-1}) + \frac{(1-t)(1+c)(1-b)}{2}(P_{i+1} - P_i) \tag{4.2}$$

and

$$DD_i = \frac{(1-t)(1+c)(1+b)}{2}(P_i - P_{i-1}) + \frac{(1-t)(1-c)(1-b)}{2}(P_{i+1} - P_i) \tag{4.3}$$

In addition, the first tangent (i.e. the destination derivative of the first control point) and the last one (i.e. the source derivative of the last control point) are set to the velocity of the end-effector on the initial trajectory. In this way, there is no discontinuity when going from the initial trajectory to the deformed one or when going back to the initial one.

Given this formulation, we are now able to add sharp corners to an end-effector's trajectory by setting the tension parameter of a specific control point to 1 when required. Reconsider the example of the wrist in which we need to adjust its position so that it reaches a stationary location over a period of time. In this case, there is no need for the trajectory to be smooth in space. The control parameters of the corresponding control point are then adjusted to add a sharp corner to the trajectory (the tension is set to one).

In addition, we need to reparameterize $\boldsymbol{S}(u)$ so that the interval of time $[P_{begin}, P_{end}]$ corresponds to the same value of parameter u. This reparameterization is handled by the time curve $T(t)$.

The Time Curve

The *time curve* $\boldsymbol{T}(t)$ is defined as an increasing piecewise linear function. This function establishes the correspondence between each time of the animation t and the parameter u of $\boldsymbol{S}(u)$. It is constructed as follows:

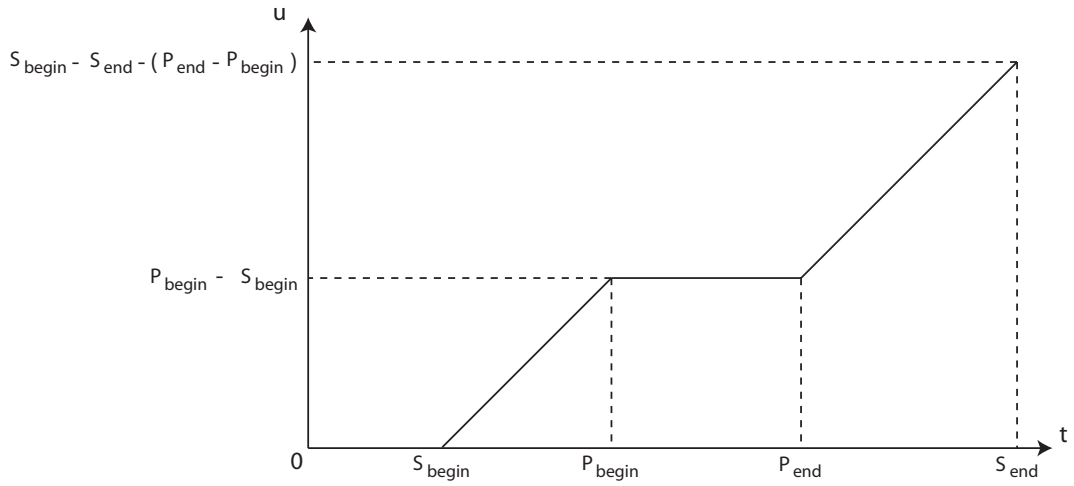

Figure 4.2: Example of a time curve corresponding to a shape-constraint starting at time S_{begin}, ending at time S_{end} and containing one constraint point defined as $(P_{location}, P_{begin}, P_{end})$.

- Each segment corresponding to a time interval between two constraint points is linear, with a slope of one, so that when the time in the animation is increased by Δt, it is increased by $\Delta u = \Delta t$ in the trajectory curve.

- Each segment corresponding to a time interval $[P_{begin}, P_{end}]$ is constant so that all the values in the time interval $[P_{begin}, P_{end}]$ correspond to the same value of u in $S(u)$.

Figure 4.2 shows an example of the time curve corresponding to the shape-constraint shown in Figure 4.1.

Finally, given a time t_α, the corresponding three-dimensional end-effector's position P in the shape-constraint is then defined as:

$$P = \boldsymbol{S}(\boldsymbol{T}(t_\alpha)) \tag{4.4}$$

4.2.3 Shape-Constraints Modes

When editing the trajectory of an end-effector, it is often desirable to be able to choose whether we want to take the initial trajectory into account or not. In other words, sometimes we need to define an *absolute* trajectory in space and sometimes we prefer to add a *relative displacement* in order to keep the final trajectory of the end-effector as close to the initial one as possible. To do so, we extend the concept of displacement map [Witkin and Popovic, 1995] [Bruderlin and Williams, 1995] to the three-dimensional case. Each control point of the trajectory curve is expressed in a reference frame (not necessarily in the end-effector frame) as seen in Figure 4.3. The representation of the trajectory curve then differs depending on the mode of the shape-constraint.

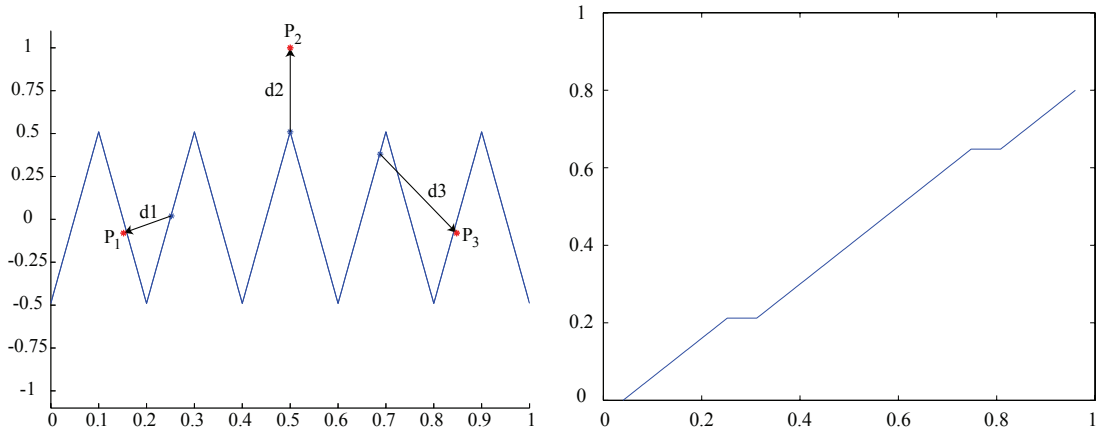

Figure 4.3: For clarity, we consider the reference as being the original motion. The constraint points P_1, P_2 and P_3 are stored as the displacements d_1, d_2 and d_3 with respect to the reference. P_1 and P_3 last for 15 frames. The animation is composed of 250 frames (10 seconds). The final shape-constraint is only active between frames 10 and 240. **Left:** Original trajectory in space. The blue dots indicate the initial positions of the end-effector while the red dots indicate the final desired positions. **Right:** Associated time curve using normalized time.

Absolute Mode

To construct the trajectory curve, we first add the origin of the reference frame to each control point of $\boldsymbol{S}(u)$. The trajectory curve is then the Kochanek-Bartels spline constructed from these control points: it represents the final trajectory of the end-effector. Figure 4.4 shows the initial trajectory curve as well as the final trajectory of the end-effector. This shape-constraint's mode is useful whenever the animator needs to entirely reshape the trajectory of an end-effector.

Relative Mode

In this mode, the trajectory is the Kochanek-Bartels spline constructed from the control points of $\boldsymbol{S}(u)$. The trajectory is then added to the reference trajectory to obtain the final one. This results in a trajectory that is smoothly deformed while retaining the global shape of the initial one as much as possible. Figure 4.5 shows the trajectory curve and the associated final trajectory. This shape-constraint's mode may be used in cases when the animator only wants to add an offset to the initial trajectory. For example, consider two virtual characters shaking hands. With this mode, it is possible to offset the global location of the right hands of the character while preserving the important features of the initial motion (the shaking itself).

Relative with Condensation Mode

We use the same method as for the relative mode. However, as the reference trajectory is moving while the end-effector should stay stationary, we readjust the trajectory curve so that the final trajectory pauses at required points even if the reference one is moving. This

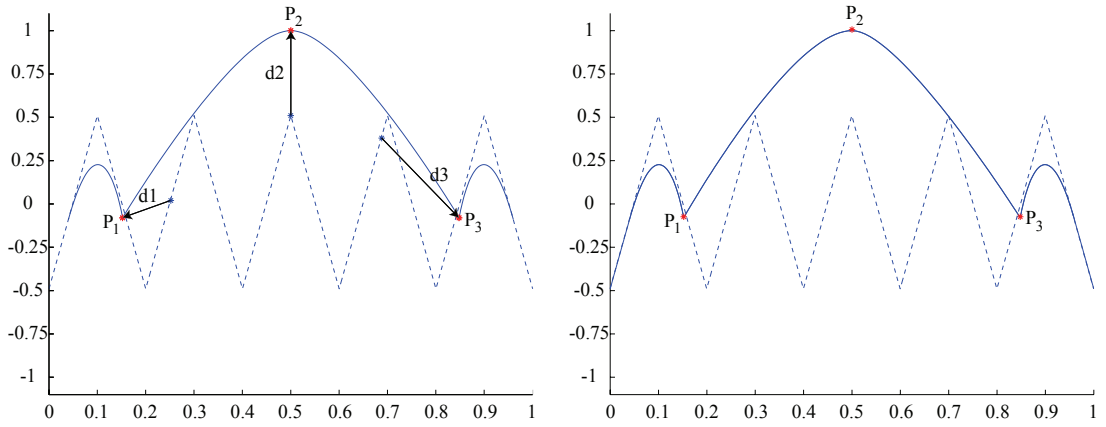

Figure 4.4: The dashed line represents the reference trajectory. **Left:** Trajectory curve in absolute mode. **Right:** Final trajectory.

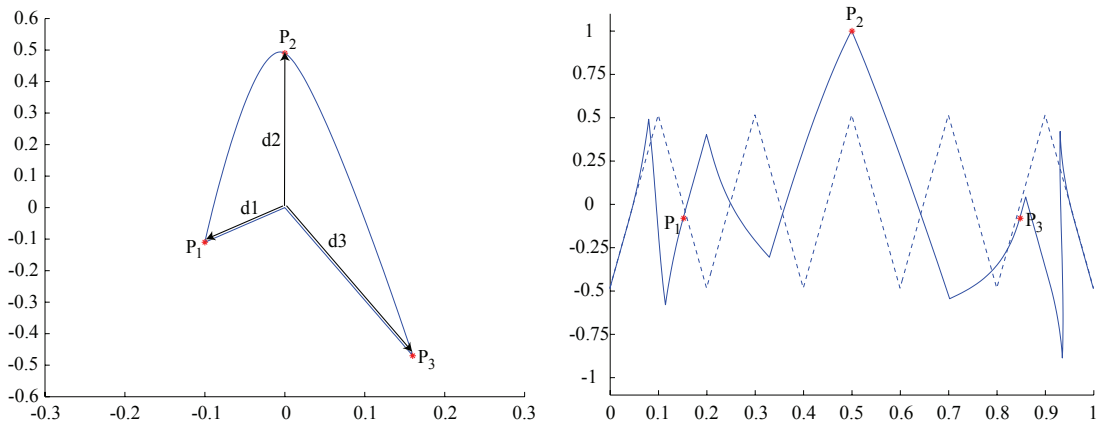

Figure 4.5: The dashed line represents the reference trajectory. **Left:** Trajectory curve in relative mode. **Right:** Final trajectory.

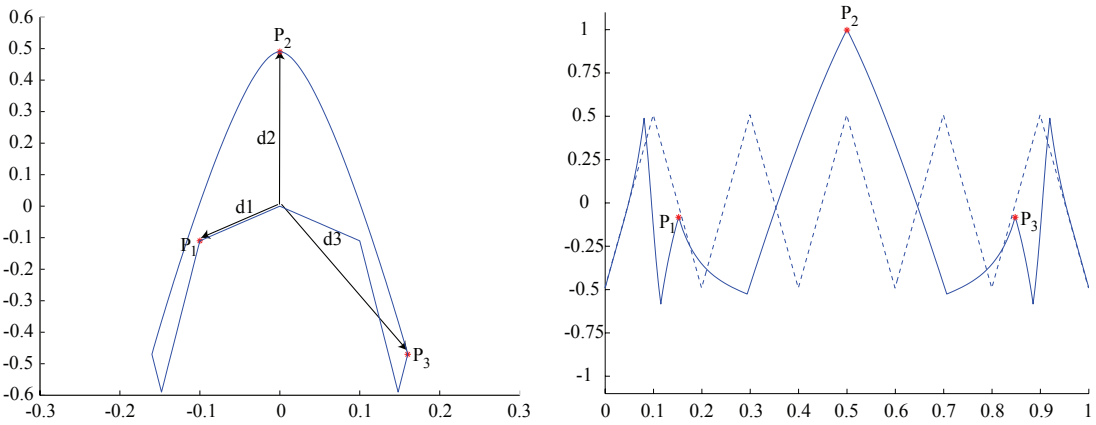

Figure 4.6: The dashed line represents the reference trajectory. **Left:** Trajectory curve in relative with condensation mode. **Right:** Final trajectory.

corresponds to subtracting the displacement in the reference trajectory for periods of time for which the end-effector is required to stay stationary. Figure 4.6 shows the trajectory curve and the associated final trajectory. It is important to note that for this specific mode, the time curve is set to the identity as we handle the fact that the end-effector has to stay stationary directly with the trajectory curve. This class is used in particular to correct artifacts such as foot sliding (see Section 4.3). Indeed, in this case, we want to adjust the location of the foot so that it stays stationary for predefined periods of time while it is moving in the original animation. These adjustments are gradually introduced to the initial motion to retain as many of its original characteristics as possible.

4.2.4 Shape-Constraints Examples

Figure 4.7 shows several examples of shape-constraints in different modes. In particular, note the sharp corner of the absolute shape-constraint when the middle constraint-point lasts for a period of time (Top right of Figure 4.7).

4.3 Footplant Constraints

A particularly important feature that any motion editing framework should offer is an intuitive and efficient way to edit footplants. Indeed, motion capture data often (if not always) contains artifacts. A particularly noticeable artifact is when a foot is moving while it should stay planted on the ground [Kovar et al., 2002b]. Hence, correcting all these *footskates* is a key point when producing high-end animations. It is therefore also of interest to be able to change the position and/or the orientation of footplants so that they correctly interact with the surrounding environment.

The shape-constraints previously presented allow the animator to efficiently and intuitively edit the trajectory of end-effectors. However, a footplant is much more complicated

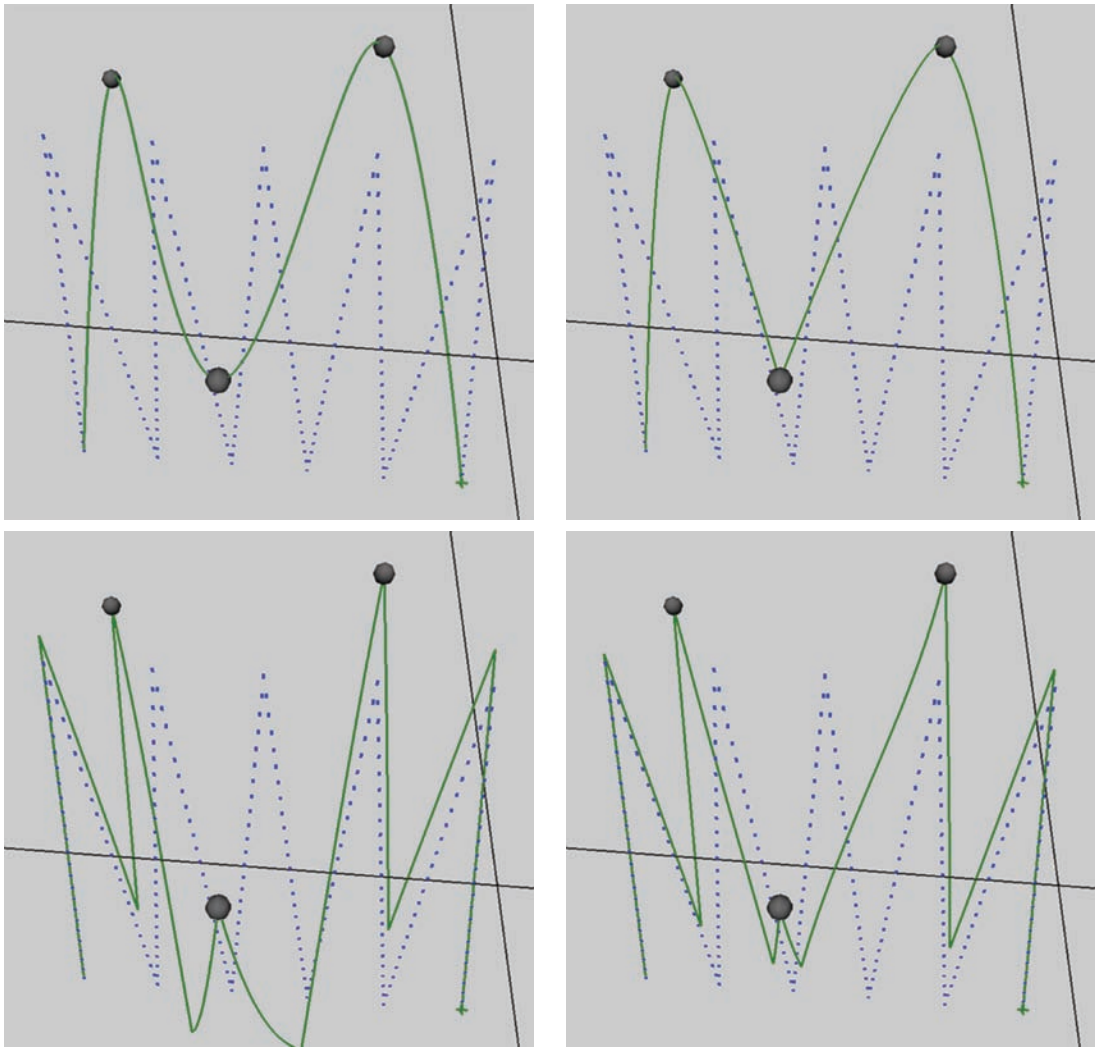

Figure 4.7: Examples of shape-constraints with different modes. **Blue dots:** Initial trajectory. **Green curve:** Generated shape-constraint. **Top left:** Absolute shape-constraint. **Top right:** Absolute shape-constraint with the middle constraint-point lasting for 40 frames. **Bottom left:** Relative shape-constraint. **Bottom right:** Relative shape-constraint with condensation.

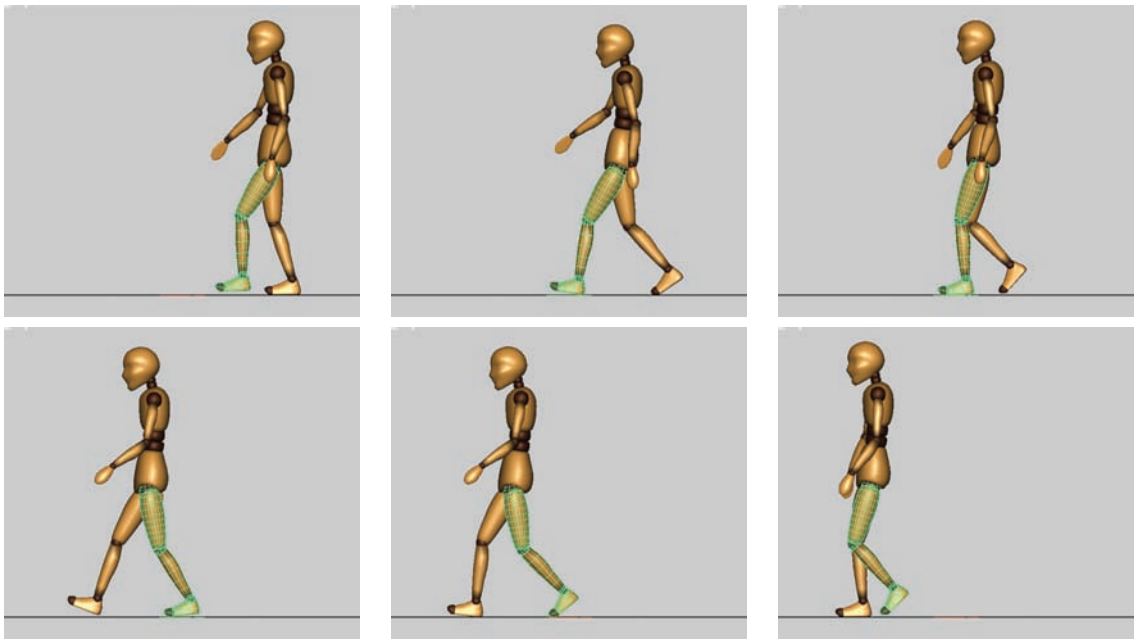

Figure 4.8: The four main events defining a footplant. **Top left:** Before the footplant. **Top middle:** Heel strike event. **Top right:** Toe strike event. **Bottom left:** Heel off event. **Bottom middle:** Toe off event. **Bottom right:** After the footplant.

than a single trajectory. Indeed, it often operates on several parts of the body at the same time. For example, the heel's and toe's positions are very important when defining a footplant. Moreover, a footplant is decomposed in a sequence of important events that must occur at specific times. As a result, footplants editing requires a complex management of end-effectors position and timing. We thus found it useful to define a specific class of high-level constraints dedicated to footplants adjustments.

4.3.1 Specification of a footplant constraint

A footplant is defined using four important events:

1. **Heel Strike (HS):** instant when the heel touches the ground
2. **Heel Off (HO):** instant when the heel leaves the ground
3. **Toe Strike (TS):** instant when the toe touches the ground
4. **Toe Off (TO):** instant when the toe leaves the ground

For example, if we consider a classical walking animation, the sequence of events is $[HS, TS, HO, TO]$ as shown in Figure 4.8.

Footplant constraints are then defined using the following low-level constraints:

1. One shape-constraint SC_{heel} controlling the heel position. This constraint controls the events *HS* and *HO* using a constraint point CP_{heel} which lasts from *HS* until *HO*.

2. One shape-constraint SC_{toe} to control the toe position. This constraint controls the events *TS* and *TO* using a constraint point CP_{toe} which lasts from *TS* until *TO*.

As the foot is considered as a rigid body, the distance between the toe and the heel must remain constant. This directly implies that the distance between both shape-constraints defining the footplant must be constant as well. However, this is hardly manageable because at creation time, we only know the position of the heel and the toe:

- at the beginning of the footplant adjustment (the beginning of the shape-constraints),
- during the period of time constraining both heel and toe (corresponding to the time interval [TS,HO]),
- at the end of the footplant adjustment (the end of the shape-constraints).

As a consequence, the constraints on the heel and the toe may not be achievable simultaneously during some period of time. We then must decide which trajectory (between the heel and the toe ones) is the most important one.

4.3.2 Dynamic Priority Swap

Statically choosing which of the heel or the toe trajectories is the most important is not acceptable. Indeed, as shown in Figure 4.9, if the heel trajectory is assigned a higher priority, the toe penetrates the floor during interval of time [HO,TO]. Conversely, if the toe trajectory is considered as the most important one, the heel penetrates the floor during interval of time [HS,TS].

We make the assumption that constraints handling contact with other objects in the scene, and particularly with the ground, must be more important than those handling position "in the air". For example, consider two points p_1 and p_2, p_1 being on the surface of an object and p_2 being in space without any collision with any object in the scene. In this configuration, if we slightly move p_1 and p_2 (for example introducing an error of 1mm), it is visually more noticeable that p_1 is not at its correct position than p_2. As a consequence, the heel constraint should be assigned a higher priority while being on the ground (that is period of time [HS,HO]). Similarly, the toe constraint should be assigned a higher priority during period of time [TS,TO]. However, this is not possible because intervals [HS,HO] and [TS,TO] overlap during period [TS,HO] (see Figure 4.10). However, in this case, both constraints are achievable as it corresponds to the situation in which both the heel and the toe are on the ground: the priority is then not as important and may be "arbitrarily" chosen without affecting the results. As a consequence, we must assign a higher priority to the heel constraint during period [HS,TS]. Similarly, we must assign a higher priority to the toe constraint during period [HO,TO]. We then have to dynamically swap the priorities during time interval [TS,HO]. We arbitrarily chose to swap the priorities at time HO. Figure 4.10 schematically summarizes the dynamic priority swap concept. It is important to note that

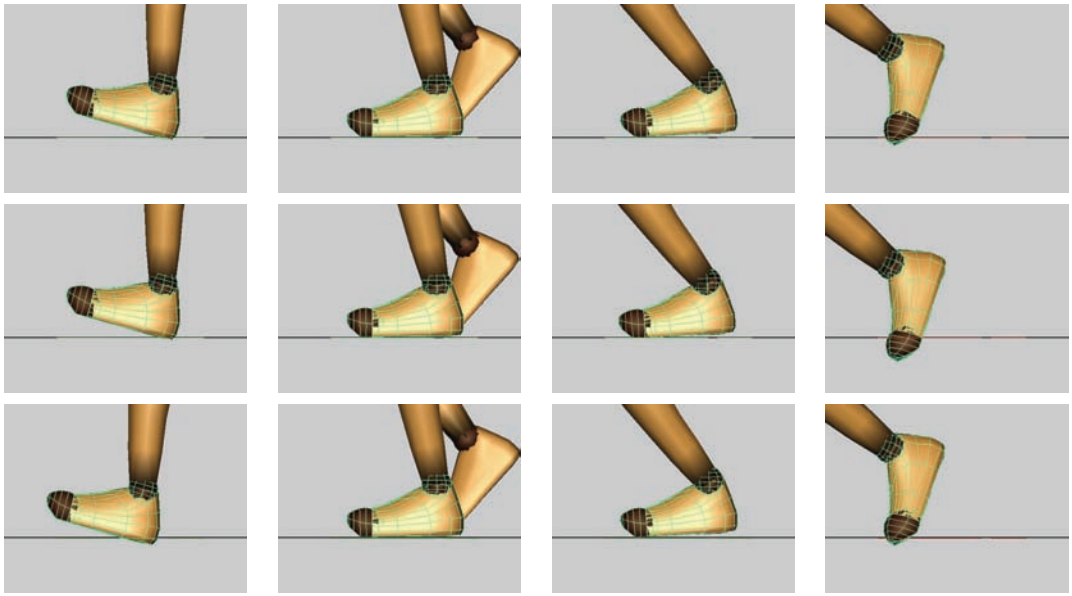

Figure 4.9: Comparison of footplant enforcement with and without dynamically swapping the priorities between the heel and the toe. From left to right: heel strike, toe strike, heel off and toe off. **Top row:** Footplant adjustment: we dynamically swap the priorities between the heel and the ankle. **Middle row:** Footplant adjustment: the ankle has a higher priority level. The position of the toe at event TO is clearly inaccurate: the foot penetrates the ground. **Bottom row:** Footplant adjustment: the toe has a higher priority level. The position of the ankle at event HS is clearly inaccurate: the foot penetrates the ground.

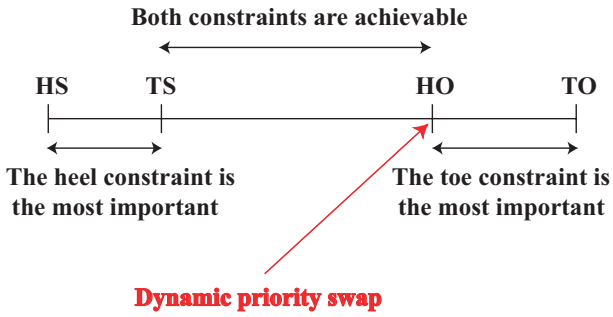

Figure 4.10: Dynamically swapping the priorities between the heel and the toe constraints.

as both constraints are achievable when we swap the priorities (and are actually achieved), no discontinuity occurs. However, in the general case, if we swap the priorities between two constraints, the final result may contains discontinuities if one of the constraints is not achieved.

The purpose of the footplant constraints is not to modify the path of the motion but to locally adjust the animation since large adjustments may lead to large rotations of the pelvis and the root joint when the legs are fully extended. A solution could be to identify such situations and avoid too important rotational components by stretching the legs as in [Kovar et al., 2002a]. However, such a solution is not acceptable because changing the size of the limbs may be problematic when using the animation in other applications. Instead, we rely on our IK solver by reducing, when needed, the recruiting level of the footplant constraints so that the pelvis and the root joint do not participate to the achievement of the solution anymore. Furthermore, as our motion editing algorithm aims at adjusting the location of footplants, it would probably fail when the target positions are too far from their original locations.

4.4 Balance Control

In our framework, we choose to control the CoM position through inverse kinetics [Boulic et al., 1996]. Unbalanced postures are then adjusted to improve realism or to apply some additional effects. The final CoM position $\boldsymbol{P}'_{CoM}$ first needs to be estimated. For this, we consider the initial CoM's position $\boldsymbol{P}_{CoM}$ in the input motion (and its corresponding position $\boldsymbol{P}'_{CoM}$ in the output motion) as well as a set A of n_a points on the body in the input motion (and the corresponding set of points A' in the output motion). A may be different from one motion to another, and as a result, A' may be different as well. $\boldsymbol{P}'_{CoM}$ is then defined using the relation:

$$\boldsymbol{P}'_{CoM} - \frac{\sum_{i=1}^{n_a} \boldsymbol{A}'_i}{n_a}, A' = \boldsymbol{P}_{CoM} - \frac{\sum_{i=1}^{n_a} \boldsymbol{A}_i}{n_a}, A \tag{4.5}$$

where $\boldsymbol{A}_i$ (resp. $\boldsymbol{A}'_i$) is the position of the i^{th} point of A in the input motion (resp. output motion). $\sum_{i=1}^{n_a} \boldsymbol{A}_i/n_a$ is the position of the barycenter of A in the input motion and $\sum_{i=1}^{n_a} \boldsymbol{A}'_i/n_a$ is its corresponding position in the output motion. For example, when the animator changes the position of the footplants, A being composed of the heel and toe joints of each leg, the

final CoM position takes into account the final location of the feet to compute the goal for the CoM (see Section 5 for examples on footplants editing).

Figure 4.11 shows the initial trajectories of a character's CoM and the barycenter of A for a simple walking animation. In this example, A is composed of the heel and toe joints of each leg. It is interesting to note the high correlation between the two trajectories from cycle to cycle, as highlighted by the superposition of the red curves (figure 4.11 Right).

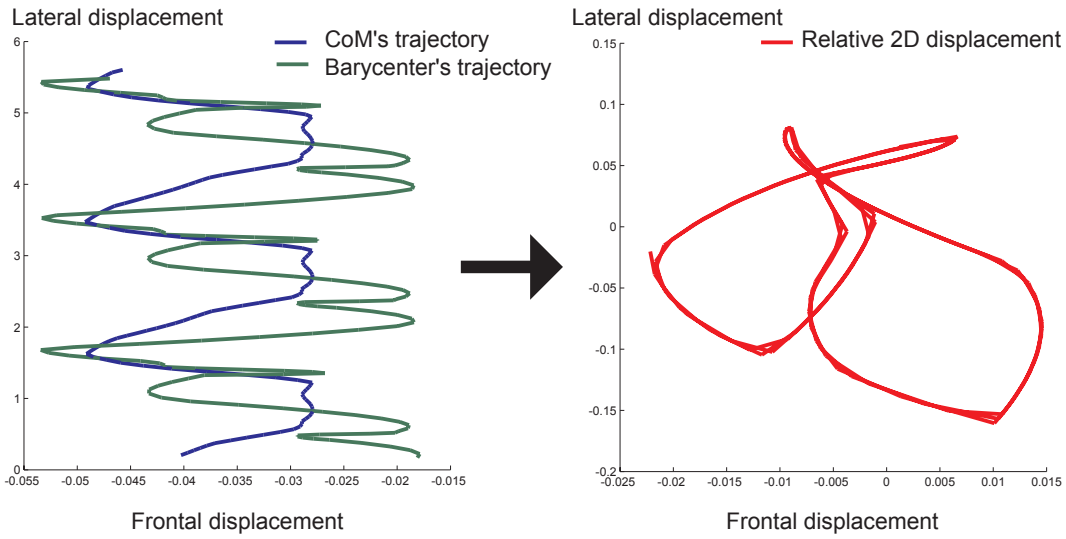

Figure 4.11: Left: Input trajectories of the CoM and the barycenter of A for a walking motion **Right:** Relative 2D displacement between the CoM and the barycenter of A. The units are different on both axes to highlight the details.

A *3D displacement map* may be added to the resulting curve to produce new behaviors. Notice that the choice of A is crucial as it directly influences the deformed motion. However, in most cases, it remains intuitive: for example, for classical walking or running motions, A should contain points of the feet.

While this method cannot ensure that the character is dynamically balanced, it has the advantage of treating the balance constraint as any other constraint in our framework. Moreover, as the deformed motion is usually close enough to the initial one, the adjustments are small, and constraining the CoM position to its original trajectory often provides good results. However, for highly dynamic motions, or when adding drastic changes to the initial animation, physical laws should be taken into account.

4.5 Conclusion

In this chapter, we have presented important constraints integrated in our motion editing framework. These constraints may then be used to modify an initial animation so that it fits the animator's requirements.

In particular, we have introduced a versatile class of constraints: the *shape-constraints*. The animator is now able to design a wide range of spatial trajectories. Moreover, these con-

straints may contain stationary points represented by sharp corners in the trajectory. Furthermore, these constraints can be expressed in a reference frame allowing relative constraints between joints, to shift a joint position or to define an absolute trajectory.

As discussed before, footplants are a recurring problem when dealing with raw motion capture data. We have then presented a class of constraints dedicated to the adjustment of the position and/or orientation of footplants. Additionally, we have presented a simple approach to adjust unbalanced postures by controlling the position of the CoM thanks to inverse kinetics. This approach offers the animator the possibility to improve the overall quality of the final animation.

In the next chapter, we detail our algorithm for computing the final animation achieving, as much as possible, the constraints specified by the animator.

CHAPTER 5

Prioritized Motion Deformation

Producing high-quality character animations is still a research area of interest. One popular technique to achieve this is *motion capture*, as it is able to create realistic animations in a short period of time. Unfortunately, one of its major advantages is also its major drawback: the animation is recorded by mimicking the motion of a performer. Thus, the final animation must be planned before the capture is done and is only valid for virtual humans having the same proportions as the live performer. For this reason, these animations are not directly reusable and need additional adaptations.

Recently, motion databases have become commercially available. Given a motion database, it now becomes a new challenge to create the animations we need for each virtual human we want to animate. Graph-based motion synthesis [Arikan and Forsyth, 2002][Kovar et al., 2002a][Lee et al., 2002] and blending techniques [Kovar and Gleicher, 2003][Park et al., 2002][Perlin, 1995] consider the database as a whole to construct new motions. The resulting animations are built from finely crafted combinations of the input data. These techniques are well-suited for video games for example. Conversely, motion editing techniques adapt a single animation to fit specific needs [Menache, 1999].

Our method aims at adding significant deformations to an input motion, while retaining as many of its characteristics as possible. The initial animation is deformed using prioritized constraints. The key feature of this technique is that prioritized constraints are sorted into priority-layers. Constraints belonging to the highest priority-layer are enforced first. Then, those of the next priority-layer are satisfied as much as possible without disturbing the previous ones, and so on. It is important to note the difference between weighted and prioritized constraints (see figure 3.1 for an analogy with conflicting tasks in an IK context). When con-

flicts arise between constraints, the weighting strategy leads to a compromise where none of the constraints are met while the priority strategy ensures that the *important constraints* are achieved and *less important one* are satisfied *as much as possible*.

Motion deformation constraints presented in chapter 4 are integrated in this framework. Additionally, motion deformation constraints to control end-effector orientation and/or to attract the results toward the input motion have been included. As a consequence, the animator is able to deform an animation in a very flexible way, with an arbitrary number of priority-layers and constraints.

In our framework, each frame is individually deformed with an IK solver so that a set of predefined constraints is satisfied. Being a per-frame approach, a filtering process is used to smooth the results when needed. This algorithm can be repeated, within an interactive design loop, to reenforce important constraints.

Per-frame IK plus filtering methods have already been presented in chapter 2. The next section examines them in more detail. Section 5.2 describes the motion deformation algorithm. In Section 5.3, we tackle issues related to the continuity of the final motion. In Section 5.4 we analyze the convergence properties of our algorithm. In Section 5.5, we demonstrate how our method can be applied to deform a wide range of motions. Finally, we discuss its limitations and we conclude in Section 5.6.

5.1 State of the Art in *per-frame IK plus filtering*

In this section we detail major per-frame IK plus filtering works. In particular, we highlight their important advantages and drawbacks for future comparisons with our framework.

The Coach-Trainee Metaphor

Boulic and Thalmann [Boulic and Thalmann, 1992] presented the first example of a motion editing using IK on a per-frame basis to enforce constraints. This paper clearly sets the foundations for future work using this approach. The authors used a numerical IK solver in order to simultaneously handle two tasks: the primary and the secondary tasks. The primary task is responsible for kinematics constraints. These latter are defined using half-space constraints in order for the user to easily define basic needs such as feet not penetrating the floor. The primary task is called the *trainee*. The secondary task is in charge of tracking the original motion so that the corrected motion does not deviate too much from the initial one. This task is called the *coach*. This motion editing algorithm has been applied to correct movements generated by a walking engine.

Hierarchical Motion Editing

Lee and Shin [Lee and Shin, 1999] introduced the first example of per-frame IK plus filtering. Their interactive motion editing system addresses many common tasks such as retargetting, transitions and/or editing. They introduced two key concepts of per-frame IK plus filter-

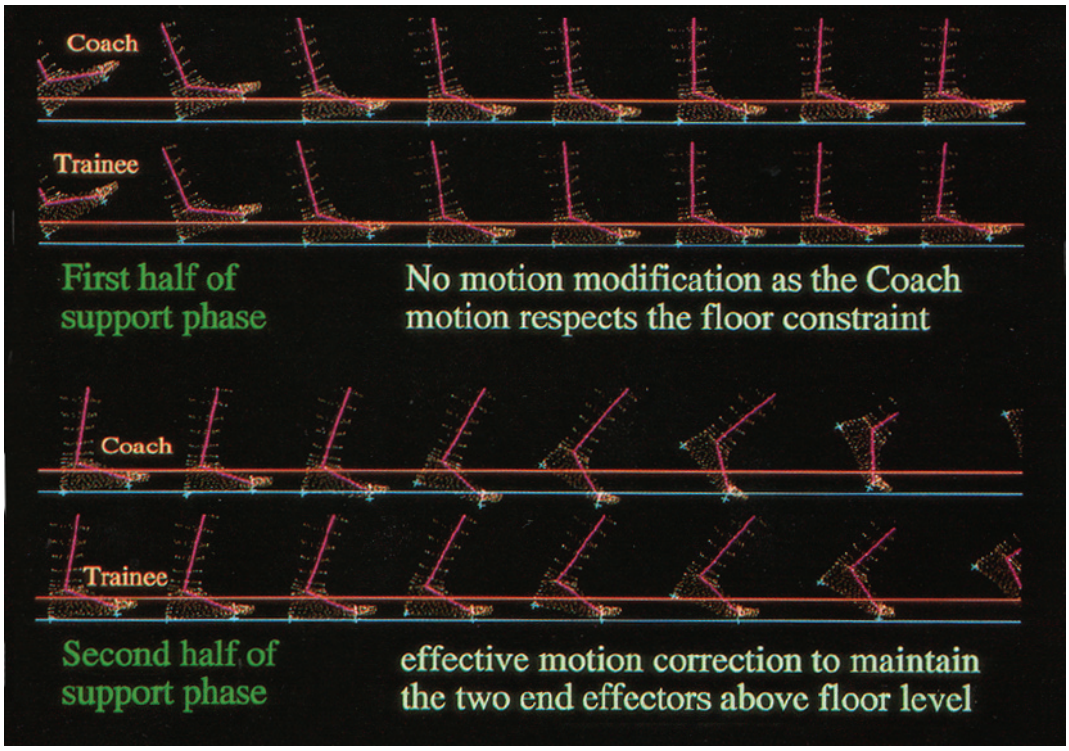

Figure 5.1: Correction of a walking motion (source: [Boulic and Thalmann, 1992]).

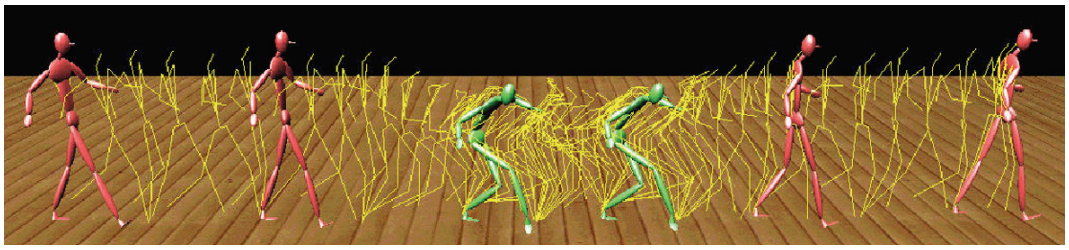

Figure 5.2: Transitions between walking and sneaking (source: [Lee and Shin, 1999]).

ing methods: the *intra-frame* and the *inter-frame* consistencies. The intra-frame consistency represents the set of spatial constraints a virtual human must achieve at each frame of the animation. This is usually done by using an IK solver. The inter-frame consistency specifies that neighboring adjusted frames have to be as similar as possible to avoid adding jerkiness in the final animation. This is usually done by filtering the *changes* to the initial motion in order not to add high frequencies. The inevitable consequence is that it potentially destroys intra-frame consistency. As a result, these phases are usually iteratively repeated. Their system makes specific choices for each key aspect of the approach: the IK solver and the filtering process.

They implemented a highly specialized IK solver for human-like articulated figures (see Section 3.1.4 for more details). The final implementation is very fast. The authors have had to make several sacrifices in order to achieve such performances. By construction, their IK solver can handle human-like articulated figures only. Secondly, the set of constraints that

it can handle is limited. Finally, it is not practical if the final postures significantly deviate from the input ones.

The authors enforced inter-frame consistency using a hierarchical B-spline-based filter. Each displacement map is adaptively refined by hierarchically fitting B-splines defined by uniform sequences of knots. The more knots we use, the finer the results. The main problem of using B-spline-based filtering is that it becomes computationally expensive if the number of knots in the B-splines is large. Indeed, the problem of fitting a B-spline to scattered data points is reduced to finding the set of control points which best interpolates the data points. This is done by minimizing an objective function using a least-squares method.

Online Motion Editing

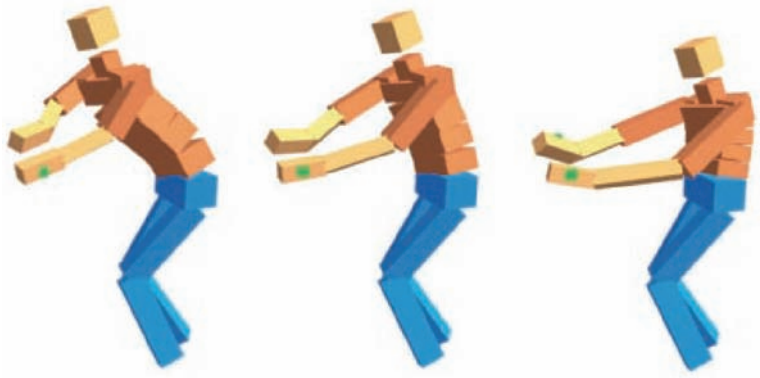

Figure 5.3: Snapshots taken from the retargetted motion (source: [Choi and Ko, 2000]).

In specific contexts (video games, real-time performance, etc), motion editing systems can only rely on past and present frames: future frames are a-priori unknown. They then have to process each frame and display them on the fly. Such applications may be found in games for example where the animations must react to unpredictable events. These systems are inherently per-frame methods as they cannot take the future into account and cannot change the past as it has already been displayed.

Choi and Ko presented in [Choi and Ko, 2000] the first work on online motion retargetting. Their IK solver is based on the motion rate control method. The primary task is to track the motion of the end-effectors. The secondary task is to imitate the motion of the source character as much as possible. As a consequence, the inter-frame consistency is implicitly enforced.

A similar technique was presented by Shin et al. in [Shin et al., 2001]. As in the work of Lee and Shin [Lee and Shin, 1999], their IK solver is specialized in handling human-like articulated figures (see Section 3.1.4 for more detail). The novelty of their approach is that each end-effector is given an importance that may dynamically change in order to decide which of the position of end-effectors and the posture of the limbs must be preserved. Indeed, if the characters do not have the same size and/or proportions, they generally cannot be simultaneously achieved. The importance value is directly related to the distance between end-effectors and surrounding objects in the scene. Finally, the online motion filtering stage

relies on a Kalman filter because it is capable to predict and to adjust the results. The major limitation of this technique is the concept of importance that is similar to a weighting strategy. Moreover, the filtering process uses Kalman filters. Whilst these are online filters, that is, they do not need any information about future frames, they have been designed to be applied to linear systems which is not the case when considering human motions, which usually contain periods of severe nonlinearity. Hence, it may happen that the filter totally loses track of the initial signal and produces undesired results.

Figure 5.4: Morphology-independent motion adaptation (source: [Kulpa et al., 2005]).

Finally, Kulpa et al. [Kulpa et al., 2005] proposed an efficient implementation of the Cyclic Coordinate Descent algorithm to adapt an animation in real-time. While Shin et al. [Shin et al., 2001] used a simplified skeleton, Kulpa et al. proposed a methodology to represent motion in order for the related data as constraints to be independent from the underlying hierarchy instead. Hence, important constraints such as foot-contacts are preserved while scaling the skeleton. Whilst this method does not use any kind of filtering, it provides very good results for real-time motion adaptation of virtual characters. To ease the adaptation, the underlying skeleton is divided into groups, each of which being an individual kinematic chain. This has the advantage to ease the computation of a solution. However, as no synergy exists between groups, it may lead to unrealistic results. Moreover, they proposed a control scheme to constrain the CoM to its original position. This method works well in cases where the feet are located at the same place in the initial and the adjusted motions. However, if we need to change the position of the feet, this algorithm leads to unbalanced postures.

Motion Retargetting

Monzani et al. proposed in [Monzani et al., 2000] a method to retarget animations to characters having both geometrical and topological differences using an intermediate skeleton. This latter serves as a bridge between all the potential skeletons we need to map the animation to. This method uses a numerical IK solver similar to ours. Hence, using the optimization vector reduces the need to filter the results as each adjusted posture is attracted to its initial configuration. To achieve smooth transitions between original end-effectors trajectories and constraints trajectories, the authors decided to add an ease-in period (resp. an ease-out

Figure 5.5: Motion conversion to various characters. The original motion is captured on the red skeleton on the left (source: [Monzani et al., 2000]).

period) before (resp. after) the constraints are activated (resp. deactivated). During the ease-in period, the end-effector trajectory is linearly interpolated to smoothly reach the specified goal. During the ease-out period, they preferred to linearly interpolate the postures instead of the trajectories because of potential conflicts between constraints leading to discontinuities when one of them is deactivated. According to the authors, this method only gives good results when the adapted motion remains close to the original one. In particular, linear interpolation may produce noticeable discontinuities because it does not take end-effectors velocity into account.

Footskate Cleanup

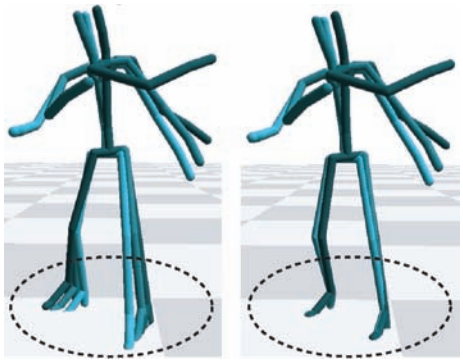

Figure 5.6: Original versus cleaned motions (source: [Kovar et al., 2002b]).

Kovar et al. proposed in [Kovar et al., 2002b] a method dedicated to the problem of footsliding removal. Their algorithm uses analytical methods only as the root and the spine orientation are not adjusted in their framework. This method is basically divided into three steps. First the root position is computed. Then, each limb posture is independently adjusted so that each end-effector reaches its goal. During this stage, each limb may be additionally stretched to avoid sharp changes. Finally, the resulting displacement maps are smoothly propagated to surrounding frames to avoid discontinuities. While this method works well

for the specific problem of footsliding removal, it lacks in genericity and may not work for general motion editing problems. Furthermore, the IK solver they used does not allow user-defined end-effectors specification.

In the next sections we present our motion deformation framework.

5.2 Per-Frame Inverse Kinematics

Our approach is similar to the one presented by Choi et al. in [Choi and Ko, 2000]. We use the numerical IK solver presented in chapter 3 in order to define prioritized constraints while deforming an animation. Moreover, the iterative construction of the solution channels the convergence through intermediate solutions, enforcing the highest prioritized constraints first. In this way, if the iterations number of the IK solver is not large enough to ensure that all the prioritized constraints are met, it at least ensures that the most important ones are achieved.

One important issue while deforming animation on a per-frame basis is the choice of a starting configuration. Several works directly rely on the associated posture in the input motion [Lee and Shin, 1999][Shin et al., 2001][Kovar et al., 2002b]. This configuration choice is only acceptable for methods where the final animation stays close to the initial one. Indeed, as the IK solver starts from the original posture, neighboring deformed postures may be quite dissimilar and the low-pass filtering step inevitably leads to unpleasant results.

In [Choi and Ko, 2000] and [Monzani et al., 2000], each configuration is entirely dependent on the previous deformed one for the IK solver. Similarly, we also make the assumption that each character's posture in the output motion is similar to the previous one. However, joints which are not recruited by constraints should exactly reproduce their original motion. Hence, in our framework we additionally make the distinction between joints participating to the deformation and the others.

Let $\boldsymbol{m_{initial}}(t) = (\boldsymbol{p_r}(t), \boldsymbol{q_1}(t), \cdots, \boldsymbol{q_{n_{joints}}}(t))$ be the initial motion. Furthermore, let $\boldsymbol{m_{deformed}}(t) = (\boldsymbol{p'_r}(t), \boldsymbol{q'_1}(t), \cdots, \boldsymbol{q'_{n_{joints}}}(t))$ be the deformed one. For a given instant of time t_α, the starting state of the deformed posture $\boldsymbol{m_{deformed}}(t_\alpha)$ is then defined as:

$$\boldsymbol{m_{deformed}}(t_\alpha) = (\boldsymbol{p^*_r}(t_\alpha), \boldsymbol{q^*_1}(t_\alpha), \cdots, \boldsymbol{q^*_{n_{joints}}}(t_\alpha)) \oplus \boldsymbol{d}(t_\alpha) \tag{5.1}$$

where (the same holds for $\boldsymbol{p^*_r}(t_\alpha)$):

$$\boldsymbol{q^*_i}(t_\alpha) = \begin{cases} \boldsymbol{q'_i}(t_\alpha - \Delta t) & \text{if the joint is controlled} \\ \boldsymbol{q_i}(t_\alpha) & \text{otherwise} \end{cases}$$

for $1 \leq i \leq n_{joints}$. Δt is the time interval between two consecutive frames. Each adjusted posture is then attracted toward its corresponding one in the input motion, thanks to the optimization vector described in chapter 3.

5.3 Enforcing Continuity

A commonly accepted assumption follows from the observation that high frequencies in motion are important as they generally carry much of the naturalness of an animation [Witkin and Popovic, 1995][Gleicher, 1998][Gleicher, 2001]. Hence, high frequencies must not be disturbed and should be added or removed with care.

High frequencies may be principally added during the IK step (because each frame is independently adjusted) and because of the constraints activation/deactivation.

5.3.1 Filtering the Adjustments

Adjusting each frame individually may violate the inter-frame consistency. Thus, to ensure that our approach effectively produces natural looking motions, we need to low-pass filter the deformation we want to add to the original animation. Doing so, we ensure that no high-frequencies are added (or removed) to the original motion. During the IK step, each joint is attracted toward its original value using the optimization vector described in Chapter 3. Since surrounding postures in the original motion are close to each other, we then ensure that we minimize the deviation from the original motion as much as possible. Moreover, as we are working in an offline framework, we have a total control over the iterations number of the IK solver. Hence, providing that the iterations number is large enough, we also ensure that the residual errors due to the lower priority constraints, and in particular the optimization vector, are minimized as much as possible. As a result, we also limit the risks of adding discontinuities to the final animation. In cases where the iterations number is too low however, there is no guarantee regarding the continuity of the results.

In the vast majority of motions we have edited, attracting the deformed motion toward the original one was sufficient to avoid adding discontinuities. Hence, in these cases, no filtering was used. However, it may happen that the original motion is particular enough that any deformation would generate artifacts in the final motion. For example, when a leg is nearly fully extended, a small adjustment to the toe position induces important changes to the articulations of the leg. In such cases, we need to low-pass filter the added deformation. We use a Finite Impulse Response (FIR) filter (a convolution) to perform this task. If the discontinuity is too important, this filtering process significantly alters important constraints and we need one more pass.

It may also happen that the original motion contains noise such as jerkiness in the end-effectors position. Enforcing position constraints on these end-effectors then produces artifacts since we need to add high frequencies to the original animation to compensate for this erratic motion. A solution may be to stretch the skeleton as in [Kovar et al., 2002b]. However, as previously explained in section 4.3, this solution is not acceptable in our framework because changing the size of the limbs may be problematic when using the animation in other applications. We instead prefer to filter the results and to allow the end-effectors to not strictly achieve their task in order to avoid adding artifacts.

Furthermore, the CoM position control is also very useful to avoid the occurrence of discontinuities in the translation components of the root of the hierarchy. Indeed, as these latter are not considered while attracting the deformed posture toward the original one, the

IK solver tends to translate the root as much as possible to limitate the deviation of the remaining joints from the initial posture. Hence, while adding realism to the final motion, the CoM control also improves the robustness of our approach.

Finally, it is important to emphasize that as we use a numerical IK solver and thus can attract the solution toward the original animation, discontinuities only occur in very marginal cases.

5.3.2 Constraints Activation/Deactivation

Another issue when dealing with continuity is the activation/deactivation of constraints. In our framework, the position constraints are not considered as locations in space but as continuous trajectories instead. Hence, to enforce a position constraint at a specific time t_α, we define a shape-constraint between times $t_\alpha - \Delta t_{c_b}$ and $t_\alpha + \Delta t_{c_e}$ where Δt_{c_b} is the duration to go from the original trajectory to the goal location and Δt_{c_e} the duration to go from the goal location back to the original trajectory.

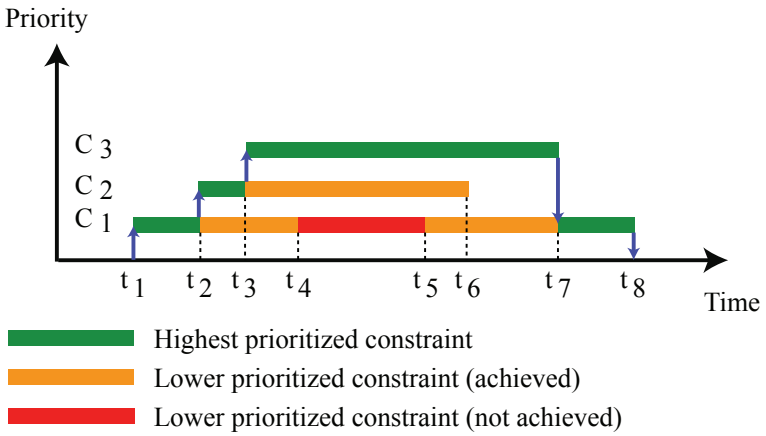

Figure 5.7: General scheme of conflicting prioritized constraints. The constraint C_1 is activated a time t1. When C_2 is activated, C_1 is still achieved. At time t3, all the constraints are met. At time t4, C_1 and C_3 conflict. C_3 has a higher priority and is achieved while C_1 minimizes the residual error. Finally, C_1 is achieved at time t5 before C_3 is deactivated.

Moreover, the animator must take special care when dealing with conflicting constraints. Figure 5.7 shows an example of conflicting prioritized constraints. We must ensure that when a constraint is deactivated, the conflicting constraints immediately below it in the hierarchy of priorities are achieved. Failing to do so may lead to discontinuities.

For example, consider the example of a 2D point whose trajectory is along the x axis (see figure 5.8). To edit its motion, we use two shape-constraints C_1 and C_2, C_1 having a lower priority than C_2. In this particular configuration, these constraints conflict. C_1 is active during frames 30 to 60 while C_2 is active during frames 10 to 40. During the period of activity of C_2, C_1 has no impact on the final trajectory. When C_2 is deactivated, C_1 is not achieved. Thus the deformed trajectory "jumps" onto C_1 producing a discontinuity in the trajectory of the 2D point.

Algorithm 1 Time and space consistency enforcement between potentially conflicting constraints

STEP 1: SORT POTENTIALLY CONFLICTING CONSTRAINTS

$PCCSETS \leftarrow \emptyset$ **/* Potentially conflicting constraints sets */**
$CONSTRAINTSLIST \leftarrow$ the list of constraints defined by the user
$alreadyOverlapped \leftarrow false$
$overlappedSet \leftarrow \emptyset$
for all constraint C in $CONSTRAINTSLIST$ **do**
  $alreadyOverlapped \leftarrow false$
  for all set S in $PCCSETS$ **do**
    if $\neg alreadyOverlapped$ **then**
      if $activityOverlapping(C, S)$ **then**
        $S \leftarrow S + C$
        $overlappedSet = S$
        $alreadyOverlapped \leftarrow true$
      end if
    else
      if $activityOverlapping(C, S)$ **then**
        $overlappedSet \leftarrow overlappedSet + S$
        $remove\ S$
      end if
    end if
  end for
  if $\neg alreadyOverlapped$ **then**
    $newSet \leftarrow C$
    $PCCSETS \leftarrow newSet$
  end if
end for

STEP 2: ENFORCE TIME AND SPACE CONSISTENCY IN EACH SET

for all set S in $PCCSETS$ **do**
  $activeConstraints \leftarrow \emptyset$
  for all constraint C in S **do**
    for all constraint $activeC$ in $activeConstraint$ **do**
      if C and $activeC$ control the same end-effector **then**
        Change the constraint with lower priority so that it connects to the higher priority one in time and space. If both have the same priority, then identically modify both to connect them in space and time.
      end if
    end for
    $activeConstraints \leftarrow activeConstraints + C$
  end for
end for

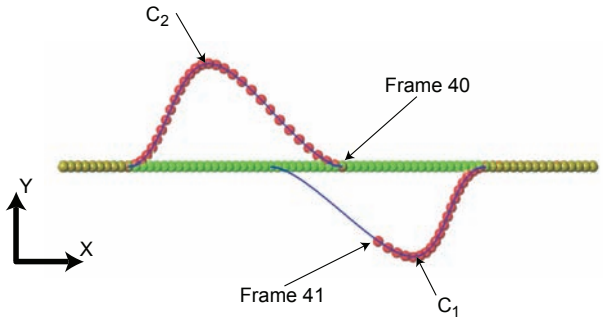

Figure 5.8: Configuration leading to discontinuities: the shape-constraint C_1 has a lower priority than the shape-constraint C_2. **Green points:** Original animation. **Red points:** Deformed animation. **Yellow points:** Superposition of original and deformed animations. **Blue curves:** The specified shape-constraints. Note the discontinuity between frames 40 and 41.

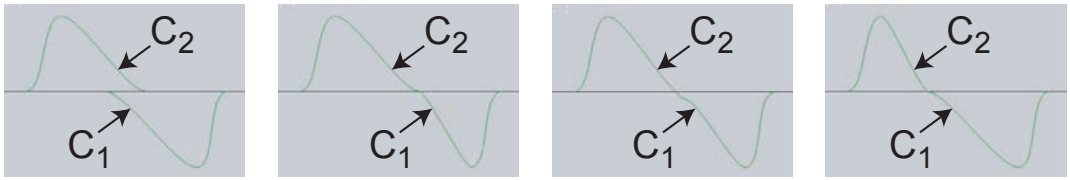

Figure 5.9: Time and space consistency enforcement. **Left:** Initial configuration. **Middle left:** C_1 has a lower priority than C_2. **Middle right:** C_1 and C_2 have the same priority. **Right:** C_1 has a higher priority than C_2.

In our framework, we solve this problem by enforcing a time and space consistency between potentially conflicting constraints controlling the same end-effector. We assume that two constraints are potentially conflicting if their periods of activity overlap in time. All the constraints are first sorted in sets containing all the constraints having overlapping activity period. Hence, a set contains all the constraints that may directly or indirectly disturb the achievement of the others in the same set. Then, the activity period of constraints controlling the same end-effector as well as their position in space are adjusted so that they are connected in time and space. In this way, we avoid the problem of overlapping activity periods shown in Figure 5.8 as well as gaps between consecutive constraints controlling the same end-effector. Our method is detailed in Algorithm 1. Figure 5.9 shows some results of the time and space enforcement. Note that the final curve is different depending on the priority of the constraints.

5.4 Convergence and Stopping Criteria

Analyzing the convergence of our motion editing algorithm for an entire animation is similar to analyzing its convergence for a single frame. Furthermore, we exclude from our discussion configurations which diverge by construction. For example, as detailed in section 3.3.1.3, if the rule stated by Equation 3.18 is not met, the IK solver cannot converge, and as

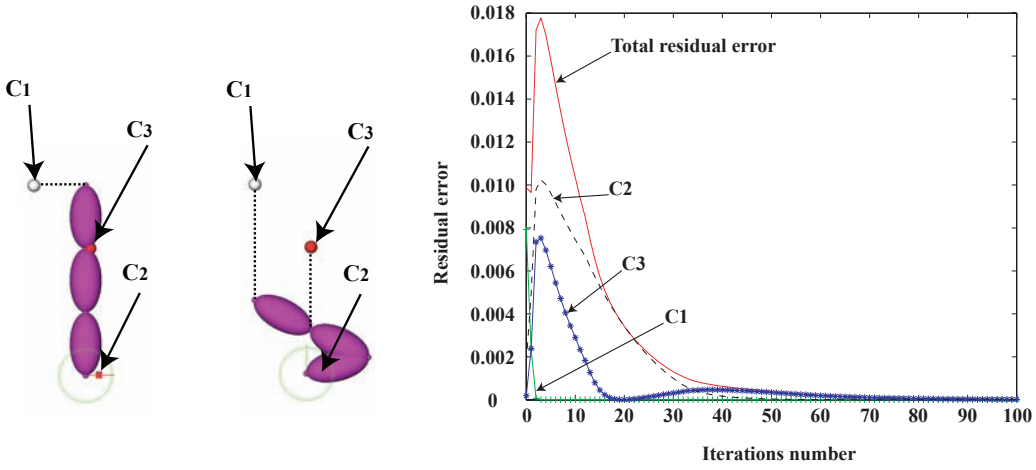

Figure 5.10: Convergence of the algorithm for three conflicting constraints C_1, C_2 and C_3 where priority C_1 > priority C_2 > priority C_3. **Left:** Initial configuration. **Middle:** Final configuration. **Right:** Residual error for each constraint. Note that higher priority constraints may disturb the convergence of lower priority ones.

a consequence, our motion editing algorithm cannot either.

We choose a test configuration exhibiting convergence interaction between conflicting constraints (see left of Figure 5.10).

The kinematic chain contains 3 DoFs and is controlled by three constraints:

1. C_1 controls the position of the chain's tip. It has the highest priority. This constraint is only concerned by vertical alignment of the end-effector and the goal. Hence, the end-effector is free to move along the vertical line passing through the goal (the white ball).

2. C_2 controls the position of the CoM of the kinematic chain. It has a middle priority. As for C_1, C_2 is only concerned by vertical alignment of the chain's CoM and the goal (the red cube).

3. C_3 controls the position of the joint just under the chain's tip. It has the lowest priority. As for C_1 and C_2, C_3 is only concerned by vertical alignment the end-effector and the goal (the red ball).

We choose this case as it may be considered as the worst one for several reasons. First of all, the number of DoFs is much lower than in the case of a virtual human. Hence, high priority constraints tend to limitate the remaining space for lower priority ones to converge. As a result, the influence of high priority constraints onto lower priority ones is very important as these latter are not able to compensate for the disturbance induced by the former. Secondly, the number of constraints controlling the same kinematic chain is large. Finally, the constraints have opposite goals: C_1 greatly disturbs C_2 and C_3 as the direction to achieve their respective goals are almost opposite.

The residual errors are shown in the right of Figure 5.10. As expected, while minimizing ϵ_{C_1} (the residual error related to C_1), the algorithm increases ϵ_{C_2} and ϵ_{C_3} (the residual errors related to C_2 and C_3). Once C_1 is achieved, ϵ_{C_2} then strictly decreases. Even though C_2 and C_3 are conflicting, minimizing ϵ_{C_2} also minimizes ϵ_{C_3} except between frames 20 and 40 approximately. Finally, when C_2 has reached its goal, ϵ_{C_3} also strictly decreases.

The construction of the solution ensures that once a constraint has reached its goal, it cannot be disturbed by lower priority ones. Then, in the worst case, the algorithm first minimizes the residual error related to the constraint of highest priority. Once this is done, it minimizes the residual error related to the constraint just below in the hierarchy of priorities and so on. We can then conclude that the algorithm always converges as long as we do not consider diverging configurations which cannot be satisfied by construction.

Now that we know our algorithm converges, we must decide when it has converged *enough*, that is we must choose a stopping criterion. We could choose a static threshold for the total residual error beneath which we consider that the algorithm has converged. However, in some cases, one constraint (or more) is not achievable. Hence, our algorithm would not stop at all. To overcome this problem, we use a more common stopping criterion which is the variation of the total residual error.

Additionally, we use a static threshold to limitate the maximum iterations number. This may be useful for the animators who wants to obtain a quick and coarse estimation of the final animation. As shown in Figure 5.10, if the algorithm is stopped before it has time to converge, it at least ensures that high priority constraints are achieved since the solution is constructed so that it channels the convergence through intermediate solutions which enforce the highest prioritized constraints first. As a consequence, using priorities to specify the importance of constraints is suitable to build a coarse to fine architecture: we first give the user a rough estimation of the final animation and hierarchically refine the solution to achieve more aesthetic results.

5.5 Experimental Results

The final system is integrated into Alias™/Maya5 as plug-ins and MEL scripts. Figure 5.11 shows a screenshot of the application. For this particular example, it took less than one minute to create the required constraints. Indeed, the end-user only needs to specify the timing of each shape-constraint as well as the position and timing of the points it has to pass through. Moreover, assigning a priority to each constraint is straightforward as the only point which matters is the relative priority between them. For example, a set of constraints C_1, C_2 and C_3 with priorities 2, 10 and 15 gives the same results as with priorities 1, 2 and 3. Finally, though it has not been used to generate the results presented in this section, the user can assign a weight to each constraint. Thus, when two or more constraints with the same priority-level conflict, the conflict is solved using the weighting strategy.

We used our motion deformation algorithm to create a wide range of animations. All the animations are generated on an IBM T40p (Pentium M 1.6 GHz, 1Go RAM, ATI mobility FireGL 9000).

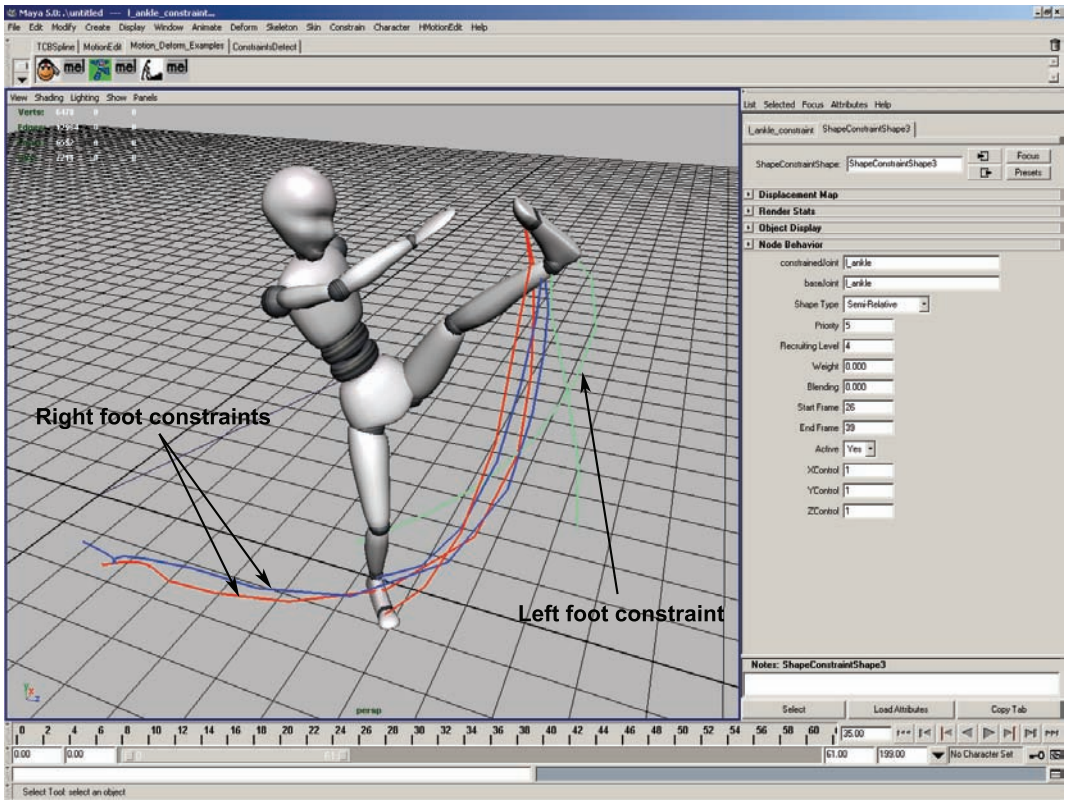

Figure 5.11: Motion deformation system integrated into Alias^{TM}Maya® 5

Karate Motion

In this example, the right foot is constrained to follow its original trajectory (high-priority). The left ankle of the character is constrained to reach a higher location (low-priority). Finally, the CoM is controlled to enforce balance (middle-priority). Once again, the priority level assigned to each constraint allows to decide whether to keep the left foot planted, or to constrain the right foot to reach its goal at all costs. In figure 5.12 we demonstrate that controlling the CoM position as well as assigning priorities to constraints may facilitate the process of motion deformation. In figure 5.13, we constrained the right ankle to reach goals at different heights. All these examples were generated in less than 2 seconds with visualization.

Editing Footplants

The initial animation contains 200 frames (8 seconds) and 8 footplants which we modified. Additionally, the CoM is controlled to enforce balance (low-priority). To obtain the catwalk, we also used a rotational constraint and a shape-constraint on the torso. The deformed animations shown in figure 5.14 were generated in approximately 40 seconds without visualization. Once again, it is important to emphasize that the purpose of footplant constraints in our framework is not to edit the whole path of the motion but to locally correct artifacts such as footslidings instead.

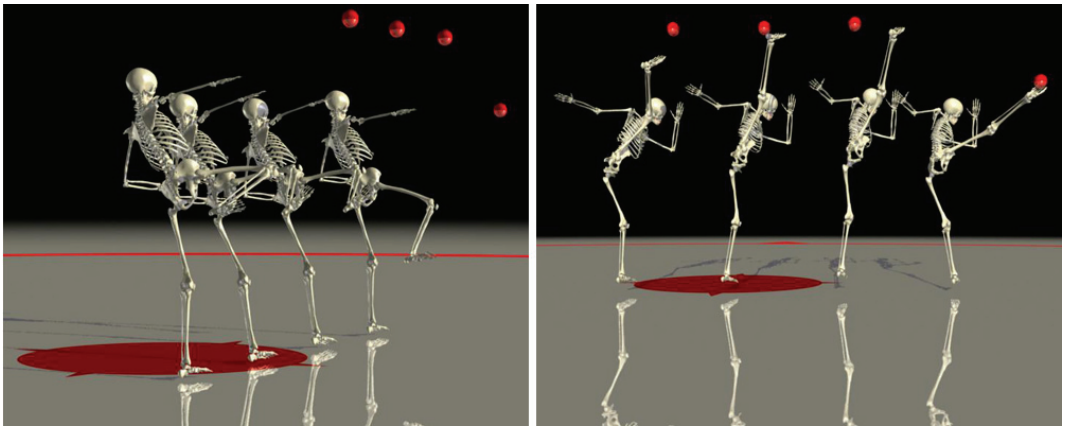

Figure 5.12: **Left:** Deformed motion. The ankle's goal cannot be reached without disturbing higher priority constraints. **Middle left:** The CoM is not controlled anymore resulting to unbalanced postures. **Middle right:** Resulting motion using weighting constraints. The location of the right foot is disturbed. **Right:** Original motion

Golf Swing

In this example, the initial animation contains 66 frames (3 seconds approximately). The toes have been first corrected so that they do not slide on the ground anymore. The the left hand motion (the one to which the golf club is attached) has then been modified in order to amplify the overall swing. Finally, the right hand is constrained to follow the trajectory of the left one. Additionally, the CoM is controlled to enforce balance. The deformed animation shown in figure 5.15 was generated in approximately 1 second without visualization.

Putting it all together

Finally, we applied our method to a walking animation to obtain a "climbing stairs" motion. We used different classes of constraints. We used four shape-constraints to design the "walking on stairs pattern" and one shape-constraint to constrain the relative position of the right elbow with respect to the torso. The orientation of the right arm was also constrained in order for the character to hold a tray horizontally. Finally, the CoM of the character was equally constrained. The final result is shown in figure 5.16. The whole animation needed two passes as the filtering process significantly disturbed the footplant constraints. Finally, the whole animation was generated in approximately 40 seconds without visualization.

5.6 Discussion and Conclusion

In this chapter we have presented an interactive method for adding significant changes to an animation. However, some improvements may be added to enhance its robustness.

The actual filtering process is simple but suffers from one drawback: our method becomes inherently off-line. We could improve our method by using on-line filters such as in

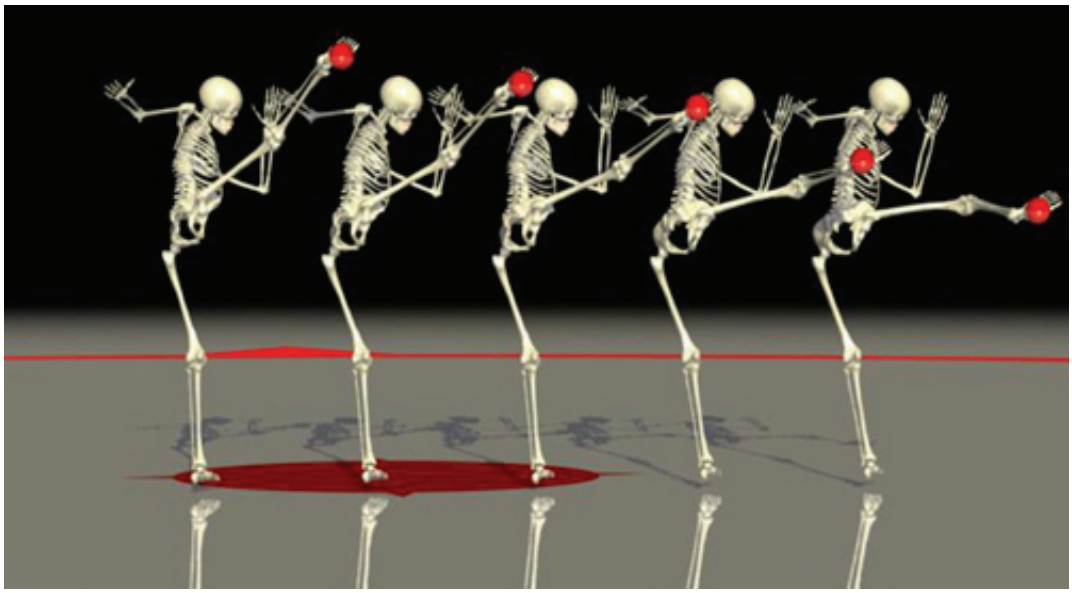

Figure 5.13: Reaching goals at different heights.

[Tak et al., 2002] and [Shin et al., 2001].

The main characteristics of such filters are the support (the size of the convolution) as well as the weights assigned to each value. However, it was not possible to choose generic values because the results had to be filtered in very marginal situations only.

As we directly rely on the input motion to estimate the CoM's position in the resulting animation, it becomes difficult to use in cases where the original motion is too noisy. We could improve the robustness of our approach by first cleaning the input motions (filtering and enforcing important constraints) using commercial tools.

We have presented in Section 5.3 an algorithm to avoid discontinuities due to of conflicting constraints controlling the same end-effector. However, discontinuous trajectories may be generated in various contexts. For example, it may be due to the overlapping kinematic

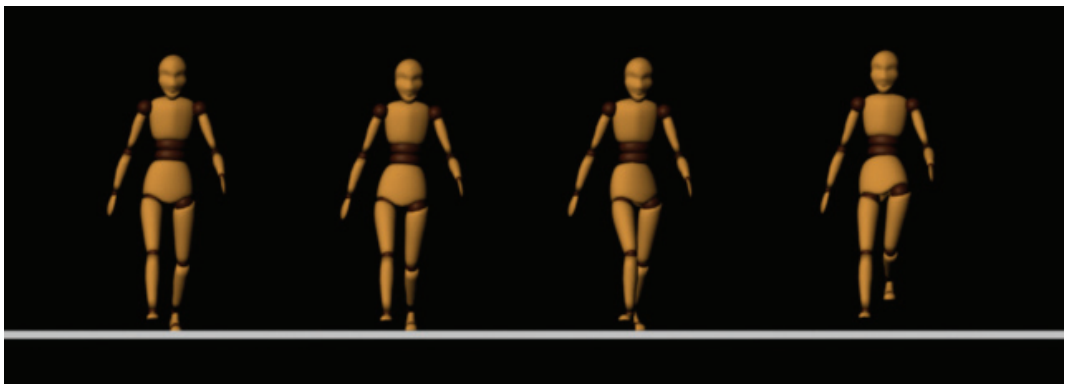

Figure 5.14: **Left:** Original motion. **Middle left:** Footplants are enforced. **Middle right:** The position and orientation of the original footplants are modified to obtain a catwalk. **Right:** The heights of the footplants are modified for the character to walk on steps

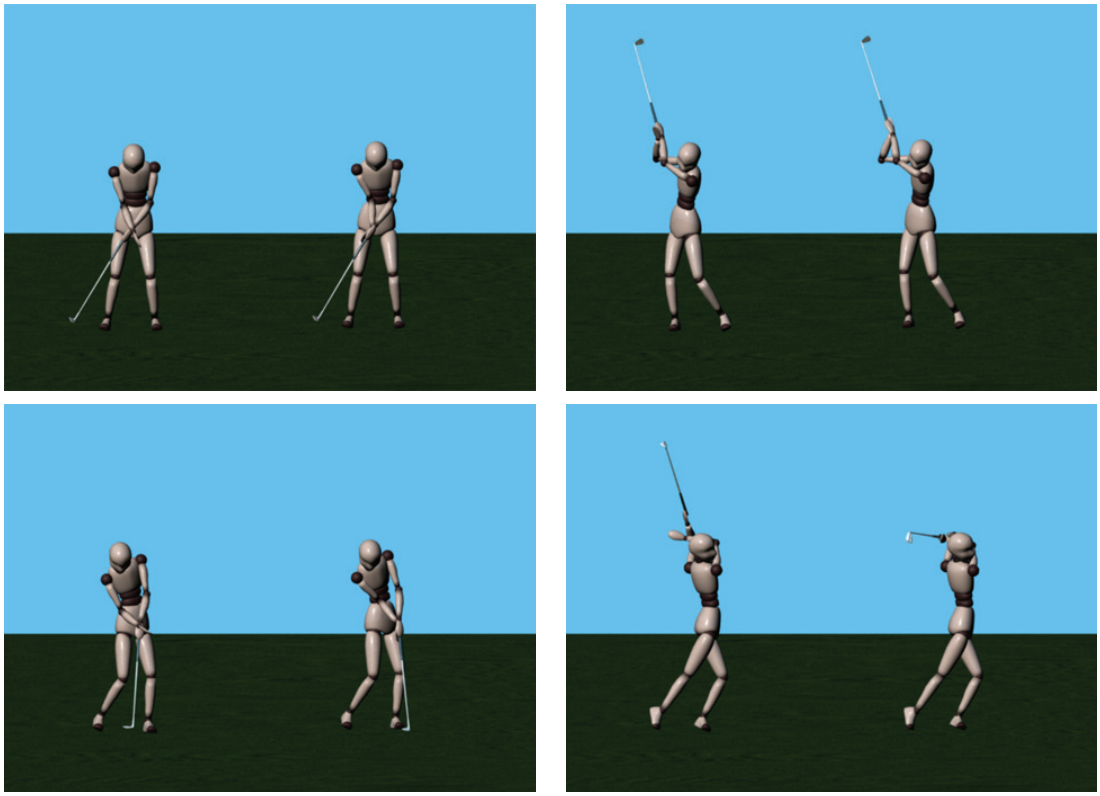

Figure 5.15: A golf swing motion. **Left character:** Initial motion. **Right character:** Edited motion.

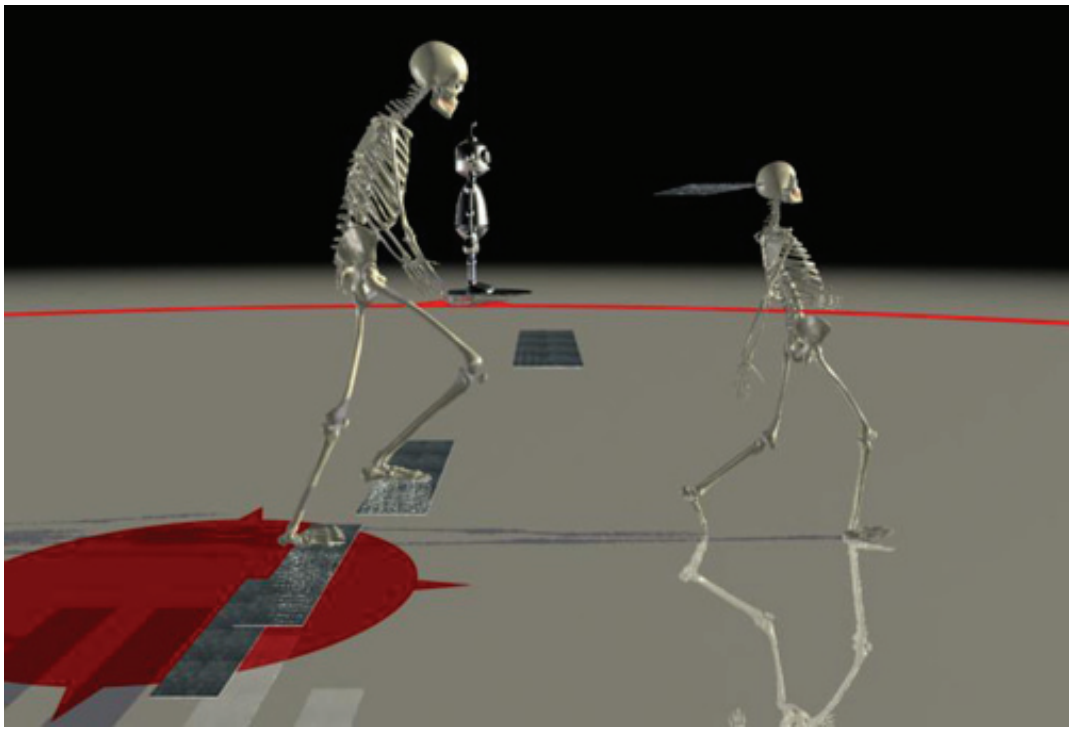

Figure 5.16: Example of a deformed animation with five shape-constraints, one rotational-constraint and CoM position control.

chains themselves, the different priorities of the constraints, their recruiting level, their activity period, interdepending constraints, etc. Badler et al. proposed in [Badler et al., 1980] an algorithmic solution to handle conflicts between overlapping kinematic chains. They did not consider priorities or even dynamically changing recruiting level, and thus made choices which could not be done in our framework. In our framework, we only consider conflicts between constraints controlling the same end-effector since it represents the most common situation when generating discontinuous trajectories. If we wanted to handle all the possible cases, the algorithm would be much more complex than the one presented in [Badler et al., 1980]. Additionally, in many cases, there is no optimal solution. As a consequence, such an algorithm would have to make a lot of compromises and choices which would be too subjective to be generic and robust. For this reason, we preferred to let the animator decide whether it is better to decrease the priority of a constraint or to change its recruiting level for example.

Our framework improves classical motion editing techniques, as animators can add large deformations without ending up with unbalanced results. Moreover, the priority concept greatly helps when animators need to arbitrate conflicting constraints. Our algorithm allows to assign a priority to each constraint. This priority is used to arbitrate conflicts between constraints. Our scheme ensures that high-priority constraints won't be disturbed by low-priority ones. Furthermore, we have proposed a simple and efficient algorithm to avoid generating discontinuous end-effectors trajectories while adding new constraints. Finally, while we have mainly focused our discussion on motion deformation, our method is also well-suited to deal with retargetting problems.

For the moment, all these constraints must be explicitly specified by the animator. However, in some cases specifying all the constraints by hand is a tedious and time-consuming process. We found useful to be able to automatically detect the initial set of constraints the user could be interested in.

In the next chapter, we present a method which is able to detect geometric constraints so that the animator only needs to adjust them instead of specifying them all by hand.

CHAPTER 6

Geometric Constraint Detection for Motion Capture Animation

Constraint-based motion editing techniques are designed to change existing motion sequences while retaining as many of their initial characteristics as possible. These characteristics are often rendered explicit using *geometric constraints* (simply called constraints in the remainder of this chapter) i.e. some parts of the virtual character remain stationary with respect to some reference frame for a period of time. However, in some cases, manually defining all these constraints can prove to be time-consuming. Consequently, it is often desirable to automate the detection of such constraints in order to simplify and speed up the motion editing process.

We divide constraints into two subcategories:

1. **Intrinsic Constraints:** they represent constraints that are only related to the motion of the virtual character. They can be detected by considering only the animation itself. For example, footplants are intrinsic constraints. The right hand of a character rotating around an axis fixed in the world coordinate system is also an intrinsic constraint.

2. **Interaction Constraints:** they represent an interaction between the virtual character and objects in the scene. As a consequence, they *cannot* be detected by considering the character's animation alone: we must also take the motion of some objects of interest in the scene into account. For example, if the character grabs a bottle on the table and moves it to another location, there is an interaction constraint between the hand and the bottle: the hand is stationary with respect to the bottle.

It is important to note that interaction constraints with stationary objects *are considered* as intrinsic constraints as they can be detected by only considering the motion of the virtual character.

In this chapter, we present a framework which helps the animator edit the motion of a virtual character by semi-automatically detecting the intrinsic constraints in the initial animation (we later explain why no constraint detection algorithm can be fully automated). The animator is then free to edit (or remove) these constraints so as to retain only those of interest (some constraints turn out to be useless depending on the result the animator wants to achieve).

The next section gives a definition of the constraints we are interested in. Section 6.2 reviews previous work on constraint detection. Section 6.3 provides an overview of our method. In Section 6.4, we present our algorithm for semi-automatically detecting intrinsic constraints in motion. In Section 6.5 we present experimental results. Finally, we discuss its limitations and conclude in Section 6.6.

6.1 Constraints Definition

The purpose of this chapter is to present a method to semi-automatically detect constraints related to specific body parts of a virtual character. As stated before, they are directly related to its motion. Hence, if we only consider the motion of a body part, we can then identify three types of constraints:

- the body part is stationary in space. This is a **space constraint**,
- the body part is rotating around a line. This is a **line constraint**,
- the body part is rotating around a point. This is a **point constraint**.

Moreover, all these constraints share a common property: they each occur during a specific time interval $[t_{begin}, t_{end}]$.

The next sections separately detail these constraints.

Space Constraints

A space constraint emphasizes the fact that if we apply the motion of the considered body part to all the points in space, then they all remain stationary. In particular, the body part is also stationary. This occurs when all 6 DoFs of a body part are fixed. A space constraint is then parametrized with:

- a position in space,
- an orientation in space.

Space constraints are the most meaningful constraints as they represent events in the motion that are easily identified. For example during a walking motion, a space constraint occurs each time one foot stays planted on the ground.

Line Constraints

A line constraint emphasizes the fact that if we apply the motion of the considered body part to all the points in space, then all the points lying on a specific line (and only those) remain stationary. In particular, the body part rotates around this line. This occurs when 5 DoFs of a body part are fixed (3 for the translation and 2 for the rotation).

A line constraint is then parametrized with:

- an origin in space
- a direction in space.

A line constraint may be assimilated to a revolute joint: it only has one DoF. For example, a line constraint occurs when one is opening a door: the hand rotates around the axis of rotation of the door.

Point Constraints

A point constraint emphasizes the fact that if we apply the motion of the considered body part to all the points in space, then a single point remains stationary. In particular, the body part rotates around this point. This occurs when the 3 DoFs of the translation of a body part are fixed.

A point constraint is then parametrized with:

- a position in space.

A point constraint may be assimilated to a spherical joint: it has three DoFs. For example, a point constraint occurs when the fortune teller cleans her crystal ball: the right hand rotates around the center of the ball.

Sliding Constraints

Sliding constraints may be assimilated in particular to prismatic joints, cylindrical joints, planar joints or screw joints. These constraints are more difficult to detect and necessitate complex minimization techniques that our algorithm cannot handle: indeed, our main focus is to develop interactive methods and it is often hard to combine minimization methods with interactive rates. As a consequence, this group of constraints is not handled by our algorithm.

6.2 State of the Art in *Constraints Detection*

Very few results on automatic (or even semi-automatic) constraint detection can be found in the literature. A classical application of constraint detection is the identification of footplants in motions. Several methods [Kovar et al., 2002a; Menardais et al., 2004] used specific thresholds on the position and velocity of the feet to detect these constraints. Similarly,

Lee et al. extended this approach in [Lee et al., 2002] to body segments and objects in the environment. They consider their relative velocity and position to decide whether a body segment is in contact with an object in the scene or not. However, the thresholds used in these techniques are closely related to the nature of the considered motions. As a consequence, they have to be finely tuned depending on whether the virtual character is walking, running, jumping, etc.

In [Bindiganavale and Badler, 1998] the authors presented a method mapping the animation of a subject being motion captured (the primary agent) to another virtual character having different proportions (the secondary agent). They essentially focused on motions containing interactions with the surrounding environment and thus introduced a method to detect interactions between the character and objects in the environment. The main point of their technique is to identify collisions between end-effectors and the objects in the scene. However, to avoid checking for collision at every frame of the animation, they supposed that potential frames of interest are located at the zero-crossing of the second derivative of the end-effectors. Finally, they used tagged-objects having predefined sites of interest to, in particular, avoid checking for collisions with all the objects in the scene.

Finally, Liu and Popović [Liu and Popovic, 2002] proposed a generic method to detect intrinsic constraints. All the points remaining stationary from one frame to another are first computed. These spaces of stationary points may be of dimension:

- 1: all the points on a line in space remain stationary
- 3: all the points in space remain stationary

They then compute the intersections of these points with body parts of the character to determine the dimension and duration of the intrinsic constraints. It is important to note that despite what was stated in [Liu and Popovic, 2002], no rigid transformation exists such that a single point or all the points belonging to a plane remain stationary *from one frame to another*.

However, all these methods prove to be unreliable if the original motion is noisy such as motion capture data. In the next sections we propose a technique to semi-automatically extract intrinsic constraints (from the motions themselves) using motion capture animations as input. We demonstrate, using several examples, that our algorithm is robust enough to accurately detect constraints even with highly noisy data.

6.3 Method Overview

To simplify the exposition and without loss of generality, we focus our discussion on the detection of constraints related to a single animated object. Formally, the problem of detecting all the constraints related to a given object $\boldsymbol{O}$ with respect to a reference frame $\boldsymbol{R}$ (e.g. the world coordinate system or the local frame of a moving object) over a period of time is to identify their duration as well as their parameters depending on their type (space, line or point). Our algorithm is composed of the successive stages shown in Figure 6.1. Basically, given the animation of an object, we first detect its associated constraints between each pair

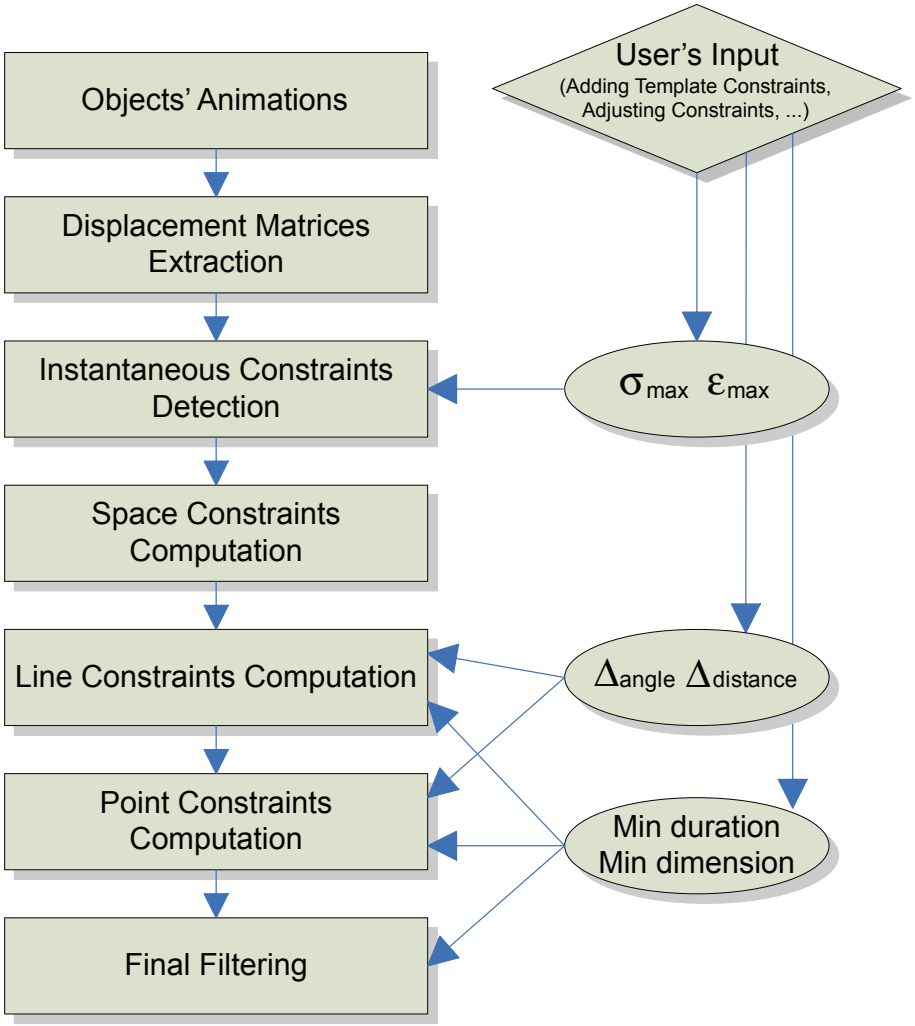

Figure 6.1: Overview of the constraint detection algorithm. **Boxes:** Stages of the algorithm. **Diamond:** User's input: he may manually add constraints (called template constraints), remove wrong constraints, etc. **Ellipsoids:** Important algorithm's parameters.

of successive frames: the *instantaneous constraints*. Then, each of these instantaneous constraints are merged as much possible with its neighbors to end up with a minimal set of constraints. Each of these steps is summarized in the next sections and detailed in Sections 6.4.1, 6.4.2, 6.4.3 and 6.4.4.

6.3.1 Displacement Matrices Extraction:

We are interested in the constraints related to a given object $\boldsymbol{O}$ with respect to a reference frame $\boldsymbol{R}$ over a period of time. In our framework, the motions of $\boldsymbol{O}$ and $\boldsymbol{R}$ are expressed in the world coordinate system. As a consequence, the first step of our algorithm is to express the relative motion of object $\boldsymbol{O}$ with respect to the reference frame $\boldsymbol{R}$. This results in a sequence of *displacement matrices* expressing the displacement of $\boldsymbol{O}$ in $\boldsymbol{R}$ from each frame to the next one. The displacement matrices extraction step is explained in more detail in

Section 6.4.1.

6.3.2 Instantaneous Constraint Detection

The problem of constraint detection is similar to identifying all the points in space remaining stationary for each displacement matrix. As previously stated, these *fixed points* define stationary geometries that may be of dimension:

- 1: all the points on a line in space remain stationary,
- 3: all the points in space remain stationary.

In the remainder of this chapter, these stationary geometries are referred to as *instantaneous constraints*, as they only last for one frame.

Special care must be taken during this step, as we are dealing with motion capture data. Indeed, in such a context, high-dimensional constraints rarely (or never) occur. For example, even though an object is *visually* still, it is *numerically* moving. In other words, while its associated displacement matrix should be the identity it is never the case. We, therefore introduce two parameters, σ_{max} and ϵ_{max} in our algorithm, which increase its robustness during the instantaneous constraint detection stage. These parameters directly depend on the noise present in the original data. They are exploited in the singular value decomposition used during this stage. This is why, they are often referred to as the SVD-related parameters. The animator can interactively change them to detect the expected instantaneous constraints. However, this is often far from being sufficient and practical. As a consequence, given a small set of predefined constraints determined by the animator (which we refer to as *template constraints* in the remainder of this chapter), we automatically estimate these parameters to considerably reduce manual tweaking. We explain their importance and their computation in more detail in Section 6.4.2. At the end of this stage, we end up with a list of instantaneous constraints. Note that their parameters are already known at this stage. That is, for a space instantaneous constraint, we know its position and orientation, for a line instantaneous constraint, we know its origin and its direction and for a point instantaneous constraint, we know its position.

6.3.3 Computation of the Effective Constraints

The previous stage provides a list of instantaneous constraints lasting one frame only. However, this is not the minimal set of constraints we are interested in: we need to estimate their real duration as well as their real position by merging possible neighboring instantaneous constraints as much as possible.

Space Constraints Computation: If during n consecutive frames there are $n-1$ space instantaneous constraints, we can advantageously replace all of them by a single *space constraint* lasting n frames. We then need to compute the average position of those $n-1$ space instantaneous constraints. Once again, it is important to note that as we are dealing with

motion capture data, space instantaneous constraints may not be consecutive in time while they should in practice. As a result, we also allow space constraints which are close in space as well as in time to be merged. This stage is explained in more detail in Section 6.4.3.2.

Line Constraints Computation: Given the results of the previous stage, we now merge space instantaneous constraints and/or line instantaneous constraints as much as possible to determine the exact duration and the exact position of *line constraints* . Due to the noisy nature of the data, we introduce thresholds (Δ_{angle} and $\Delta_{distance}$) to decide whether two lines in space are the same or not. In the remainder of this chapter, they may be referred to as the line-related parameters. Note that some particular space constraints may be removed to extend the duration of line constraints. Moreover, as stated before in the case of space constraints, a line constraint may not be continuous in time while it should be in practice. Hence, line constraints that are close in space as well as in time are merged. We give more detail about line constraints computation in Section 6.4.3.3.

Point Constraints Computation: At this stage, the final point constraints are computed depending on the intersections of several lines constraints. Here again, some particular line constraints may be removed to extend the duration of point constraints. For the same reasons as above, point constraints that are close in space as well as in time are merged. Finally, two points that are *visually* the same may be *numerically* different. For this reason, as for the line constraints, we consider a distance threshold ($\Delta_{distance}$) to decide whether two points are identical or not. This is detailed in Section 6.4.3.4.

6.3.4 Final Filtering

The user may not be interested in all the constraints. He could instead focus on constraints lasting for a minimum amount of frames and/or being of a specific dimension. This stage filters the previous results so that constraints not meeting the user's requirements are ignored.

6.4 Intrinsic Constraint Detection

In this section, we detail all the steps presented in Section 6.3. We are essentially concerned in detecting *intrinsic constraints*. In this section the term "constraints" refers to intrinsic constraints if not explicitly stated otherwise.

6.4.1 Displacement Matrices Extraction

As previously stated, the first step of the algorithm is to express the motion of object $\boldsymbol{O}$ with respect to a reference frame $\boldsymbol{R}$. As we focus on intrinsic constraint detection, the reference frame is equivalent to the world coordinate system (see Figure 6.2 for a conceptual description of the matrices involved in this computation).

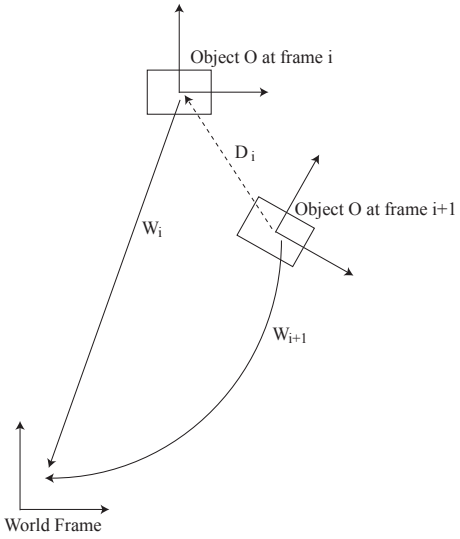

Figure 6.2: Transformations used to compute the displacement matrix $\boldsymbol{D_i}$ from frame i to frame $i+1$. W_i is the transformation from the $\boldsymbol{O}$ local coordinate system at frame i to the world coordinate system. W_{i+1} is the transformation from the $\boldsymbol{O}$ local coordinate system at frame $i+1$ to the world coordinate system. D_i is the $\boldsymbol{O}$ displacement matrix between frame i and frame $i+1$.

The only data we can rely on correspond to the animation of object $\boldsymbol{O}$. This latter may be viewed as a sequence of transformation matrices giving the position of object $\boldsymbol{O}$ at each frame of the animation. As a consequence, let us consider $\boldsymbol{W}_i$ the matrix transforming, at frame i, a point $\boldsymbol{p}$ expressed in the $\boldsymbol{O}$ local coordinate system to $\boldsymbol{x_i}$ expressed in the world coordinate system. More formally we have:

$$\hat{\boldsymbol{x_i}} = \boldsymbol{W_i}\hat{\boldsymbol{p}} \tag{6.1}$$

with $\hat{\boldsymbol{x_i}}$ (resp. $\hat{\boldsymbol{p}}$) the homogeneous coordinates of point $\boldsymbol{x_i}$ (resp. point $\boldsymbol{p}$). Similarly, at frame $i+1$, we have:

$$\begin{aligned} \hat{\boldsymbol{x_{i+1}}} &= \boldsymbol{W_{i+1}}\hat{\boldsymbol{p}} \\ &= \boldsymbol{W_{i+1}}\boldsymbol{W_i}^{-1}\boldsymbol{W_i}\hat{\boldsymbol{p}} \\ &= \boldsymbol{W_{i+1}}\boldsymbol{W_i}^{-1}\hat{\boldsymbol{x_i}} \end{aligned}$$

We can then define the displacement $\boldsymbol{D_i}$ of point $\boldsymbol{x}$ between frame i and frame $i+1$ as:

$$\boldsymbol{D_i} = \boldsymbol{W_{i+1}}\boldsymbol{W_i}^{-1} \tag{6.2}$$

This formulation is similar to the one presented in [Liu and Popovic, 2002]. It represents the displacement of object $\boldsymbol{O}$ with respect to the world coordinate system. However, this *global formulation* is not correct as it introduces a bias in subsequent stages of the algorithm. Indeed, in this formulation, the displacement matrix $\boldsymbol{D_i}$ is directly dependent on the global position of object $\boldsymbol{O}$ at frames i and $i+1$. We explain in Section 6.4.2 why this may lead to inaccuracies in our algorithm.

We therefore reformulate the problem by expressing the displacement $\boldsymbol{D_i}$ of object $\boldsymbol{O}$ with respect to its previous position:

$$\begin{aligned} \boldsymbol{D_i} &= \boldsymbol{W_i}^{-1}\boldsymbol{W_{i+1}}\boldsymbol{W_i}^{-1}\boldsymbol{W_i} \\ &= \boldsymbol{W_i}^{-1}\boldsymbol{W_{i+1}} \end{aligned}$$

This *local formulation* is much more accurate as it is independent from the global position of object $\boldsymbol{O}$. In Appendix A.1 we demonstrate that the global formulation is not correct in our context and why we should use the local formulation instead.

We are now able to compute all the displacement matrices $\boldsymbol{D_i}$ related to the animation of object $\boldsymbol{O}$. The next section details our algorithm to estimate, for each displacement matrix, all the points in space remaining stationary. These points serve as a basis to determine the final set of constraints we are looking for.

6.4.2 Instantaneous Constraint Detection

Given the displacement matrix $\boldsymbol{D_i}$ of an object $\boldsymbol{O}$ from frame i to frame $i+1$, we need to find all the points $\boldsymbol{p}$ remaining stationary in space. More formally, we have to solve:

$$\boldsymbol{D_i}\hat{\boldsymbol{p}} = \hat{\boldsymbol{p}} \tag{6.3}$$

where $\hat{\boldsymbol{p}}$ is the homogeneous coordinates of a point $\boldsymbol{p}$ expressed in the $\boldsymbol{O}$ local coordinate system at frame i and $\boldsymbol{D_i}$ the displacement matrix of $\boldsymbol{O}$ from frame i to frame $i+1$.

Thus, we need to solve:

$$(\boldsymbol{D_i} - \boldsymbol{I_4})\hat{\boldsymbol{p}} = 0 \tag{6.4}$$

In our framework, we are interested in finding *all* the solutions of Equation 6.4. Indeed, suppose that the displacement matrix $\boldsymbol{D_i}$ represents a rotation along an axis $\boldsymbol{R_{axis}}$. It is clearly not satisfactory to know that a specific point (actually lying on the axis $\boldsymbol{R_{axis}}$) is stationary is space: we need to determine the equation of that axis of rotation. As a consequence, a straightforward method finding a single solution (the least squares for example) is not usable.

As $\boldsymbol{D_i}$ is a rigid transformation, it can be rewritten as:

$$\boldsymbol{D_i} = \begin{bmatrix} \boldsymbol{R_i} & \boldsymbol{t_i} \\ \boldsymbol{0_3} & 1 \end{bmatrix} \tag{6.5}$$

with $\boldsymbol{R_i}$ and $\boldsymbol{t_i}$ respectively the rotational and translational components of $\boldsymbol{D_i}$.

We then reformulate Equation 6.4 as follows:

$$(\boldsymbol{R_i} - \boldsymbol{I_3})\boldsymbol{p} + \boldsymbol{t_i} = 0 \tag{6.6}$$

$$\boldsymbol{A}\boldsymbol{p} = -\boldsymbol{t_i} \tag{6.7}$$

Using a singular value decomposition, we can express matrix $\boldsymbol{A}$ as:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{6.8}$$

with $\boldsymbol{U}$ and $\boldsymbol{V}$ being 3×3 orthogonal matrices and $\boldsymbol{\Sigma}$ a 3×3 diagonal matrix [Press et al., 1992]. Matrix $\boldsymbol{\Sigma}$ is a diagonal matrix containing the 3 *singular values* $\sigma_{i=1,2,3}$ (with $\sigma_1 > \sigma_2 > \sigma_3$) of matrix $\boldsymbol{A}$. Moreover, matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ span the *range* and the *nullspace* of matrix $\boldsymbol{A}$: the columns of $\boldsymbol{U}$, whose same-numbered elements σ_i are *nonzero*, are an orthonormal set of basis vectors which span the range. The columns of $\boldsymbol{V}$, whose same-numbered elements σ_i are *zero*, are an orthonormal basis for the nullspace [Press et al., 1992].

To solve our problem we then need to compute:

1. A basis for the nullspace of $\boldsymbol{A}$,
2. A particular solution $\boldsymbol{p_{particular}}$ to Equation 6.7.

As stated before, a basis of the nullspace of $\boldsymbol{A}$ is computed by taking the columns of $\boldsymbol{V}$ whose same-numbered elements σ_i are *zero*. Generally, no σ_i is exactly zero. We therefore introduce a threshold σ_{max} beneath which the singular values are zeroed.

The particular solution $\boldsymbol{p_{particular}}$ (in the least squares sense) corresponding to Equation 6.7 is computed as:

$$\begin{aligned} \boldsymbol{p_{particular}} &= -\boldsymbol{A}^{-1}\boldsymbol{t_i} \\ &= -\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T\boldsymbol{t_i} \end{aligned}$$

At this point, we should obtain all the solutions we are interested in: they are given by the particular solution $\boldsymbol{p_{particular}}$ and the basis of the nullspace of $\boldsymbol{A}$. Unfortunately, Equation 6.7 is not always consistent. In this case, this method actually produces a solution while it should not. We then need to additionally check whether such a solution is relevant or not. To do so, we compute the residual error ϵ of Equation 6.7. This residual error is defined as:

$$\epsilon = \|\boldsymbol{A}\boldsymbol{p_{particular}} + \boldsymbol{t_i}\| \tag{6.9}$$

Then, whether ϵ is under a specific threshold ϵ_{max} or not determines if the solution can be retained or must be discarded.

In summary, we can directly influence the solution of Equation 6.7 using two parameters:

1. σ_{max}: threshold beneath which the singular values are zeroed. The number of null singular values defines the dimension of the instantaneous constraint,
2. ϵ_{max}: residual error of equation 6.7 (after zeroing the singular values smaller than σ_{max}) beneath which the solution is acceptable.

The problem of detecting instantaneous constraints can then be reformulated as estimating these SVD-related parameters so that we effectively detect the *expected* instantaneous constraints. An accurate estimation of the SVD-related parameters is crucial for the algorithm. Indeed, if they are underestimated (resp. overestimated), the algorithm detects too few (resp. too many) instantaneous constraints. Several issues then arise:

- It is not reasonable to ask the animator to provide such parameters as the results tend to be difficult to foresee.
- These parameters are very different from one motion to another.
- Motions containing aberrant frames (called hereafter outliers in the remainder of this chapter) are quite common. In such a situation, estimation is much more arduous.

Hence, we rely on the animator to provide a small set of predefined constraints (the ***template constraints***) to help the algorithm calibrate these parameters and then accurately detect the instantaneous constraints for the entire animation.

Template Constraint Specification

To specify a template constraint, the user only needs to specify a time interval $[t_{begin}, t_{end}]$ as well as the type of constraints expected (i.e. space, line or point). As a consequence, a template constraint may be thought of as a set of consecutive displacement matrices for which we precisely know in advance the solution of the instantaneous constraint detection.

For instance, if the user specifies that a line constraint should occur during time interval $[t_{begin}, t_{end}]$, we know that for all associated displacement matrices, σ_3 must be inferior to σ_{max}. Moreover, we also know that for each displacement matrix, the associated residual error ϵ must be inferior to ϵ_{max}. Hence, *in theory*, these displacement matrices are sufficient to estimate the SVD-related parameters.

However, as we are dealing with motion capture data, the presence of noise (and particularly, the presence of outliers) tends to bias the estimation of these parameters. In the next section, we propose an algorithm to overcome this problem. We first begin by defining what we call an outlier in our context. We then show that a naive (straightforward method) is clearly insufficient. We finally detail our method to robustly estimate the SVD-related parameters based on a template constraint.

Robust Computation of the Instantaneous Constraints

In this paragraph, we focus our discussion on the estimation of the SVD-related parameters based on a single template constraint only. In the next paragraph, we detail how this method is extended to handle several template constraints simultaneously.

Before detailing our algorithm, we first need to define what we consider as an outlier. An outlier is an observation that lies outside the overall pattern of a distribution [Moore and McCabe, 1999]. In our case, we must slightly restrict this definition. Indeed, an outlier is a particular datum biasing the estimation of some model. In our case, the model we want to estimate is the SVD-related parameters. This means that we must consider outliers only while estimating the SVD-related parameters. As a result, in our context, an outlier is a frame which is labeled (by the user) as being part of a template constraint while it should not: the so-called *aberrant frame*.

Given a template constraint, a possible naive method is to first estimate the σ_{max} parameter so that the solution of Equation 6.7 is of the required dimension for each associated

displacement matrix. Afterward, we can easily compute the ϵ_{max} threshold to accept the solution at frames where the template constraint has been specified. This naive method is summarized in Algorithm 2.

Algorithm 2 Naive method to estimate the SVD-related parameters

```
D ← displacement matrices corresponding to the specified template constraint
σ_max = maximum singular value so far (initialized to infinity)
ε_max = maximum residual error so far (initialized to infinity)
required_dim = required dimension
for all D_i in D do
    S = extract σ_{4−required_dim} from D_i
    E = compute residual error given D_i and S
    if S > σ_max then
        σ_max = S
    end if
    if E > ε_max then
        ε_max = E
    end if
end for
```

However, such an approach is far from being efficient as it leads to an inaccurate estimation of the parameters resulting in the detection of too many instantaneous constraints.

Indeed, supposing that a template space constraint of dimension three has been specified between frame a and frame b (i.e. the considered object remains stationary between frame a and frame b). Supposing also that during this period of time, the frame i (with $a < i < b$) is an outlier. In this case, σ_{max} is overestimated to ensure that the solution of Equation 6.7 is of dimension three for the specified interval and particularly for the intervals $[i-1, i]$ and $[i, i+1]$.

Moreover, this overestimation of σ_{max} directly influences the estimated ϵ_{max}. Indeed, the residual error of equation 6.7 where the singular values have been zeroed is very important. As a consequence ϵ_{max} is overestimated as well. Finally, the algorithm tends to detect too many instantaneous constraints.

We propose a robust estimation of the SVD-related parameters based on the least median of squares method (LMedS) [Rousseeuw and Leroy, 1987] to identify and reject potential outliers. Let SV be the set of singular values containing:

- all the σ_1 corresponding to the associated displacement matrices if the specified template constraint is a space constraint,
- all the σ_3 corresponding to the associated displacement matrices if the specified template constraint is a line constraint.

We want to find all the $\sigma_i \in SV$ that *significantly* deviate from the others:

1. For each $\sigma_i \in SV$, we compute the median of its squared residuals M_i as:

$$M_i = med\, r_i^2(\sigma_i, SV)$$

where $r_i^2(\sigma_i, SV)$ is the residual error associated to σ_i with respect to SV.

2. We retain M_{min} (and its associated singular value σ_{med}) the smallest M_i among all the M_is

3. We then compute the robust standard deviation as

$$\hat{\sigma} = 1.4826[1 + 5/(N_{SV} - 1)]\sqrt{M_{min}}$$

where M_{min} is the minimal median and N_{SV} the number of singular values.

4. Finally, we reject all the singular values such that:

$$r_i^2(\sigma_i, SV) \geq (2.5\hat{\sigma})^2$$

The reader can refer to [Rousseeuw and Leroy, 1987] for a more detailed explanation of the LMedS method. It is important to note that in our method, we *do not* perform any random selection as the space of possible solutions is SV. We can therefore afford to estimate *all* the possible solutions as there are relatively few. σ_{max} is then the maximum of all the *good* singular values remaining in SV. Finally, ϵ_{max} is estimated so as to detect the expected instantaneous constraints using the same method.

Figure 6.3 shows a comparison of both methods in the case of footplant detection. The

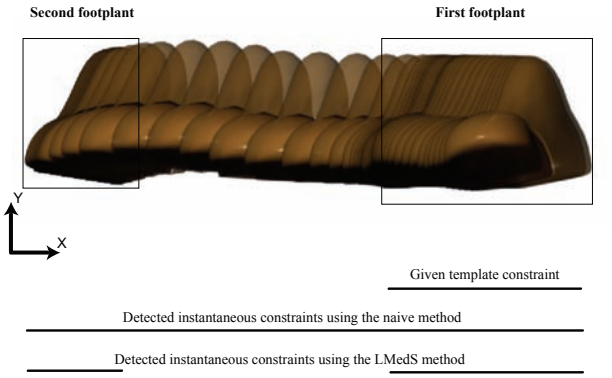

Figure 6.3: Detecting footplants: while the LMedS-based method detects two footplants, the naive method yields to an erroneous estimation by merging both footplants into a single one.

animation represents two footplants. In this example, we immediately see that while the animation contains two footplants, the naive method only detects one. The first frames of the animation are very noisy. Indeed, we can easily see that whereas the foot is labeled as being stationary, it is actually rotating (and also moving in translation). This motion induces an important bias in the estimation of parameter σ_{max}. This wrong estimation directly produces a wrong estimation of parameter ϵ_{max} as well. Using the LMedS method, this bias is avoided and our method produces the expected results.

Figure 6.4 shows the numerical values corresponding to the example shown in Figure 6.3. While beginning frames are specified as being a template constraint by the animator, they should not be retained during the computation of the SVD-related parameters. While our

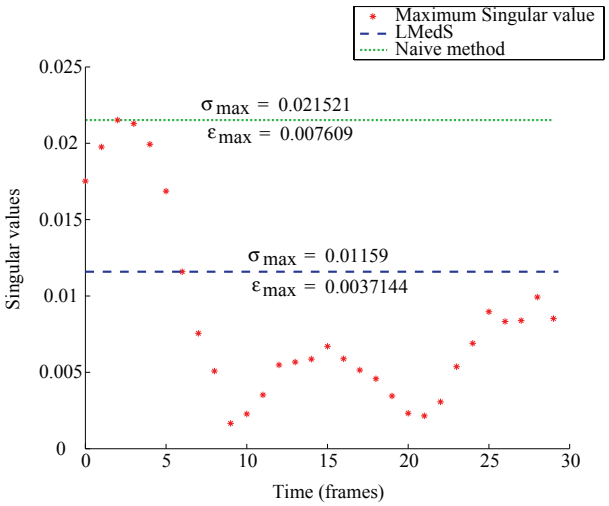

Figure 6.4: Comparison between naive and LMedS methods.

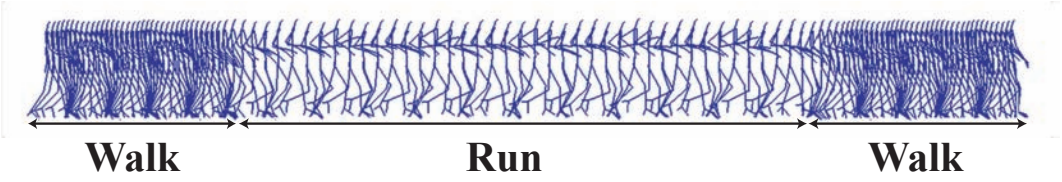

Figure 6.5: Test motion: the character first walks, then runs and finally walks again.

robust estimation effectively detects and rejects these outliers, the naive one produces wrong estimations. As a consequence, the naive method detects too many instantaneous constraints (actually, at each frame) while our method clearly identifies the two footplants we want to detect.

The next section presents a method handling several template constraints at once.

Dynamic Estimation of the SVD-related parameters

We have shown so far how to automatically estimate the parameters related to instantaneous constraint detection: the so-called SVD-related parameters. Given a single template constraint, we estimate the SVD-related parameters for the whole motion. However, if the motion is long enough, it often contains different actions. Each of these actions may lead to different *noise patterns* and then, SVD-related parameters which are correct for a particular part of the motion, may not be suited for the entire movement. For example, let us consider a motion in which the virtual character first walks, then runs and finally walks again as shown in figure 6.5. In this case, it is likely that for the time interval during which the character is running, the SVD-related parameters are higher than for the parts of the motion for which the character is walking. Figure 6.6 shows, for each frame, the maximum singular value as well as the associated residual error for the considered motion.

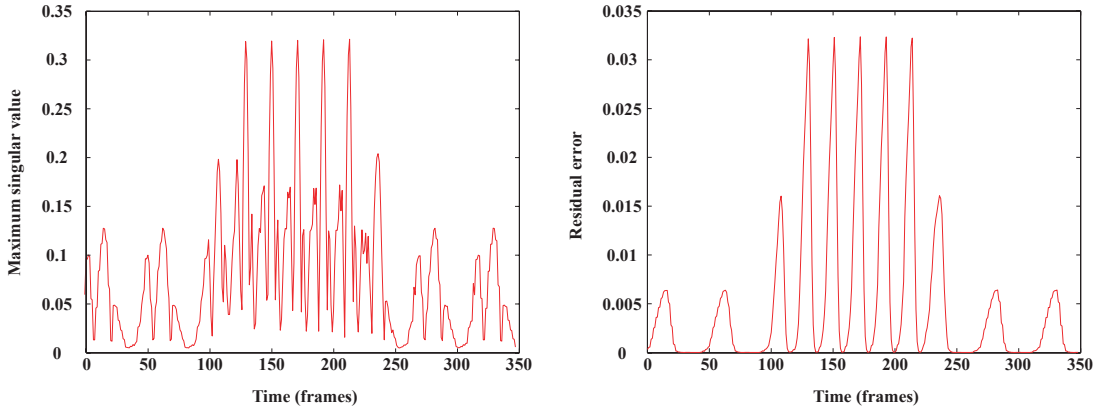

Figure 6.6: The maximum singular value and the corresponding residual error for the animation shown in Figure 6.5. **Left:** Maximum singular value. **Right:** Associated residual error

As a consequence, it is not satisfactory to consider static SVD-related parameters for the entire animation. Hence, we propose a coarse to fine approach to represent the SVD-related parameters. Each of these parameters is represented using a cubic interpolation spline. For each given template constraint, we robustly compute, using the previously detailed method, the associated SVD-related parameters. Each parameter is then used as two control points in its associated spline: one control point for the begin frame of the template constraint, and one for its end frame. Figure 6.7 shows an example of the splines associated to the SVD-related parameters after adding a set of several template constraints.

Doing so, we ensure that the SVD-related parameters are accurate for the period of time defined by a given template constraint. Subsequent additions of template constraints then refine the curves associated with the SVD-related parameters. As a consequence, the SVD-related parameters are dynamically changed to ensure an accurate instantaneous constraint detection.

In this section, we have shown how to compute constraints between two frames: the so-called *instantaneous constraints*. Moreover, we have detailed a robust algorithm able to estimate key parameters of our method even when the initial motion is noisy and/or when the animator mislabels frames as being part of a template constraint. This stage of the algorithm provides us with a list of instantaneous constraints. As it is not the minimal set of constraints we are interested in, we need to estimate them by merging possible neighboring instantaneous constraints as much as possible.

6.4.3 Computation of the Effective Constraints

In this section, we describe our method to compute the effective constraints. Given a list of instantaneous constraints, we need to compute the minimal set of constraints. Thus, we need to merge instantaneous constraints as much as possible. For instance, if during n consecutive frames there are $n-1$ space instantaneous constraints, we want to replace all of them by a single space constraint lasting for n frames. The merging process may then be decomposed

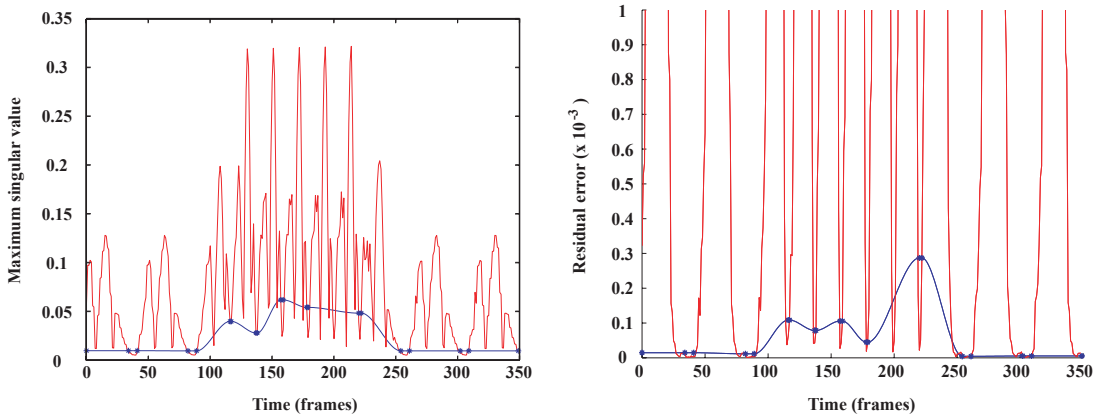

Figure 6.7: Dynamic SVD-related parameters estimation using cubic interpolating splines. The associated animation is shown in Figure 6.5. **Left:** Maximum singular value (in red) and corresponding threshold spline (in blue). **Right:** Residual error (in red) and corresponding threshold spline (in blue). The units are different from Figure 6.6 to highlight the details.

into two successive steps:

1. First estimate the real duration of the constraints,
2. Then estimate their average position depending on their dimension and the previously computed duration.

We consider the following requirements while merging instantaneous constraints:

1. We want to promote high-dimensional constraints. Indeed, we believe that it is semantically more important for the animator to know that a specific part of the virtual character is stationary in space rather than knowing that this latter is rotating around a specific axis of rotation. As a result, when conflicts arise we decide to first promote space constraints, then line constraints and finally point constraints. We then process the constraints in this order to ensure that low-dimensional constraints computation does not disturb high-dimensional ones.
2. We do not want constraints to overlap in time. As a consequence, if a constraint already occurs at a specific frame, it is not necessary to search for constraints of lower dimension.

In order to merge two constraints, we need to check for some requirements:

- **In space:** we need to check for some kind of *intersection* between both constraints. For example, two line constraints may intersect and result in a line if they are the same or a point otherwise.
- **In time:** we need to check whether the constraints are *temporally connected* or not: the ending frame of the first constraint corresponds to the beginning frame of the second

one. For example, two line constraints may intersect in space but occur at different times (they are separated by several frames). In this case, they should not be merged.

Two constraints have to meet both requirements to be merged. While the concept of *temporal connection* is independent from the dimension of the constraints, it is not the case for the concept of *space intersection*.

When the motion contains outliers, the method presented in the previous section discards the associated frames during the instantaneous constraint detection step. As a consequence, when we estimate whether two constraints are temporally connected or not, we have to take these frames into account to avoid ending up with artificially sliced constraints.

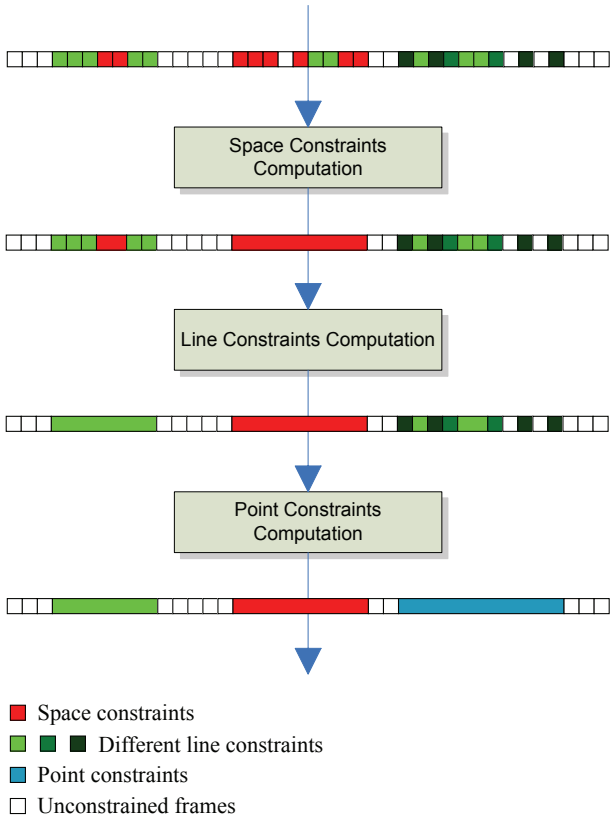

Figure 6.8: Given a list of instantaneous constraints, the successive merging stages produce a new list of constraints with a robust estimation of their respective duration. This notation is used throughout the next sections.

The next sections detail our algorithm for computing the final set of constraints given a set of instantaneous constraints. We first begin by presenting a robust approach to estimate whether two constraints are temporally connected or not. We then detail the algorithm, for each dimension, to decide whether we can merge them or not. Figure 6.8 shows an overview of the successive merging stages. Note that each merging stage also computes the average parameters for each computed constraint.

6.4.3.1 Robust Temporal Connection Estimation

As instantaneous constraints are computed with respect to thresholds that are estimated as conservatively as possible (to avoid detecting too many instantaneous constraints) it may happen that a constraint is separated into two pieces in spite of being continuous in time. Consider a simple example in which a particle is supposed to stay stationary for 10 frames (shown in Figure 6.9). Suppose also that the particle animation contains noise: the particle

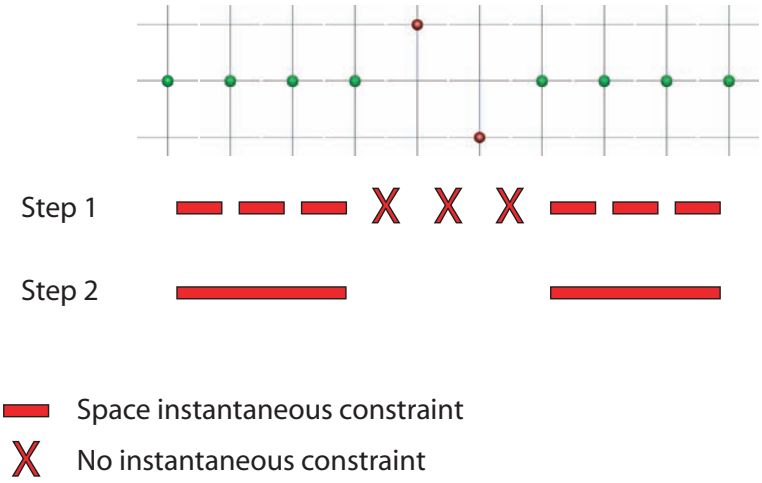

Figure 6.9: A simple particle supposed to stay stationary. **Green spheres:** Good frames. **Red spheres:** Aberrant frames. During step 1, the algorithm computes the instantaneous constraints. During step 2, the instantaneous constraints are merged when possible.

suddenly moves during three frames (the so-called aberrant frames). During the first step, the instantaneous constraints are computed. No instantaneous constraint is detected concerning the aberrant frames. When we try to merge the instantaneous constraints during the second step, we end up with two different constraints as they are temporally disconnected. In this case, we do not obtain the expected results: even though the particle is moving during three frames, we would like to *ignore* these latter and merge those two constraints into a single one.

Therefore, we propose to consider a *frame tolerance* when checking whether two constraints are temporally connected or not. This frame tolerance is defined once and then used whatever situation we are in. However, when two constraints are spatially close to one another, it is likely that even though they are temporally disconnected, they represent the same constraint: hence the frame tolerance should be large. Conversely, when two constraints are spatially far from each other in space, it is likely that even though they are temporally disconnected by only a few frames, they represent two different constraints: hence, the frame tolerance should be small.

This problem can then be stated as: given some distance $d(C_1, C_2)$ between two constraints C_1 and C_2, what is the frame tolerance for considering them as being temporally connected?

For this, we need to determine a function $f_{tol} : \mathbb{R} \longrightarrow \mathbb{N}$ returning a number of frames with respect to the chosen distance $d(C_1, C_2)$. Moreover, we require f_{tol} to respect the fol-

lowing conditions:

$$f_{tol}(d) = \begin{cases} F_{max} & \text{if } d = 0 \\ 0 & \text{if } d > d_{max} \end{cases} \tag{6.10}$$

F_{max} is the maximum number of frames allowed between two constraints in order to consider them as temporally connected. d_{max} is the maximum acceptable distance between two constraints in order to consider them as temporally connected.

We additionally want the function f_{tol} to severely decrease when the parameter d increases: we then consider the functions of the form $f(d) = \lfloor \alpha \; exp^{-\beta d} \rfloor$ where $\lfloor x \rfloor$ is the floor function giving the largest integer less than or equal to x.

Using the first condition of Equation 6.10, we have:

$$\begin{aligned} f_{tol}(0) &= F_{max} \text{ with } F_{max} \in \mathbb{N} \\ \lfloor \alpha \rfloor &= F_{max} \\ \alpha &\in [F_{max}, F_{max} + 1[\\ \text{We finally choose } \alpha &= F_{max} \end{aligned} \tag{6.11}$$

Then using the second condition of Equation 6.10, we have:

$$\begin{aligned} f_{tol}(d_{max}) &= 1 \\ \text{and } f_{tol}(d_{max}+) &= 0 \end{aligned}$$

Then we have:

$$\begin{aligned} F_{max} \; exp^{-\beta d_{max}} &= 1 \\ \text{Which leads to } \beta &= \frac{log(F_{max})}{d_{max}} \end{aligned}$$

As a result, we use the following function to define the frame tolerance between two constraints:

$$\begin{aligned} f_{tol} : \mathbb{R} &\rightarrow \mathbb{N} \\ f_{tol}(d) &= \lfloor F_{max} \; exp^{-\frac{d \log(F_{max})}{d_{max}}} \rfloor \end{aligned}$$

Figure 6.10 shows the function f_{tol} with $F_{max} = 10$ frames and $d_{max} = 10$ centimeters.

We can then robustly compute for any pair of constraints their frame tolerance as returned by the function f_{tol}. If the frame tolerance given by function f_{tol} is superior to the number of frames separating two constraints, then both constraints are temporally connected. In the remainder of this chapter, the *connected* function implicitly considers this frame tolerance.

One question still remains to be answered before we can use f_{tol}: what kind of distance should we use? This distance may only consider the position of the constraints as it is the only information we have at that time. However, the position of a constraint depends on its dimension: the position of a space constraint (resp. a line constraint) may be defined as a location (resp. an origin) in space plus an orientation (resp. a direction). In this case, the distance may consider both the Euclidean and the angular distances. As we are mixing

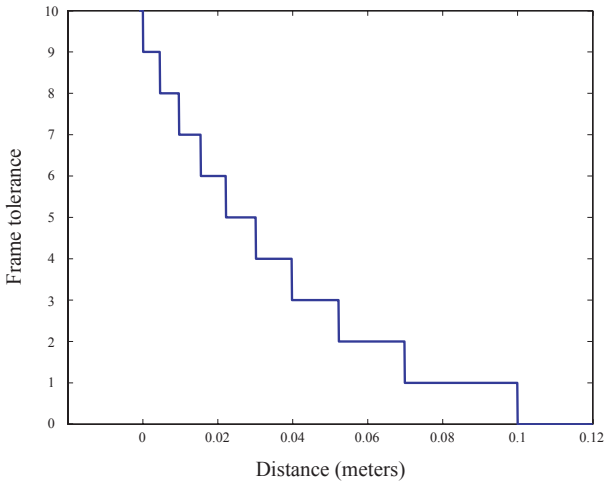

Figure 6.10: Frame tolerance with $F_{max} = 10$ frames and $d_{max} = 10$ centimeters.

quantities having different units, we should weight those two distances to end up with one distance function. However, we also need to work with point constraints. As a point has no orientation in space, such a distance function is inappropriate. We may then define two distance functions: one considering the Euclidean and angular distance when working with space and line constraints and another one only considering the Euclidean distance when working with point constraints. However, we found this solution far from being practical: indeed, two different distance functions imply two different frame tolerance functions. For genericity purposes, we therefore chose to use the Euclidean distance to compute the frame tolerance. While this choice is theoretically not correct, it proves to be in practice. In fact, most animations are stored as joint orientations for kinematic chains. Thus, slightly changing an orientation induces a translation: the amount of noise on the orientation and the translation are correlated. As a consequence, it is sufficient, in practice, to consider the Euclidean distance when estimating the frame tolerance.

6.4.3.2 Space Constraints Computation

In this section, we detail our algorithm for computing the duration of the space constraints given the initial list of instantaneous constraints resulting from the previous stages. We also explain how the average parameters of the computed constraints are estimated (i.e. the translation and the orientation components).

The computation of the duration of the space constraints involves two particular functions: the *mergeable* and the *merge* functions (used in lines 7 and 8 of Algorithm 3).

Mergeable Function In the particular case of space constraints, we consider that two space constraints C_1 and C_2 are **mergeable** if and only if they are temporally connected in time.

Figure 6.11 schematically describes this function: if the frame tolerance is more than 2 frames, then C_1 and C_2 are mergeable. Otherwise, they are considered as being distinct.

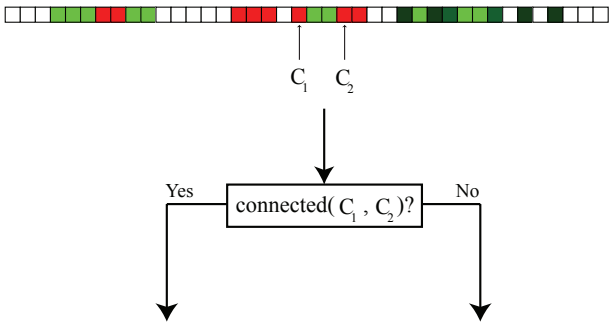

Figure 6.11: Definition of the **mergeable** function for two space constraints.

Merge Function Similarly, in the particular case of space constraints, the result of the **merging** of two space constraints C_1 and C_2 is a space constraint C_3 which continuously spans C_1 and C_2. Its translation and its orientation is computed with respect to those of C_1 and C_2 (see next paragraph). Figure 6.12 schematically describes this function. If C_1

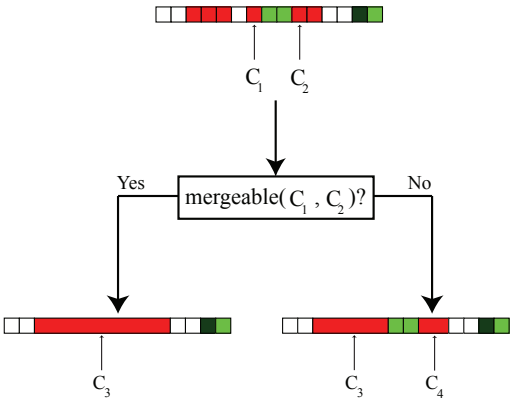

Figure 6.12: Results of the **merge** function sequentially applied to a list of instantaneous constraints.

and C_2 are mergeable, we end up with a single space constraint. Otherwise, C_1 and C_2 are considered as being distinct.

Given these functions, we compute the space constraints using Algorithm 3. Basically, we traverse the initial list of instantaneous constraints by considering only space instantaneous constraints. Line instantaneous constraints are ignored during the computation. Then, for each neighboring space constraint, we check whether we can merge them or not (i.e. whether they are temporally connected or not). If it is possible, we then replace both constraints with a single one. Note that as we want to promote high-dimensional constraints, any potential in-between line instantaneous constraint vanishes.

At this point, we have computed the duration of a space constraint corresponding to several instantaneous space constraints, each of these being parametrized by a translation and an orientation. We then have to compute its average translation and orientation.

Algorithm 3 Space constraints computation

$ICLIST_{initial} \leftarrow$ Initial list of instantaneous constraints
$CLIST_{final} \leftarrow \emptyset$ **/* Final list of constraints */**
$IC_{inbetween} \leftarrow \emptyset$ **/* List of line instantaneous constraints between two consecutive space constraints */**
Require: $ICLIST_{initial}$ contains at least one space instantaneous constraint
$IC_{active} \leftarrow$ First space instantaneous constraint in $ICLIST_{initial}$
for all remaining $IC_{current}$ in $ICLIST_{initial}$ **do**
  if $IC_{current}$ is a space instantaneous constraint **then**
    if $mergeable(IC_{active}, IC_{current})$ **then**
      $IC_{active} = merge(IC_{active}, IC_{current})$
      $IC_{inbetween} \leftarrow \emptyset$
    else
      $CLIST_{final} \leftarrow CLIST_{final} + IC_{active}$
      $CLIST_{final} \leftarrow CLIST_{final} + IC_{inbetween}$
      $IC_{active} \leftarrow IC_{current}$
    end if
  else
    $IC_{inbetween} \leftarrow IC_{inbetween} + IC_{current}$
  end if
end for
$CLIST_{final} \leftarrow CLIST_{final} + IC_{active}$
$CLIST_{final} \leftarrow CLIST_{final} + IC_{inbetween}$

Averaging Translations and Orientations The average of a list of translations is the barycenter of all the translations. Conversely, the computation of the average of orientations is much more difficult to tackle due to the non-linearity of the orientation space. Several techniques have been implemented to solve this problem using unit quaternions. A simple method is to compute the average on each component independently. The result is then re-normalized to ensure that it is effectively a unit quaternion [Azuma and Bishop, 1994]. While this method is simple and fast to compute, the result is not accurate as it turns a non-linear problem into a linear one. The average of two quaternions q_1 and q_2 may be accurately computed using the *slerp*(ω, q_1, q_2) function (spherical linear interpolation) [Shoemake, 1985] ω being the interpolation parameter. In this case, ω is set to $1/2$. To evaluate the mean of n unit quaternions, the slerp function is then hierarchically applied on each pair of quaternions. The resulting quaternion then replaces these latter. For example, if we need to average 3 unit quaternions q_1, q_2 and q_3, we have to compute *slerp*$(1/3,$ *slerp*$(1/2, q_1, q_2), q_3)$. However, this method needs to define an order on the quaternions as it is not associative. As a consequence, the resulting average may be different depending on the order we choose. Buss and Fillmore [Buss and Fillmore, 2001] proposed to consider the problem as a weighted least squares minimization problem. While this method seems to be mathematically correct, it needs numerical iterations to find the solution and thus is subject to local minima problems. Finally, Park et al. [Park et al., 2002] introduced a global linearization method. Each quaternion is first projected onto the plane tangent to some reference quaternion using the logarithm map (see Figure 6.13). Then, the average is linearly computed on this plane. Finally, the result

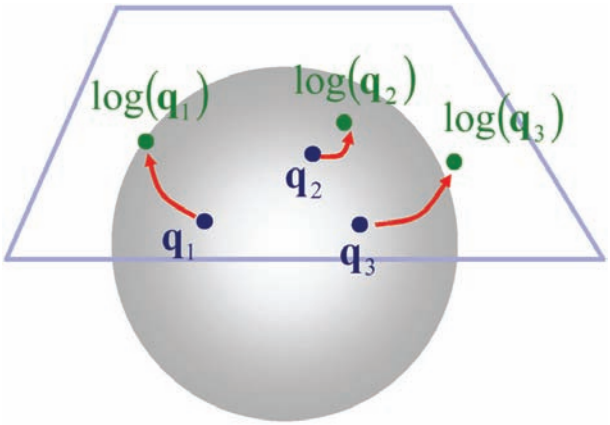

Figure 6.13: Unit quaternions averaging (Park et al. [Park et al., 2002]).

is reprojected onto the quaternion space using the exponential map. This method needs to choose a reference quaternion which is as close as possible to all the quaternions. To do so, they introduced a minimization method based on the geodesic norm. However, in the worst case, the number of quaternions they have to average is still relatively small. They can then afford to minimize an objective function to estimate a good reference quaternion to start with. In our case, the number of quaternions we want to average may be as large as the duration of the entire animation. Hence, it is computationally too expensive to used such numerical methods. Moreover, their method is mainly concerned with blending motions together. As a result, the quaternions they need to average are potentially "far" from each other. In our con-

text however, all these orientations are actually very close to each other. Indeed, we would not have detected any space constraint if it were not the case. We therefore choose to use the same method as [Park et al., 2002] but using a faster approach to estimate the reference orientation by linearly averaging the quaternions. Then the result is re-normalized to ensure that we still have a unit quaternion. This result serves as the reference orientation to compute the tangent plane to project the unit quaternions.

6.4.3.3 Line Constraints Computation

In this section, we detail our algorithm to compute the duration of line constraints given the list of constraints resulting from the previous stage. This list contains space constraints and line instantaneous constraints only. The user may define a minimal duration $min_{duration}$ for constraints he is interested in. We then rely on this parameter to decide whether a space constraint previously computed may be discarded to increase the duration of a line constraint or not.

The computation of line constraints involves three particular functions: the *equality* function (used in line 26 of Algorithm 4), the *mergeable* function (used in lines 10 and 19 of Algorithm 4) and the *merge* function (used in lines 11, 20 and 27 of Algorithm 4).

Equality Function Two line constraints are equal if they are temporally connected and if they represent the same line in space. More formally, let us consider two line constraints C_{line_1} and C_{line_2}. Then, C_{line_1} and C_{line_2} are equal if and only if:

1. $d_{Euclidean}(C_{line_1}, C_{line_2}) < \Delta_{distance}$,
2. $d_{angular}(C_{line_1}, C_{line_2}) < \Delta_{angle}$ and,
3. C_{line_1} and C_{line_2} are temporally connected.

with $d_{Euclidean}(C_{line_1}, C_{line_2})$ the minimal Euclidean distance between C_{line_1} and C_{line_2} and $d_{angular}(C_{line_1}, C_{line_2})$ the minimal angle between C_{line_1} and C_{line_2}. $\Delta_{distance}$ is a threshold determining a maximum distance to consider two lines as intersecting. Δ_{angle} is a threshold determining a maximum angle between two vectors to consider them as parallel.

Mergeable Function We consider that a line and a space constraint C_{line} and C_{space} are **mergeable** if and only if:

1. The minimal allowed duration $min_{duration}$ (set by the user) is superior to the duration of C_{space},
2. C_{line} and C_{space} are temporally connected.

In such a case, C_{space} is considered as too short and is removed during the filtering stage. In other words, it is removed if it allows to increase the duration of neighboring line constraints.

Figure 6.14 schematically describes this function: if the minimal duration (set by the animator) for a constraint is more than 2 frames and the frame tolerance is more than 2 frames then C_{line} and C_{space} are mergeable.

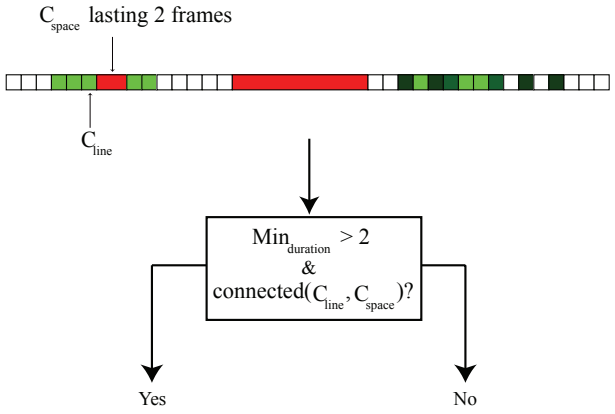

Figure 6.14: Definition of the **mergeable** function for one line and one space constraint.

Merge Function The result of the **merging** of a line constraint C_{line} and a space constraint C_{space} which are mergeable is a line constraint which continuously spans C_{line} and C_{space} with the same origin and direction as C_{line}. More formally, if C_{line} and C_{space} are respectively defined during time intervals $[a_{line}, b_{line}]$ and $[a_{space}, b_{space}]$, the result of the merging of C_{line} and C_{space} is a line constraint C'_{line} identical to C_{line} but defined during interval $[min(a_{line}, a_{space}), max(b_{line}, b_{space})]$.

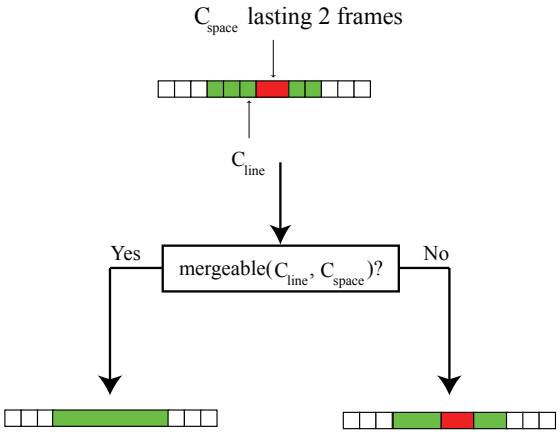

Figure 6.15: Results of the **merge** function depending on the **mergeable** function.

Figure 6.15 schematically describes this function. If C_{line} and C_{space} are mergeable, then we find a single line constraint. Otherwise, we end up with two different line constraints.

Given these three functions, we compute the line constraints using Algorithm 4. Basically, we traverse the given list of constraints. Line constraints are merged if they are temporally connected and if they are equal. Moreover, space constraints may be removed to increase the duration of line constraints when these latter are mergeable. Finally, the average line origin and direction are computed by independently averaging the directions and the origins. The directions are averaged using the technique explained in Paragraph 6.4.3.2. Note that the final direction has to be re-normalized.

Algorithm 4 Line constraints computation

```
1: CLIST_initial ← Initial list of constraints
2: CLIST_final ← ∅ /* Final list of constraints */
3: C_active ← First constraint in CLIST_initial
4: for all remaining C_current in CLIST_initial do
5:    if C_current is a space constraint then
6:       if C_active is a space constraint then
7:          CLIST_final ← CLIST_final + C_active
8:          C_active ← C_current
9:       else
10:         if mergeable(C_active, C_current) then
11:            C_active = merge(C_active, C_current)
12:         else
13:            CLIST_final ← CLIST_final + C_active
14:            C_active = C_current
15:         end if
16:      end if
17:   else
18:      if C_active is a space constraint then
19:         if mergeable(C_active, C_current) then
20:            C_active = merge(C_active, C_current)
21:         else
22:            CLIST_final ← CLIST_final + C_active
23:            C_active = C_current
24:         end if
25:      else
26:         if C_active == C_current then
27:            C_active = merge(C_active, C_current)
28:         else
29:            CLIST_final ← CLIST_final + C_active
30:            C_active = C_current
31:         end if
32:      end if
33:   end if
34: end for
```

6.4.3.4 Point Constraints Computation

In this section, we detail our algorithm to compute the duration of the point constraints given the list of constraints resulting from previous stages. It is important to note that at this point, we do not consider space constraints anymore. Indeed, space constraints which were potentially removable due to their short duration have already been merged with line constraints when possible. As a result and for clarity, in the remainder of this section, we consider that the initial list of constraints contains line constraints only.

The computation of point constraints given a list of line constraints involves two particular functions: the *mergeable* function (used in line 14 of Algorithm 5) and the *merge* function (used in line 15 of Algorithm 5). Note that the functions used in lines 6 and 7 of Algorithm 5 are related to lines and not to points.

Mergeable Function This function is similar to the one regarding line intersection. Thus, we consider that a point constraint C_{point} and a line constraint C_{line} are **mergeable** if and only if:

1. The minimal allowed duration $min_{duration}$ (set by the user) is superior to the duration of C_{line},
2. C_{point} and C_{line} are temporally connected.

Indeed, in such a case, C_{line} is too short and is removed during the filtering stage. As a consequence we may directly remove it if it allows to increase the duration of neighboring point constraints.

Merge Function This function is also very similar to the one defined for line constraints computation.

Thus, the result of the **merging** of two line constraints C_1 and C_2 which are mergeable (but not equal, or they would have been merged during the previous stage of the algorithm) is a point constraint C_{point} which continuously spans C_1 and C_2, the intersection between both lines defined by C_1 and C_2 being its position. More formally, if C_1 and C_2 are respectively defined during time intervals $[a_1, b_1]$ and $[a_2, b_2]$, the result of the merging of C_1 and C_2 is a point constraint C'_{point} defined during interval $[min(a_1, a_2), max(b_1, b_2)]$ and whose position is at the intersection of C_1 and C_2.

Moreover, the result of the **merging** of a point constraint C_{point} and a line constraint C_{line} which are mergeable is a point constraint which continuously spans C_{point} and C_{line} with the same position as C_{point}. More formally, if C_{point} and C_{line} are respectively defined during time intervals $[a_{point}, b_{point}]$ and $[a_{line}, b_{line}]$, the result of the merging of C_{point} and C_{line} is a point constraint C'_{point} defined during interval $[min(a_{point}, a_{line}), max(b_{point}, b_{line})]$.

Given these two functions, we compute the point constraints using Algorithm 5. Basically, we traverse the given list of constraints. Line constraints are merged if they are connected in time and intersect in space. Note that at this point, mergeable line constraints may not be equal as this case is handled during the previous stage. Moreover, line constraints may be removed to increase the duration of point constraints when they are mergeable.

Algorithm 5 Point constraints computation

1: $CLIST_{initial} \leftarrow$ Initial list of constraints **without considering space constraints**
2: $CLIST_{final} \leftarrow \emptyset$ **/* Final list of constraints */**
3: $C_{active} \leftarrow$ First constraint in $CLIST_{initial}$
4: **for all** remaining $C_{current}$ in $CLIST_{initial}$ **do**
5: **if** C_{active} is a line constraint **then**
6: **if** $mergeable(C_{active}, C_{current})$ **then**
7: $C_{active} = merge(C_{active}, C_{current})$
8: **else**
9: $CLIST_{final} \leftarrow CLIST_{final} + C_{active}$
10: $C_{active} = C_{current}$
11: **end if**
12: **else**
13: **/*** C_{active} **is a point constraint */**
14: **if** $mergeable(C_{active}, C_{current})$ **then**
15: $C_{active} = merge(C_{active}, C_{current})$
16: **else**
17: $CLIST_{final} \leftarrow CLIST_{final} + C_{active}$
18: $C_{active} = C_{current}$
19: **end if**
20: **end if**
21: **end for**

6.4.4 Final Filtering

The user may not be interested in all the previously detected constraints. He may instead focus on constraints lasting for a minimum amount of frames $min_{duration}$ and/or being of a specific dimension dim. This stage filters out the previous results so that constraints not meeting the user's requirements are ignored. Hence, the final list of constraints resulting from previous stages is traversed. Each constraint not meeting the requirements is discarded and removed from the list.

6.5 Experimental Results

In this section, we present some experimental results of our constraint detection algorithm. To ease the discussion, we need to introduce some terms:

1. **False positive constraint:** we consider a constraint as being a false positive if it is detected while it should not,

2. **False negative constraint:** we consider a constraint as being a false negative if it is *not* detected while it should.

The same terms are used at a finer granularity when dealing with frames that should or should not be constrained. The first sections serve to highlight important characteristics of our algorithm. In particular, we show a practical example demonstrating that the global formulation of the displacement matrices is not accurate. We then show that a dynamic estimation of the SVD-related parameters is important when a motion is composed of several different motions. Moreover, we show that a robust estimation of the SVD-related parameters greatly increases the efficiency of the algorithm when working with highly noisy motions or when the user mislabels a constraint. Finally, we show, on a wide range of motions, the accuracy of our algorithm.

6.5.1 Global versus Local Estimation

In this example, we show that the global formulation is clearly not correct and leads to results which depend on the global position of the virtual human. The initial motion is the one shown in Figure 6.5. For clarity purposes, we essentially focus our discussion on detecting time intervals during which the left foot is planted as these results are similar for the virtual character's right foot.

The initial animation is composed of 350 frames (14 seconds). It is manually labeled to compare the results of our algorithm. The motion contains 4 constraints related to the left foot during the walking stages and 6 constraints during the running stage. The results shown in Figure 6.16 have been obtained by specifying a template constraint of dimension three between frames 34 and 40. This template constraint corresponds to a footplant during

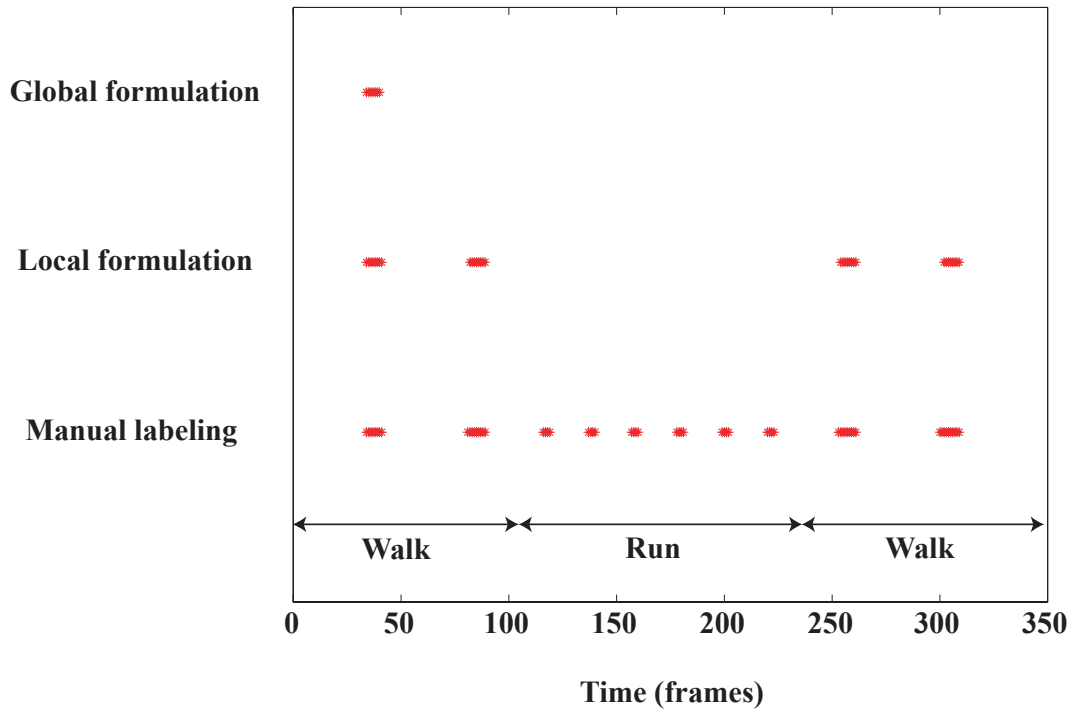

Figure 6.16: Constraint detection using global and local formulation. Space constraints are indicated using a red cross at each frame.

the walking stage of the animation. As explained in Section 6.4.1, the global formulation of the displacement matrices leads to results that are directly dependent on the global location of the animation. The farther the virtual character, the higher the residual error. Thus, as the animation starts at location $(0, 0, 0)$ and ends at location $(17, 0, 0)$, the virtual character is going away from the origin of the world. As a consequence, as the specified template constraint occurs at the beginning of the animation, the corresponding estimated SVD-related parameters are low compared to those associated to the end of the animation (even though the animation is strictly the same). For this reason, if we use the global formulation of the displacement matrices, we only detect the first footplant. Subsequent footplants require higher value for the SVD-related parameters and thus are not detected. On the other hand, if we use the local formulation of the displacement matrices, the SVD-related parameters are no longer dependent on the global location of the animation. We then detect all the footplants occurring during the walking stages.

Regarding the duration of the detected constraints, our algorithm returns, for the whole animation, 4 false positive frames which slightly enlarge the actual constraints. This means that in comparison to the animator labeling, each detected constraint is accurate more or less one frame. However, who is *objectively* right? Our algorithm or the user? This question is really hard to answer especially when dealing with noisy motion capture data: footplants are rarely sharp and may lead to variations in the labeling even among animators.

Finally we may notice that, compared to the manual labeling, our algorithm returns six false negative constraints occurring during the running stage. This is overcome using the dynamic estimation of the SVD-related parameters as shown in the next section.

6.5.2 Static versus Dynamic Estimation of the SVD-related parameters

In this section, we use the same motion as above. In most constraint detection methods, thresholds are statically fixed. For example, methods using the position and the velocity of the feet to determine whether they can be considered as stationary or not use predefined minimal values of the height and the velocity of the foot beneath which they detect a footplant. The direct consequence of such methods is that the thresholds have to be reevaluated for each new motion. Indeed, depending on the noise present in the initial animation or even depending on the type of motion, these latter may vary. Even worse, a single movement may contain different types of animation. Hence, static thresholds can be accurate for a specific part of the animation and totally inaccurate for the others. Figure 6.17 clearly demonstrates this statement. In this example, we first specify a template constraint during the walking stage of the animation between frames 34 and 40 (see case A). As previously noted, no space constraint occurring during the running stage is detected. Conversely, if the template constraint is specified during the running stage of the animation between frames 156 and 160 (see case B), we then end up with too many false positive frames. Indeed, space constraints occurring during the walking stages are detected but their duration is clearly inaccurate. While the first detected space constraint is supposed to last from frame 34 to frame 41 according to the manual labeling, it actually lasts from frame 25 to frame 44. This result is not acceptable as it corresponds to 12 false positive frames.

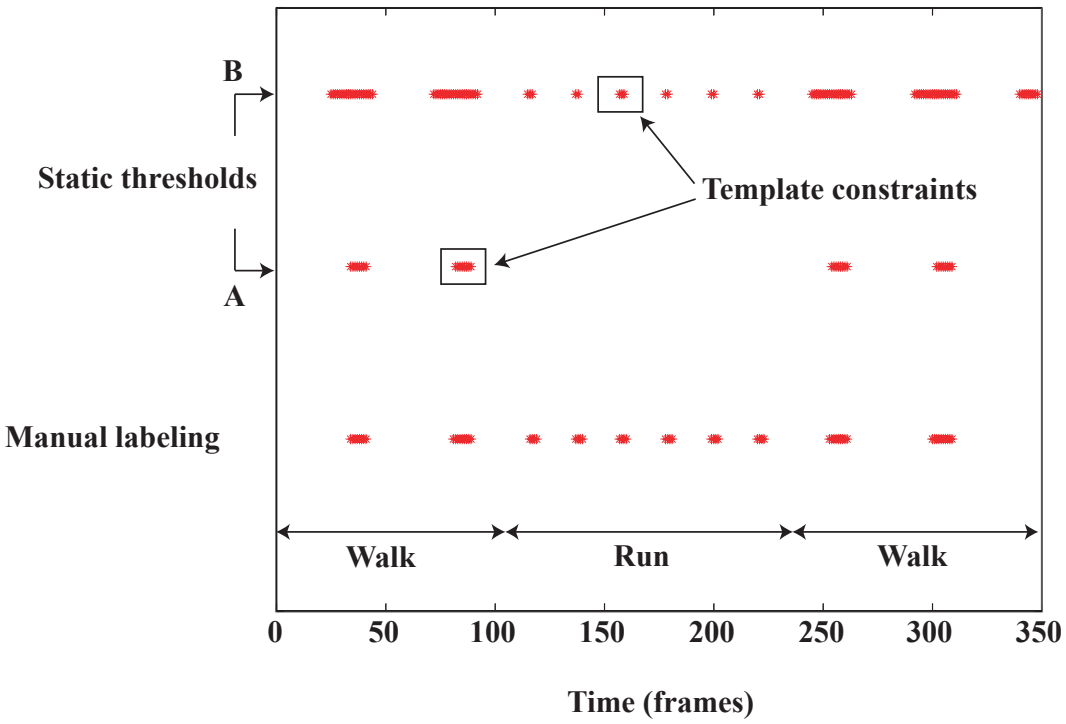

Figure 6.17: Constraint detection using static estimation of the SVD-related parameters

As a result, in our framework, the SVD-related parameters are dynamically estimated using splines updated each time the user adds a template constraint. Using the same template constraints as for the example of Figure 6.17, the results are shown in Figure 6.18. We first detect space constraints occurring during the walking stages (case A). Then, the SVD-related parameters are refined by adding a template constraint during the running stage (case B). In this case, all the space constraints are accurately detected. We must note however that space constraints occurring during the transitions from walking to running and later from running to walking are not entirely satisfactory: their duration is not long enough. This is easily explainable by the fact that these transitions are obtained using linear blending, introducing much more slidings. This problem can be overcome using additional template constraints or more simply, by editing their duration.

6.5.3 Naive versus LMedS method

In this section we show the advantages of using a robust estimation of the SVD-related parameters based on the LMedS method. The initial motion is a “sitting on a stool” animation. It is composed of 400 frames (16 seconds). For clarity purposes, we again only consider space constraints related to the left foot. According to the manual labeling, the initial motion contains, in particular, a constraint between frames 50 and 69. However, the motion is very noisy and mislabeling may occur. In our example, consider that the user is mistaken and thus specifies a constraint between frames 50 and 75, which consists in a relative error of 6 frames. The results are shown in Figure 6.19. While the algorithm should detect two constraints between frames 50 and 206, it actually detects only one. Moreover, the dura-

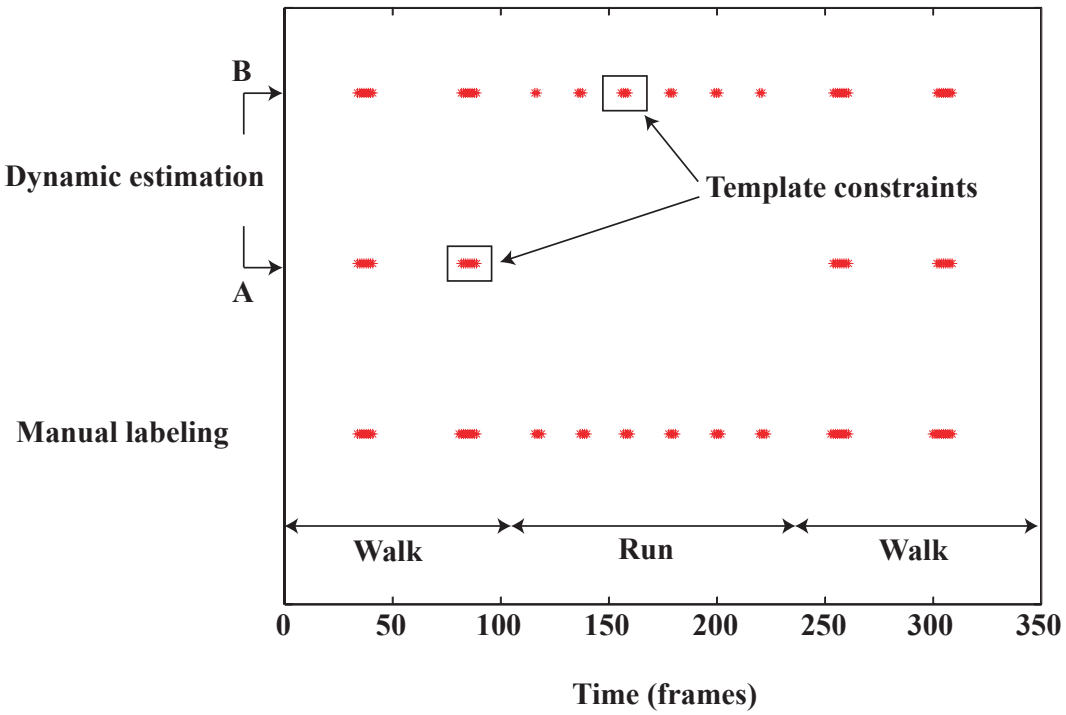

Figure 6.18: Constraint detection using dynamic estimation of the SVD-related parameters

tion of most detected constraints is not satisfactory. In particular, the next to last detected constraint lasts for too many frames. Conversely, using our robust estimation based on the LMedS method gives accurate results. We effectively detect the right number of constraints. Moreover, their duration is correctly detected: the difference between manual labeling and our algorithm is less than 5 frames (see next to last and last constraints) using one template constraint only.

6.5.4 Walking-Running-Walking Motion

In this section, we graphically show the results of our algorithm. Figure 6.20 shows the final results of the space constraints detection related to the left foot. Figure 6.21 focuses on one space constraint to highlight that even though the motion contains footslidings, our method accurately detects corresponding constraints. All 10 space constraints related to the left foot are detected using only two template constraints: one during a walking stage and the other one during the running stage.

6.5.5 Walking Around Motion

In this example, the virtual character walks around during 400 frames (16 seconds). We detect the space constraints associated to both feet. We only need to specify 2 template constraints: one for the right foot and one for the left one. The results are shown in Figure 6.22. All the constraints are correctly detected. However, some are ambiguous. When the

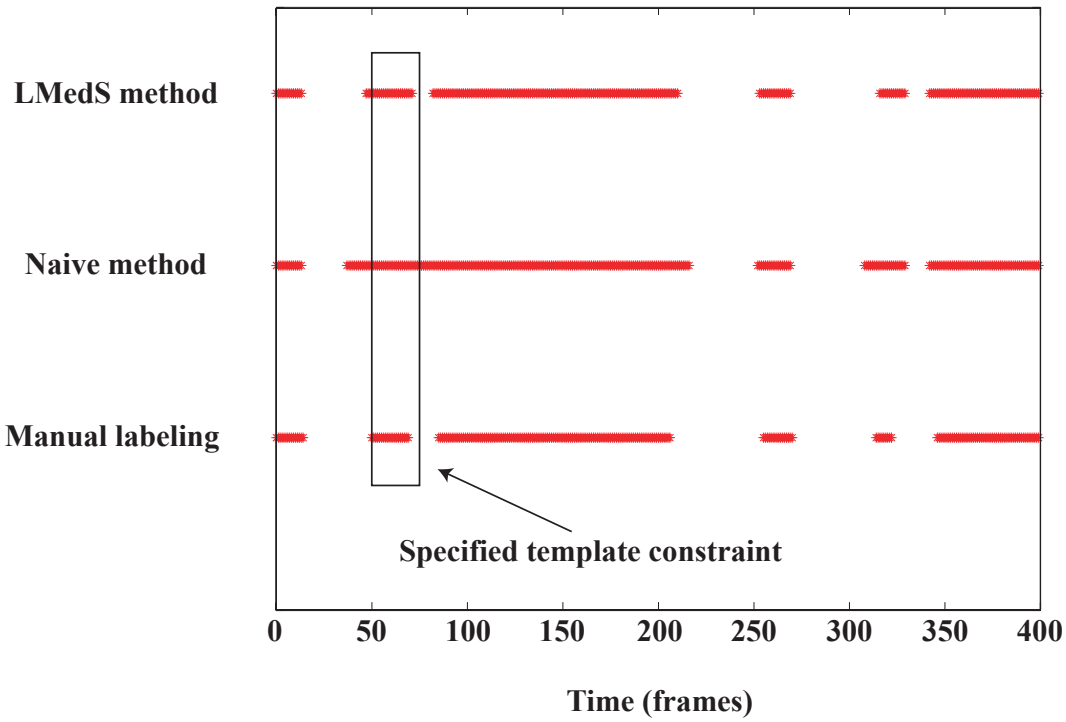

Figure 6.19: Constraint detection using naive and LMedS methods

character turns, footplants are quite difficult to identify even for the animator. In this case, it may happen that the detected constraints are too short or even sliced while they *maybe* should not.

6.5.6 Sitting on a Stool

In the example of Figure 6.23, we want to detect constraints associated with the body parts attached to the ankles. Two template constraints have been specified (one for each body part) to help the algorithm estimate the SVD-related parameters. Our algorithm is able to detect all the constraints related to the ankles. However, some footplants are very complex. For example, while the virtual character is turning around the stool, the foot is on the ground but rotating around the toe. This type of footplant is difficult to accurately detect as the foot is not stationary but rotates around a point instead.

In the example of Figure 6.24, we applied our algorithm to detect constraints related to the hips. This is very useful to reposition the stool when the character is not correctly sitting on it for example.

6.5.7 Line Constraints Detection

In this section, we present results of line constraints detection. So far, we have assimilated footplants as being only related to the time interval during which the body part attached to the ankle remains stationary. However, one may want to define a footplant more precisely

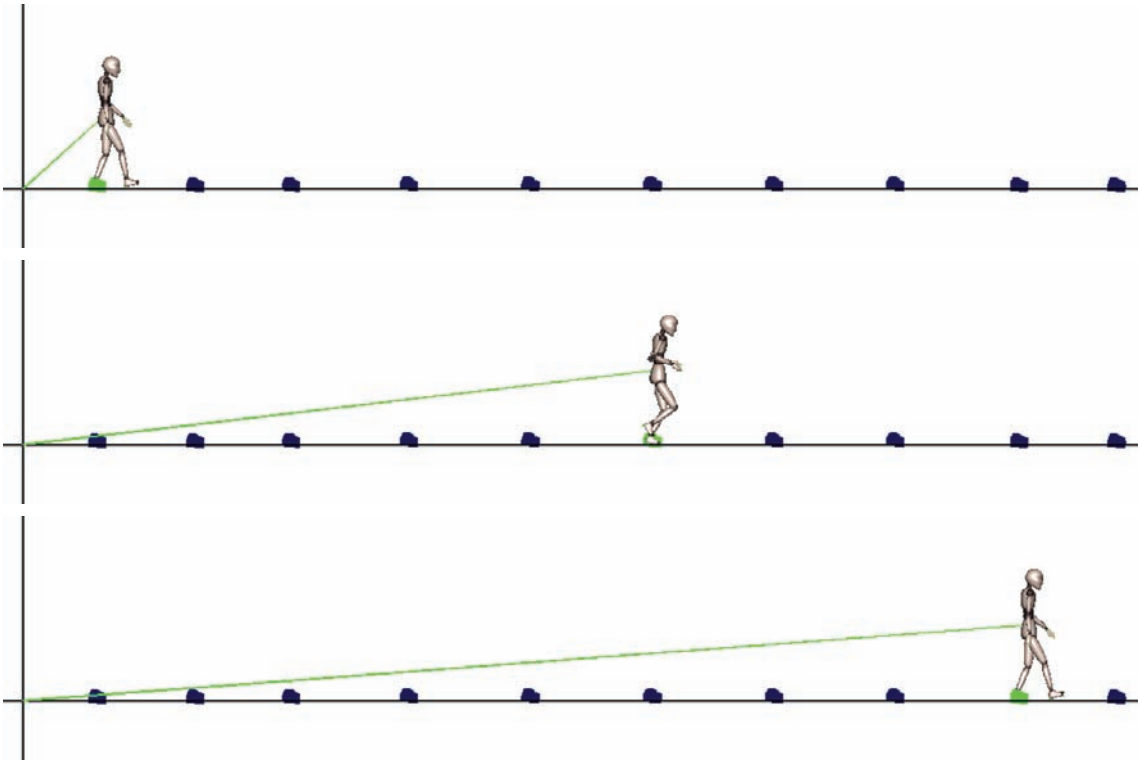

Figure 6.20: Final space constraints detected related to the left foot.

using the following sequence:

1. **A line constraint from Heel Strike to Toe Strike:** the heel is planted on the ground while the foot is rotating around an axis passing approximately through it,

2. **A space constraint from Toe Strike to Heel Off:** the foot is planted on the ground and remains stationary.

The initial motion is a walking animation composed of 200 frames (8 seconds). We first begin by specifying a template constraint between frames 80 and 92 to detect the time intervals during which the foot stays stationary on the ground. Afterward, we manually define a line constraint by adding a template constraint just before a footplant (between frames 74 and 80). The results of the constraint detection are shown in Figure 6.25. Our algorithm precisely detects all the constraints as the overall false positive/negative frames is 1 for the entire motion. This happens during the first footplant. While the line constraint should end exactly when the shape constraint starts, it actually ends 1 frame earlier. However, these results are still correct as the animator only needs, in the worst case, to edit one constraint instead of having to label 4 space constraints and 4 line constraints. Note also that due to the noisy nature of the initial motion, slight differences may occur between line constraints. For example, the first one is clearly different than the others.

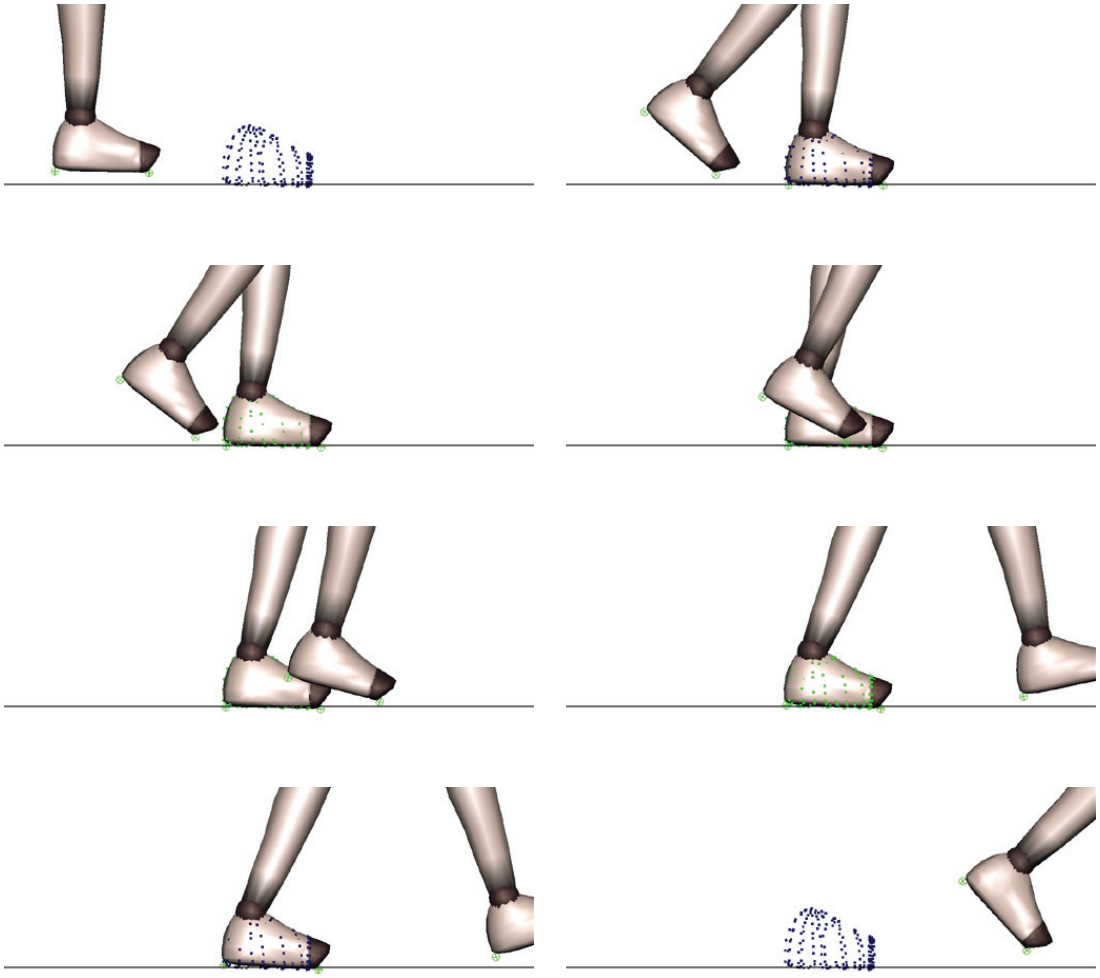

Figure 6.21: Detection of a constraint related to the body part attached to the ankle (from upper left to bottom right). The points indicates the vertices associated to the constrained body part. **Upper and Bottom Rows:** The constraint is *not* active. **Middle Rows:** The constraint is active.

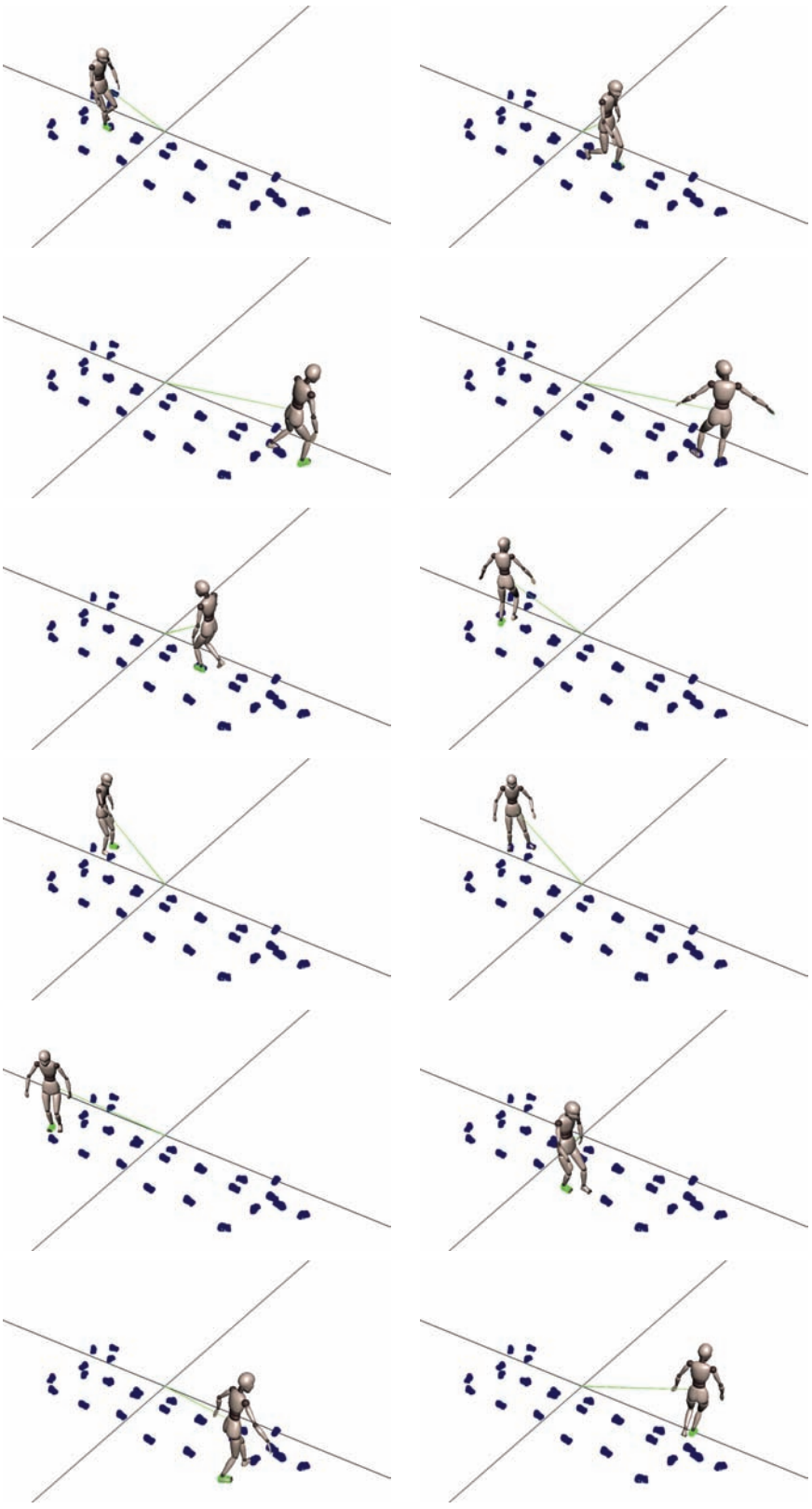

Figure 6.22: Walking around animation. The constraint detection is applied to the body parts attached to the ankles. The constraints are displayed in green when they are active and in blue otherwise. **Second and Forth Rows:** The footplant is hard to detect as the foot is rotating.

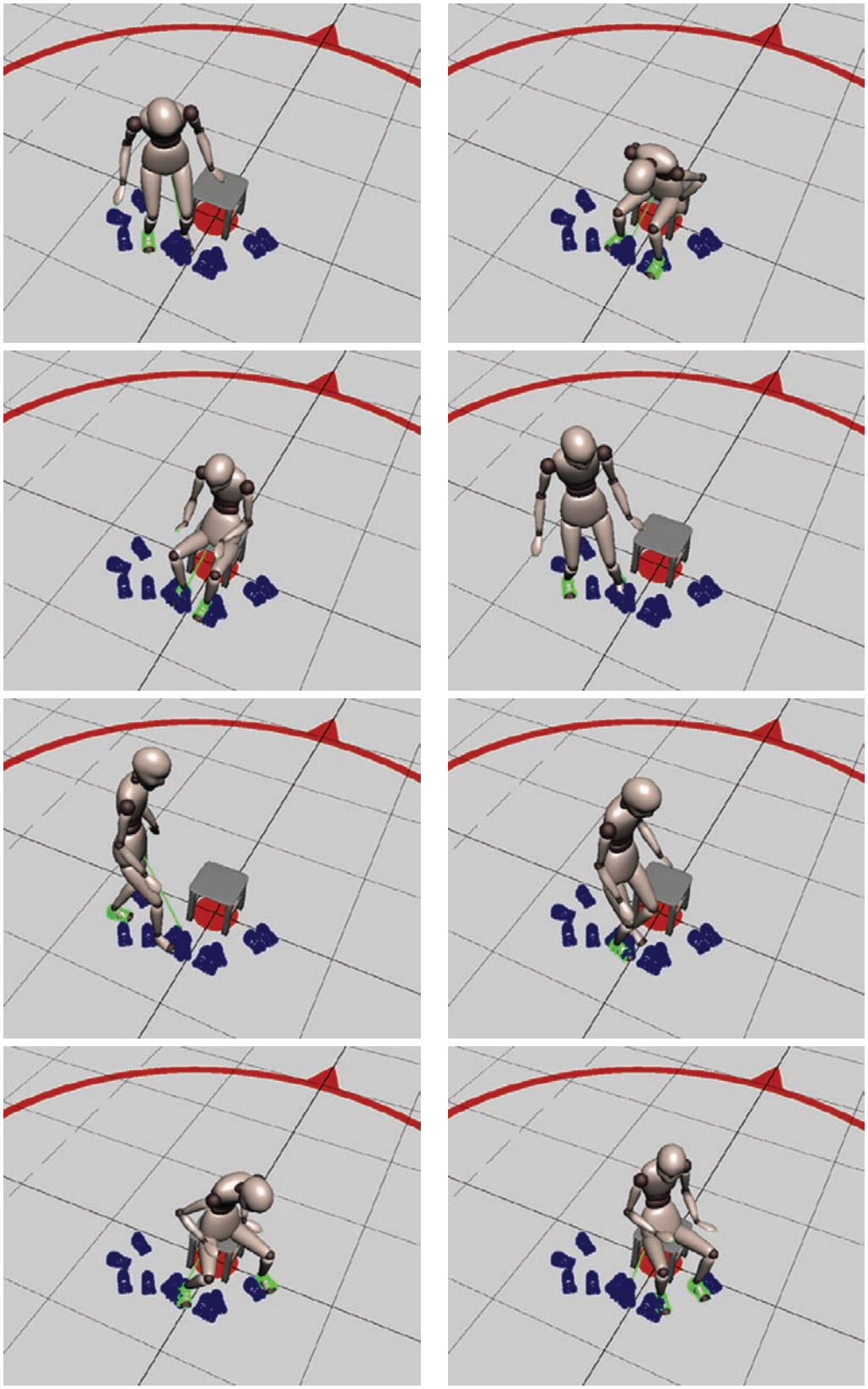

Figure 6.23: Constraint detection applied on body parts attached to the ankles. The constraints are displayed in green when they are active and in blue otherwise. The animation goes from top left to bottom right (breadth first).

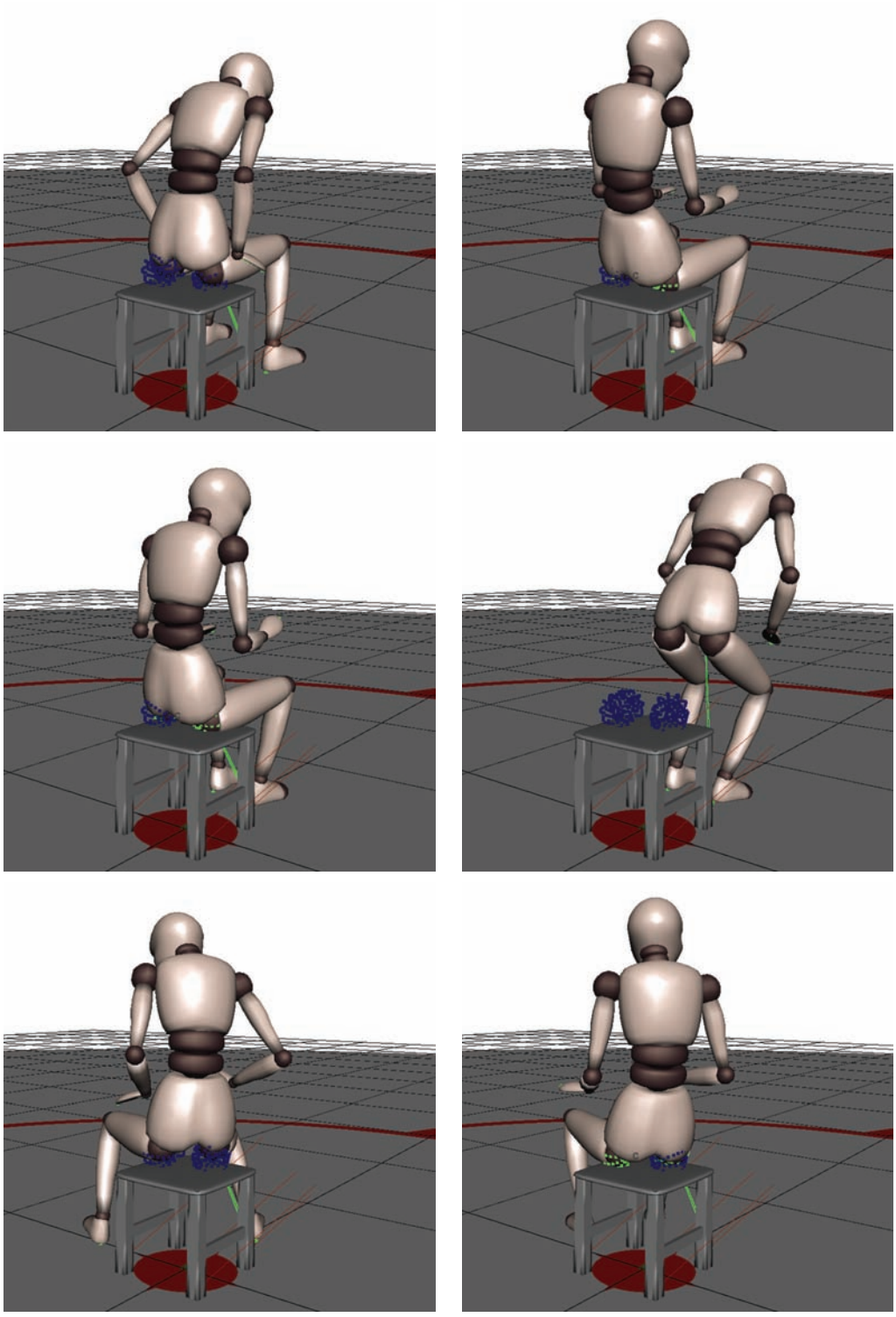

Figure 6.24: Constraint detection applied on body parts attached to the hips. The constraints are displayed in green when they are active and in blue otherwise. **Top Row:** Constraints 1 second before being active (left) and when they are just activated (right). **Middle Row:** Constraints just before being deactivated (left) and 1 second later (right). **Bottom Row:** Constraints 1 second before being active (left) and when they are just activated (right).

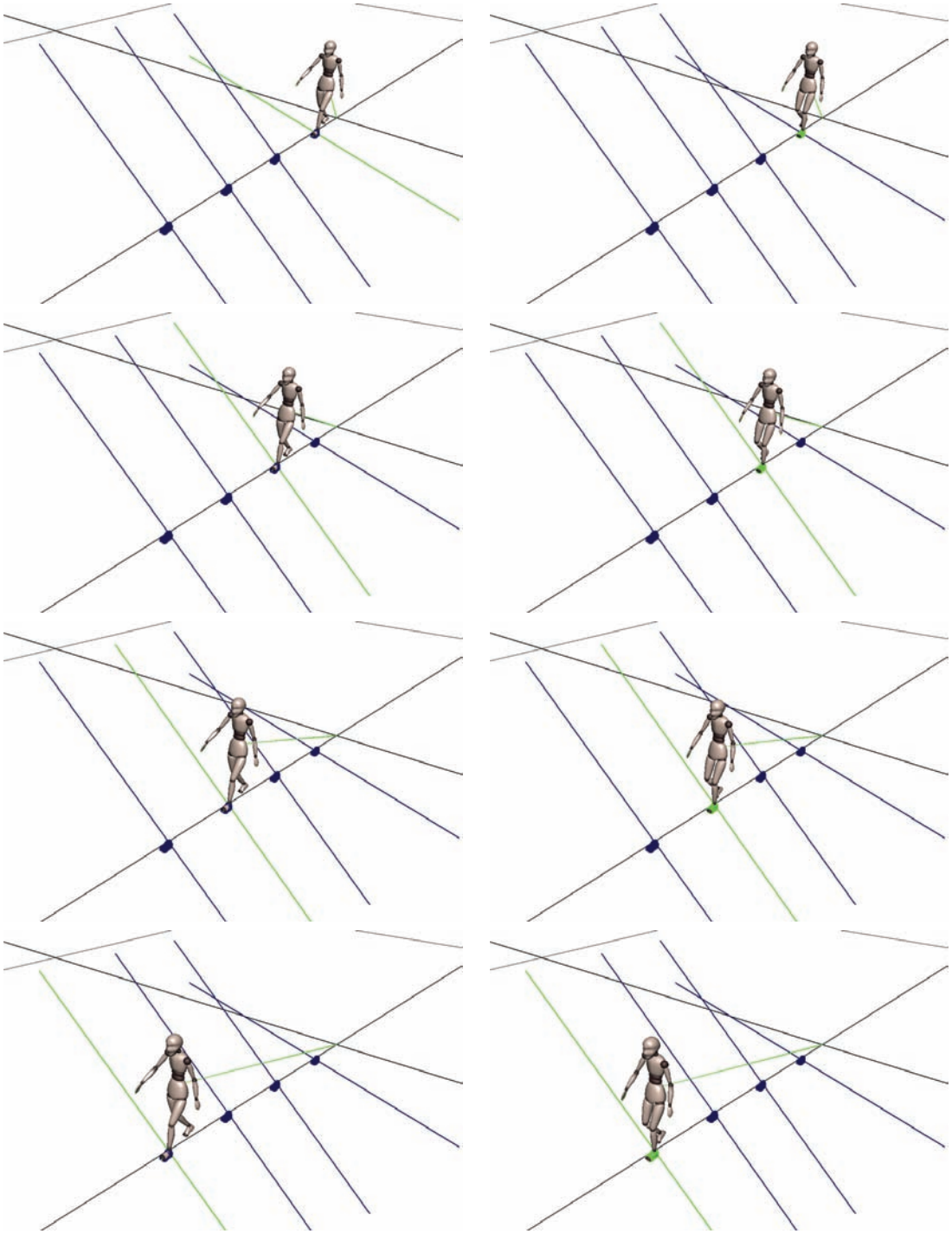

Figure 6.25: Line constraint detection applied on body parts attached to the left ankle. The constraints are displayed in green when they are active and in blue otherwise. **Left Column:** The line constraints are active. **Right Column:** The space constraints become active.

6.5.8 Point Constraint Detection

Even though our algorithm can handle point constraint detection, its benefit in a context of motion editing is moderate for several reasons:

- these constraints rarely occur in motion capture animations. Thus, it is often simpler to specify them by hand when needed.
- While space constraints are easy to identify, point constraints require more attention in general. Indeed, it is quite simple to estimate whether an object is stationary or not but more complex to decide whether it is rotating around a point or not.

However, the detection of such constraints may be of interest for many other applications such as motion indexation for example. The constraint detection may be used to automatically extract motion features from animations to ease subsequent retrieval from large databases. These motion features may also be used to reconstruct a motion given a set of constraints for example. We therefore show, in this section, that our algorithm is effective in detecting point constraints. We have tested point constraint detection on a wide variety of synthetic animations. We have also tested our algorithm on motion capture animations. We show, in particular, that even though point constraint detection becomes difficult using noisy data, it may be of interest to retrieve some semantical information from a given animation.

6.5.8.1 Dice

The initial animation contains 130 frames (approximately 5 seconds). It represents two dice rolling on a dice carpet. We detected the point constraints on both dice. The results are shown in Figure 6.26. Note that as we are working with synthetic animations, the noise is not as large as in motion capture animations and we can use minimal thresholds handling numerical errors.

6.5.8.2 Desk Lamp

The initial animation contains 180 frames (approximately 7 seconds). It represents a desk lamp looking at a ball before looking toward the camera. We detected the point constraints associated to the lampshade. The results are shown in Figure 6.27. Our algorithm detects two point constraints. The first one corresponds to the time interval during which the lamp is looking at the ball. The second one is located at the base of the lampshade when the lamp is turning to look at the camera. It actually corresponds to a joint of the underlying hierarchy.

6.5.8.3 Walking Around Motion

In this section, we reuse the example shown in Figure 6.22. As previously stated, when the character is turning, footplants become difficult to identify. It may then be of interest to identify such events. The results are shown in Figure 6.28. As expected, our algorithm detects a point constraint each time the virtual character sharply turns while the left foot is

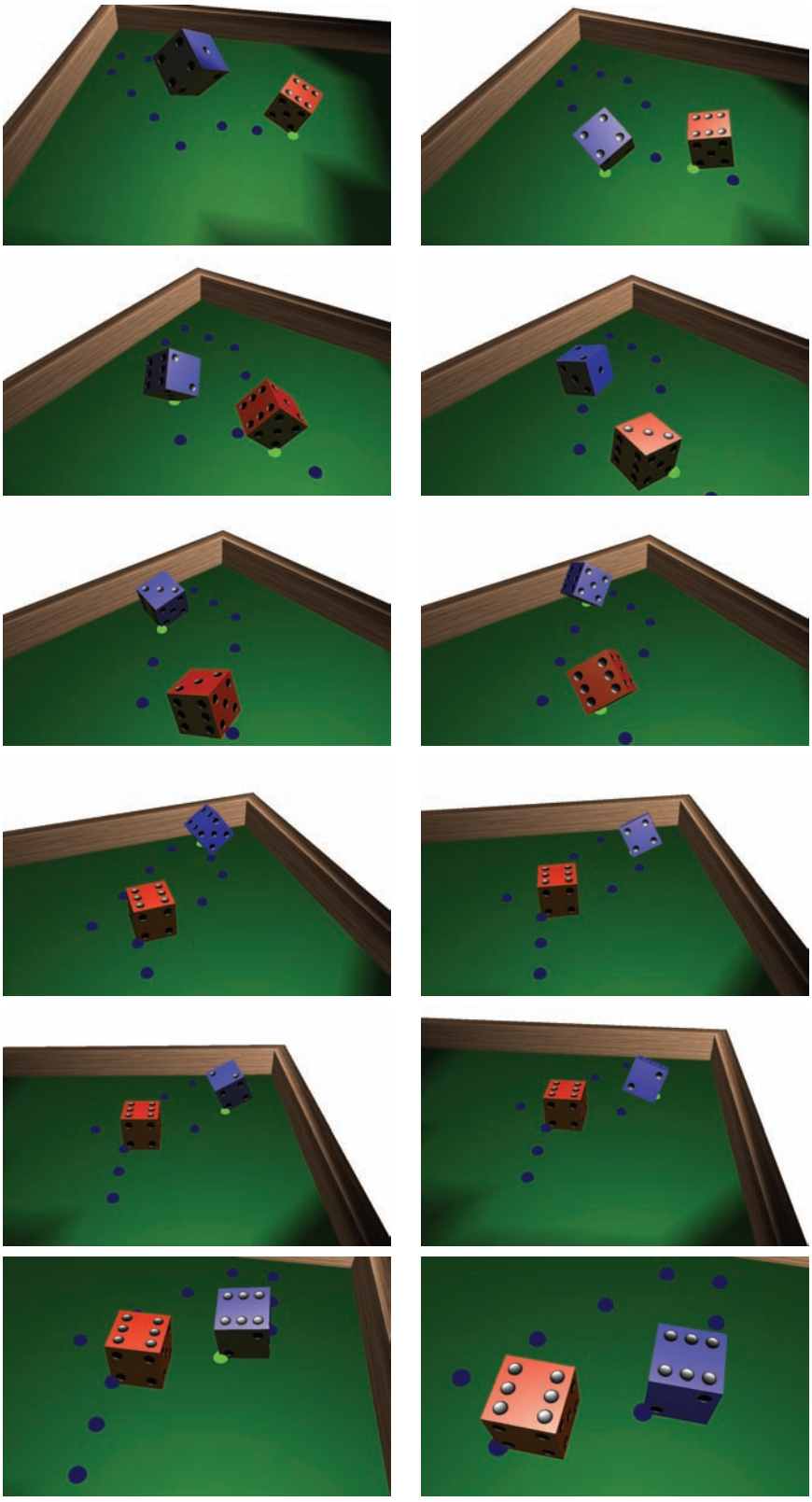

Figure 6.26: Point constraint detection applied on two dice rolling on a carpet. The constraints are displayed in green when they are active and in blue otherwise.

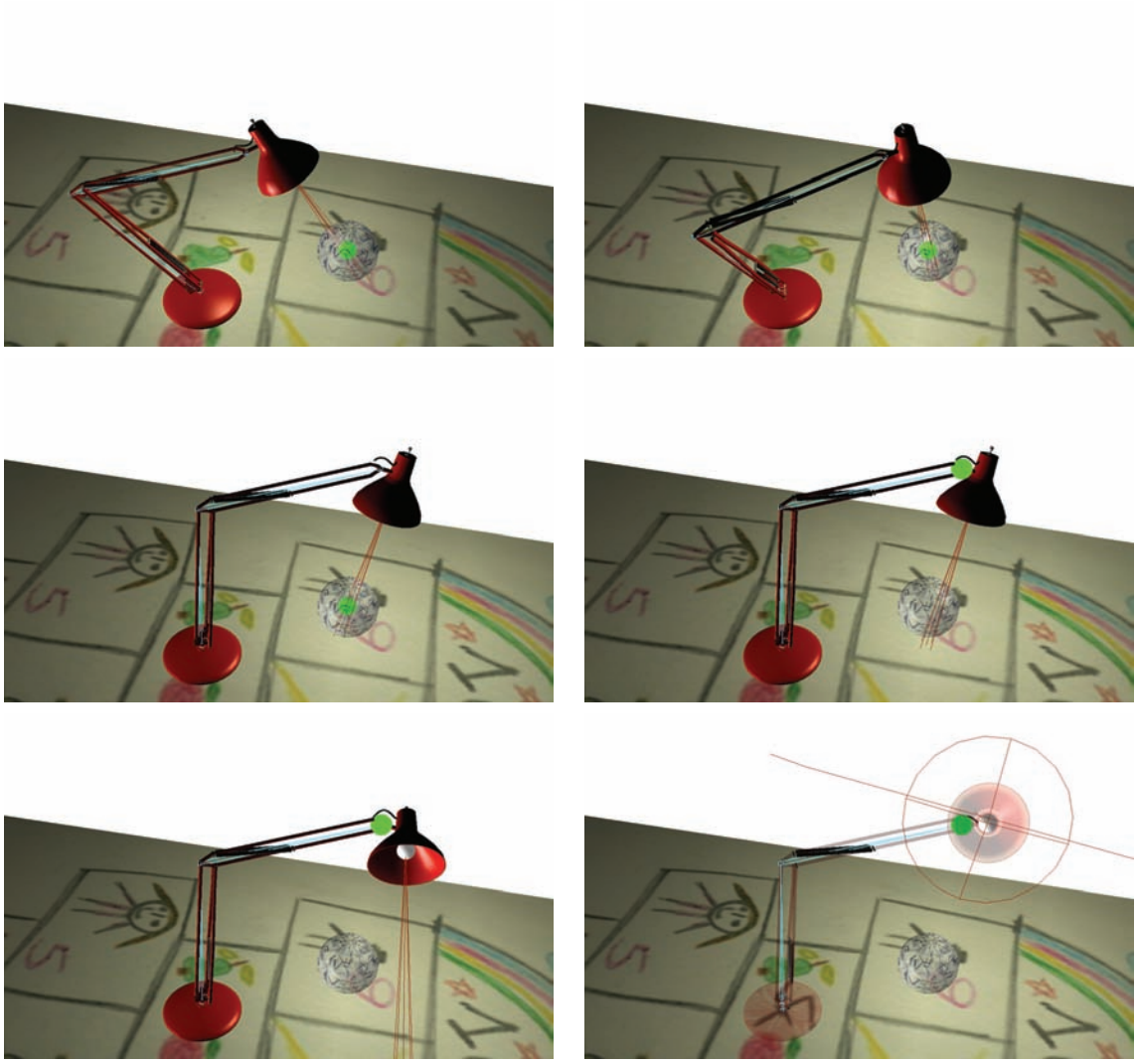

Figure 6.27: Point constraint detection applied on the shade of a desk lamp. The desk lamp is looking at a ball before looking toward the camera. The constraints are only displayed when active.

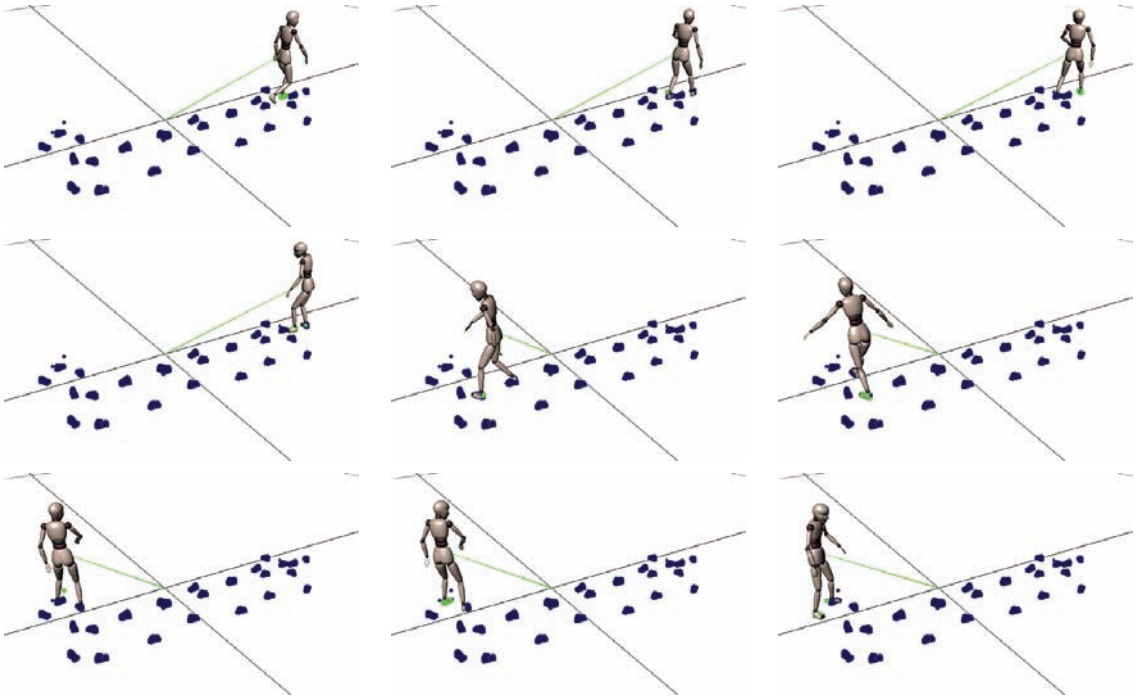

Figure 6.28: Walking around animation. The constraint detection is applied to the body parts attached to the ankles. The constraints are displayed in green when they are active and in blue otherwise. In this case, we also detect point constraints.

planted on the ground even though the animation is noisy. These point constraints may then serve to precisely annotate "turning footplants" for example.

6.5.9 Computational Cost Consideration

The detection algorithm may be divided into two parts when considering the computational cost: the Singular Value Decomposition while computing the instantaneous constraints and the merging stages. Regarding the walking-running-walking motion example, it took 11.3 ms to compute the instantaneous constraints and 4.4 ms to compute the final set of constraints. The results are similar for the walking around motion example, it took 13 ms to compute the instantaneous constraints and 4.2 ms to compute the final set of constraints. Additionally, it took 8 ms to compute the instantaneous constraints and 2.9 ms to compute the final set of constraints. Our algorithm is therefore totally suited for interactive applications as animators are able to perform several constraints detection by changing requirements (i.e. type of needed constraints, minimal duration...) with results computed at interactive rates.

6.6 Discussion and Conclusion

In this chapter, we have presented a method which helps the animator label the motion of any type of object in a scene. We have presented a robust semi-automatic method detecting geometric constraints in noisy motion-capture data. We have also demonstrated that the local formulation of the displacement matrices is more accurate than the global formulation. Given a small set of template constraints (specified by the animator) we automatically estimate key parameters (the SVD-related parameters) characterizing the noise in the data. We have detailed a robust approach to detect instantaneous constraints to discard potential outliers when estimating the SVD-related parameters. Furthermore, we have shown that a dynamic estimation of the SVD-related parameters is much more accurate in cases where the motion capture animation contains several types of motions. It is important to emphasize that animators actively participate in the geometric constraint detection process, as the concept of the "importance" of constraints is difficult to determine automatically. In [Shin et al., 2001], Shin et al. used some heuristics to estimate the importance of some constraints (end-effectors position, joint angles...). However, this method is dedicated to motion retargetting and could only be adapted with difficulty to a general motion editing process. The final result is always left to the appreciation of the animator who confirms, adjusts and/or deletes constraints depending on their subjective "importance". Finally, we have shown that our method is efficient to detect constraints related to complex articulation figures such as virtual humans. These constraints may then be used to help the animator edit the initial animation.

CHAPTER 7

Conclusion

In this thesis, we have proposed a motion deformation framework offering the user a way to specify prioritized constraints to deform an initial motion capture animation. Moreover, we have provided the user with a semi-automatic algorithm to detect important constraints in motion capture animations. In the following sections, we briefly summarize our contributions as well as future work.

7.1 Contributions

In the next sections, we present the main contributions related to the important parts of this thesis.

Designing Postures Using Inverse Kinematics

We have presented our IK solver based on the work of Baerlocher [Baerlocher, 2001], a numerical resolution framework capable of solving multiple tasks simultaneously, and which supports both task-priority and weighting strategies for the resolution of conflicts.

We have then presented an *HAnim-compliant* application providing different kinds of features such as body parts position control, CoM position control, joint weighting, joint recruiting level, etc.

Additionally, we have shown that the exponential map parameterization is more appropriate than Euler angles to represent complex articulations. While it is more complex than the Euler angles parameterization, its performances in terms of final computation time and

iterations number are equivalent. Moreover, exponential maps provide an intuitive way to represent joint limits and is singularity-free for well-chosen initial configurations.

Thanks to this test-platform, we have performed benchmarks and comparisons on several key parameters of the IK solver. Among others, we have compared the mutual influence of the integration step and the damping factor. No general equation relating the integration step, the damping factor and the convergence behavior seems to be pertinent. Indeed, the convergence behavior is closely related to the considered hierarchy of joints and to the starting configuration. However, we have experienced that a damping factor of ten and an integration step of one work well for the vast majority of examples we have tested.

Motion Deformation Constraints Definition and Design

We have presented important constraints integrated in our motion editing framework. These constraints may then be used to modify an initial animation so that it fits the animator's requirements.

In particular, we have introduced a versatile class of constraints: the *shape-constraints*. The animator is now able to design a wide range of spatial trajectories. Moreover, these constraints may contain stationary points breaking (or not) the continuity of the trajectory. Furthermore, these constraints can be expressed in a reference frame allowing relative constraints between joints, to shift a joint position or to define an absolute trajectory.

Additionally, we have presented a class of constraints dedicated to the adjustment of the position and/or orientation of footplants. Finally, we have presented a simple approach to adjust unbalanced postures by controlling the position of the CoM thanks to inverse kinetics. This approach offers the animator the possibility to improve the overall quality of the final animation.

Prioritized Motion Deformation

We have presented an interactive method for adding significant changes to an animation. Our framework improves classical motion editing techniques, as animators can add large deformations without ending up with unbalanced results. Moreover, the priority concept greatly helps when animators need to arbitrate conflicting constraints. Our algorithm allows to assign a priority to each constraint. This priority is used to arbitrate conflicts between constraints. Our scheme ensures that high-priority constraints won't be disturbed by low-priority ones. Furthermore, we have proposed a simple and efficient algorithm to avoid generating discontinuous end-effectors trajectories while adding new constraints. Finally, while we have mainly focused our discussion on motion deformation, our method is also well-suited to deal with retargetting problems.

Geometric Constraint Detection for Motion Capture Animation

We have presented a method which helps the animator label the motion of any type of object in a scene. We have presented a robust semi-automatic method detecting geometric con-

straints in noisy motion-capture data. We have also demonstrated that the local formulation of the displacement matrices is more accurate than the global formulation. Given a small set of template constraints (specified by the animator) we automatically estimate key parameters (the SVD-related parameters) characterizing the noise in the data. We have detailed a robust approach to detect instantaneous constraints to discard potential outliers when estimating the SVD-related parameters. Furthermore, we have shown that a dynamic estimation of the SVD-related parameters is much more accurate in cases where the motion capture animation contains several types of motions. Finally, we have shown that our method is efficient to detect constraints related to complex articulation figures such as virtual humans. These constraints may then be used to help the animator edit the initial animation.

7.2 Future Work

To conclude, we suggest in the next sections, some directions for future research.

Designing Postures Using Inverse Kinematics

The main drawback of numerical IK is its ability to produce non-realistic postures. Developing an accurate model of joints may then be of interest to constrain the final postures a little more and to ensure that they are not only feasible but also that they are the most plausible. While our model includes joint limits, it is clearly not sufficient. A model taking joint coupling into account would certainly enhance results. It would also be useful to introduce biomechanical parameters such as maximum joint torques or moving axis of rotation for articulations such as the knee for example.

Prioritized Motion Deformation

In our motion deformation algorithm, the consistency of the whole set of constraints is not evaluated. As a result, the final animation as "asked by the animator" may be erratic. Developing an algorithm which would be able to advise the animators when they are potentially wrong would be an important advantage over previous methods.

Geometric Constraint Detection for Motion Capture Animation

Finally, we think that the automatic generation of constraints would speed up the entire process of motion editing. Using our geometric constraints detection algorithm, it could be possible to automatically generate constraints to edit the motion. While this problem might seem easy at first sight, it is not. Indeed, if we consider footplant constraints for example, it is not only important to know the periods of time during which the feet stay stationary. It is also important to know how it moves *before* and *after* the footplant. Indeed, footplants are quite different depending on whether the virtual human is walking, running or climbing stairs for example.

APPENDIX A

Mathematical Demonstrations Related to Constraint Detection

A.1 Global versus Local Displacement Matrix Formulations

As previously detailed in Section 6.4.1, we want to solve the following equation:

$$(\boldsymbol{D_i} - \boldsymbol{I_4})\hat{\boldsymbol{p}} = 0 \tag{A.1}$$

As $\boldsymbol{D_i}$ is a rigid transformation, it can be rewritten as:

$$\boldsymbol{D_i} = \begin{bmatrix} \boldsymbol{R_i} & \boldsymbol{t_i} \\ \boldsymbol{0_3} & 1 \end{bmatrix} \tag{A.2}$$

with $\boldsymbol{R_i}$ and $\boldsymbol{t_i}$ respectively the rotational and translational components of $\boldsymbol{D_i}$. We then reformulate Equation A.1 as follows:

$$(\boldsymbol{R_i} - \boldsymbol{I_3})\boldsymbol{p} + \boldsymbol{t_i} = 0 \tag{A.3}$$

$$\boldsymbol{A}\boldsymbol{p} = -\boldsymbol{t_i} \tag{A.4}$$

Using a singular value decomposition, we can express matrix $\boldsymbol{A}$ as:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{A.5}$$

with $\boldsymbol{U}$ and $\boldsymbol{V}$ being 3×3 orthogonal matrices and $\boldsymbol{\Sigma}$ a 3×3 diagonal matrix [Press et al., 1992]. Matrix $\boldsymbol{\Sigma}$ is a diagonal matrix containing the 3 *singular values* $\sigma_{i=1,2,3}$ (with $\sigma_1 > \sigma_2 > \sigma_3$) of matrix $\boldsymbol{A}$.

Suppose now that all the sigmas in matrix $\boldsymbol{\Sigma}$ are small enough to be considered as null (we then have a constraint of dimension 3). Then, the particular solution $\boldsymbol{p_{particular}}$ corresponding to Equation A.4 is:

$$\begin{aligned} \boldsymbol{p_{particular}} &= -\boldsymbol{A}^{-1}\boldsymbol{t_i} \\ &= -\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T\boldsymbol{t_i} \\ &= -\boldsymbol{V}\boldsymbol{0_3}\boldsymbol{U}^T\boldsymbol{t_i} \\ &= [\boldsymbol{0},\boldsymbol{0},\boldsymbol{0}]^T \end{aligned}$$

The residual error is then defined as $\epsilon = \|\boldsymbol{A}\boldsymbol{p_{particular}} + \boldsymbol{t_i}\| = \boldsymbol{t_i}$.

Moreover, let $\boldsymbol{W}_i$ the matrix that transforms at frame i, a point $\boldsymbol{p}$ expressed in the $\boldsymbol{O}$ local coordinate system to $\boldsymbol{p_i}$ expressed in the world coordinate system. $\boldsymbol{W_i}$ can be decomposed as:

$$\boldsymbol{W_i} = \begin{bmatrix} \boldsymbol{R_{W_i}} & \boldsymbol{t_{W_i}} \\ \boldsymbol{0_3} & 1 \end{bmatrix} \tag{A.6}$$

In next section, we demonstrate that using a global formulation, the residual error is modified by translation: the same animation provides different residual errors depending on its global position in the scene. In Section A.1.2 we show that the local formulation is more efficient as it gives residual errors that are independent of the animation global position.

A.1.1 Global Formulation of Residual Error

As previously stated in Section 6.4.1, the displacement matrix $\boldsymbol{D_i}$ in this case is expressed as:

$$\boldsymbol{D_i} = \boldsymbol{W_{i+1}}\boldsymbol{W_i}^{-1} \tag{A.7}$$

Using Equation A.6, we can rewrite $\boldsymbol{D_i}$ as:

$$\boldsymbol{D_i} = \begin{bmatrix} \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T & \boldsymbol{t_{W_{i+1}}} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T\boldsymbol{t_{W_i}} \\ \boldsymbol{0_3} & 1 \end{bmatrix} \tag{A.8}$$

The residual error is then defined as:

$$\epsilon = \boldsymbol{t_i} = \boldsymbol{t_{W_{i+1}}} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T\boldsymbol{t_{W_i}} \tag{A.9}$$

At that point, we add a translation component to the $\boldsymbol{W_i}$ and $\boldsymbol{W_{i+1}}$ matrices. In other words:

$$\begin{aligned} \boldsymbol{t_{W_i}} &\rightarrow \boldsymbol{t_{W_i}} + \boldsymbol{t_\Delta} \\ \boldsymbol{t_{W_{i+1}}} &\rightarrow \boldsymbol{t_{W_{i+1}}} + \boldsymbol{t_\Delta} \end{aligned}$$

Then, if $\|\boldsymbol{t_\Delta}\|$ tends to infinity, we have:

$$\begin{aligned}
\lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\epsilon\| &= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{t_{W_{i+1}}} + \boldsymbol{t_\Delta} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_i}} + \boldsymbol{t_\Delta})\| \\
&= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{t_{W_{i+1}}} + \boldsymbol{t_\Delta} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T\boldsymbol{t_{W_i}} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T\boldsymbol{t_\Delta})\| \\
&= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{t_\Delta} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T\boldsymbol{t_\Delta})\| \\
&= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|(\boldsymbol{I_3} - \boldsymbol{R_{W_{i+1}}}\boldsymbol{R_{W_i}}^T)\boldsymbol{t_\Delta})\| \\
&= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{C}\boldsymbol{t_\Delta})\| \\
&= \infty
\end{aligned}$$

A.1.2 Local Formulation of Residual Error

As previously stated in Section 6.4.1, the displacement matrix $\boldsymbol{D_i}$ is expressed as:

$$\boldsymbol{D_i} = \boldsymbol{W_i}^{-1}\boldsymbol{W_{i+1}} \tag{A.10}$$

Using Equation A.6, we can rewrite $\boldsymbol{D_i}$ as:

$$\boldsymbol{D_i} = \begin{bmatrix} \boldsymbol{R_{W_i}}^T\boldsymbol{R_{W_{i+1}}} & \boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_{i+1}}} - \boldsymbol{t_{W_i}}) \\ \boldsymbol{0_3} & 1 \end{bmatrix} \tag{A.11}$$

The residual error is then defined as:

$$\epsilon = \boldsymbol{t_i} = \boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_{i+1}}} - \boldsymbol{t_{W_i}}) \tag{A.12}$$

At that point, we add a translation component to the $\boldsymbol{W_i}$ and $\boldsymbol{W_{i+1}}$ matrices. In other words:

$$\begin{aligned}
\boldsymbol{t_{W_i}} &\to \boldsymbol{t_{W_i}} + \boldsymbol{t_\Delta} \\
\boldsymbol{t_{W_{i+1}}} &\to \boldsymbol{t_{W_{i+1}}} + \boldsymbol{t_\Delta}
\end{aligned}$$

Then, if $\|\boldsymbol{t_\Delta}\|$ tends to infinity, we have:

$$\begin{aligned}
\lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\epsilon\| &= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_{i+1}}} + \boldsymbol{t_\Delta} - \boldsymbol{t_{W_i}} - \boldsymbol{t_\Delta})\| \\
&= \lim_{\|\boldsymbol{t_\Delta}\|\to\infty} \|\boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_{i+1}}} - \boldsymbol{t_{W_i}})\| \\
&= \|\boldsymbol{R_{W_i}}^T(\boldsymbol{t_{W_{i+1}}} - \boldsymbol{t_{W_i}})\| \\
&= constant
\end{aligned}$$

A.1.3 Numerical Comparisons Between Both Formulations

In this section, we numerically show that our formulation is independent of the global position of the animation while the previous one is not. The test animation we used is shown in Figure 6.5.

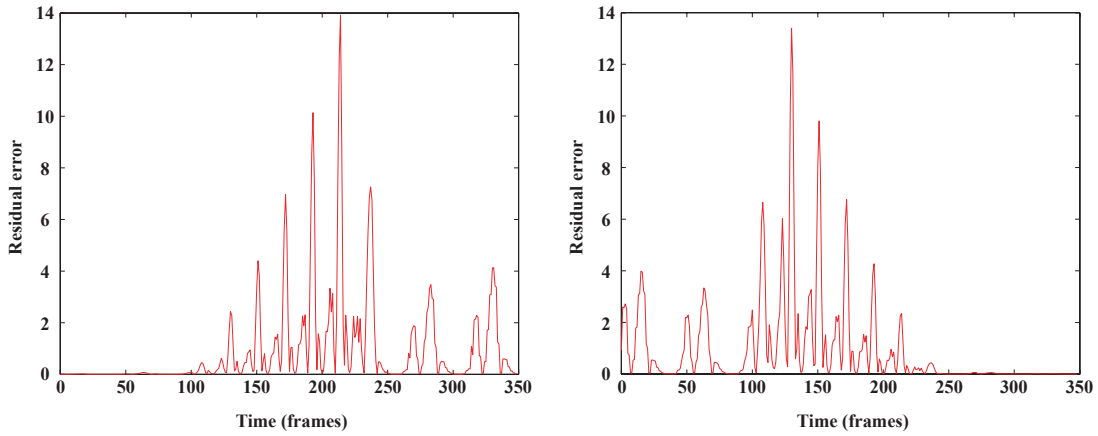

Figure A.1: Residual errors for the animation shown in Figure 6.5 with different starting positions in space. **Left:** The character initial position is $[\mathbf{0}, \mathbf{0}, \mathbf{0}]$. Its final position is $[\mathbf{17}, \mathbf{0}, \mathbf{0}]$. **Right:** The character initial position is $[-\mathbf{17}, \mathbf{0}, \mathbf{0}]$. Its final position is $[\mathbf{0}, \mathbf{0}, \mathbf{0}]$.

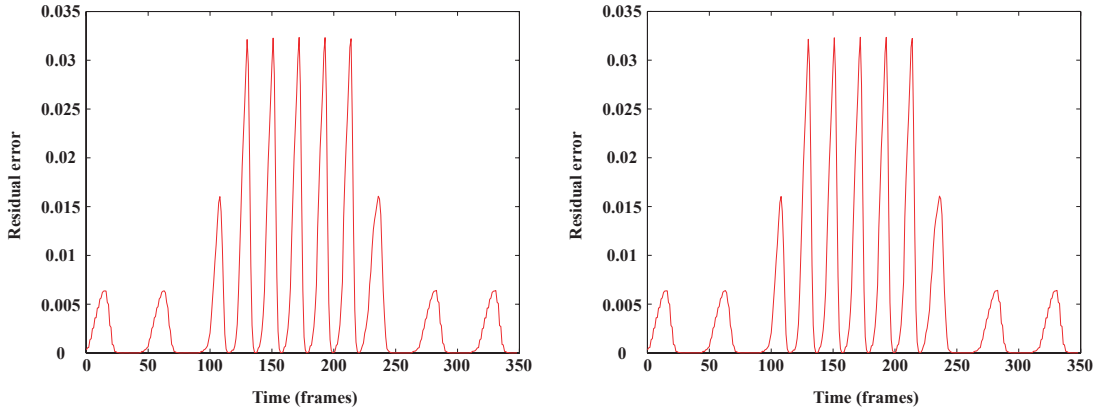

Figure A.2: Residual errors for the animation shown in Figure 6.5 with different starting positions in space. **Left:** The character initial position is $[\mathbf{0}, \mathbf{0}, \mathbf{0}]$. Its final position is $[\mathbf{17}, \mathbf{0}, \mathbf{0}]$. **Right:** The character initial position is $[-\mathbf{17}, \mathbf{0}, \mathbf{0}]$. Its final position is $[\mathbf{0}, \mathbf{0}, \mathbf{0}]$.

Figure A.1 clearly shows that the more the character moves away from the origin, the higher the residual error.

Figure A.2 shows that our formulation is independent from the global position of the animation in the scene: for the same walking motion, we approximately get the same residual error.

APPENDIX B

Tasks Description for the Examples of Chapter 3

B.1 Example of Figure 3.3: The Thinker by Auguste Rodin

Joint	Type	Priority	Recruiting Level
Ankles	Positional	15	4 (Up to the root)
Hips	Positional	10	2 (Up to the root)
Left wrist	Positional	9	9 (Up to the root)
Root (located near the pelvis)	Positional	9	0 (Itself)
Right elbow	Positional	8	8 (Up to the root)
Left elbow	Positional	7	8 (Up to the root)
Right wrist	Positional	4	1 (right wrist + right elbow)
Right wrist	Rotational	1	0 (Itself)
Left wrist	Rotational	1	0 (Itself)
All	Optimization	Lowest	All joints

Remark: This set of tasks is very flexible: changing the priority and/or the recruiting level of a task may give the same results. What really matters is that the relative importance between tasks is well defined. For example, the position of the ankles is given a high priority because whatever the final posture is, we want the ankles to stay stationary while manipulating the virtual character.

B.2 Example of Figure 3.4: Center of Mass Control

Joint	Type	Priority	Recruiting Level
Heels	Positional	15	5 (Up to the root)
Toes	Positional	15	7 (Up to the root)
Right hand	Positional	5	10 (Up to the root)
All	CoM	10	3 (All joints)
All	Optimization	Lowest	All joints

LIST OF FIGURES

BIBLIOGRAPHY

Yeuhi Abe, C. Karen Liu, and Zoran Popovic. Momentum-based parameterization of dynamic character motion. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004. 2.2.3

Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient Similarity Search In Sequence Databases. In *Proceedings of International Conference of Foundations of Data Organization and Algorithms*, pages 69–84, 1993. 2.3.1

Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from Motion. In *Proceedings of Graphics Interface*, pages 222–229, 1996. 2.2.1

Okan Arikan and D. A. Forsyth. Interactive Motion Generation From Examples. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 483–490, 2002. 2.3.2, 5

Okan Arikan, David A. Forsyth, and OBrien. Motion synthesis from annotations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 402–408, jul 2003. 2.9, 2.3.2, B.2

Amaury Aubel. *Anatomically-Based Human Body Deformations*. PhD thesis, EPFL, 2002. 1.4.1, 3.3.2.1

Ronald Azuma and Gary Bishop. Improving Static and Dynamic Registration in an Optical See-Through HMD. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 197–214, jul 1994. 6.4.3.2

Norman I. Badler, J. O'Rourke, and G. Kaufman. Special Problems in Human Movement Simulation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 189–197, jul 1980. 3.3.1.3, 5.6

Paolo Baerlocher. *Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures*. PhD thesis, EPFL, 2001. 1.4.1, 3.2, 3.2.3, 3.2.3, 3.2.5, 3.3.2.1, 3.4, 7.1

Paolo Baerlocher and Ronan Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Proceedings of International Conference on Intelligent Robots and Systems*, 1998. 3.1.2

N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 322–331, may 1990. 2.3.1

Rama Bindiganavale and Norman I. Badler. Motion Abstraction and Mapping with Spatial Constraints. *Lecture Notes in Computer Science*, 1537:70–83, 1998. 4.1, 6.2

Jonathan Blow. Inverse Kinematics with Joint Limits. *Game Developer Magazine*, apr 2002. 3.1.3

Ronan Boulic and Daniel Thalmann. Combined Direct and Inverse Kinematic Control for Articulated Figures Motion Editing. *Computer Graphics Forum*, 11(4):189–202, 1992. 5.1, B.2

Ronan Boulic, Ramon Mas, and Daniel Thalmann. A robust approach for the control of the center of mass with inverse kinetics. *Computer and Graphics*, 20(5):693–701, sep 1996. 3.3.1.2, 4.4

Ronan Boulic, Branislav Ulicny, and Daniel Thalmann. Versatile Walk Engine. *Journal of Game Development*, 1(1), 2004. 2.2, 2.1.2, B.2

Armin Bruderlin and Lance Williams. Motion Signal Processing. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 97–104, 1995. 2.2.1, 4.2.3

Samuel R. Buss and Jay P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics*, 20(2):95–126, 2001. 6.4.3.2

Marc Cardle, Michail Vlachos, Stephen Brooks, Eamonn Keogh, and Dimitrios Gunopulos. Fast Motion Capture Matching with Replicated Motion Editing. In *Proceedings of ACM SIGGRAPH, Technical Sketches and Applications*, 2003. 2.3.1

Jinxiang Chai and Jessica K. Hodgin. Performance Animation from Low-dimensional Control Signals. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2005. 2.1.4, 2.4, B.2

K-P. Chan and A. W-C. Fu. Efficient Time Series Matching by Wavelets. In *Proceedings of International Conference on Data Engineering*, pages 126–135, mar 1999. 2.3.1

Diane M. Chi, Monica Costa, Liwei Zhao, and Norman I. Badler. The EMOTE Model for Effort and Shape. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 173–182, 2000. 2.1.2

Stephano Chiaverini. Singularity-Robust Task-Priority Redundancy Resolution for Real-Time Kinematic Control of Robot Manipulators. *IEEE Transactions on Robotics and Automation*, 13(3):398–410, 1997. 3.1.2

Kwang-Jin Choi and Hyeong-Seok Ko. Online Motion Retargetting. *Journal of Vizualisation and Computer Animation*, 11:223–235, 2000. 2.2.4, 5.3, 5.1, 5.2, B.2

Michael F. Cohen. Interactive spacetime control for animation. In Edwin E. Catmull, editor, *Proceedings of the 19th Annual ACM Conference on Computer Graphics and Interactive Techniques*, pages 293–302, New York, NY, USA, July 1992. ACM Press. 2.2.3

John. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, 1986. 3.1.1, 3.2.2, 3.3.3

Hari Das, Jean-Jacques Slotine, and Thomas B. Sheridan. Inverse Kinematics Algorithms for Redundant Manipulators. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 43–48, 1988. 3.1.2

Anirvan Dasgupta and Yoshihhiko Nakamura. Making feasible walking motion of humanoid robots from human motion capture data. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1044 – 1049, may 1999. 2.2.2, 4.1

Luc Emering, Roman Boulic, and Daniel Thalmann. Interacting with Virtual Humans. *IEEE Computer Graphics and Applications*, 18(1):8–11, jan 1998. 2.3.1

C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 419–429, 1994. 2.3.1

Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable Controllers for Physics-Based Character Animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2001a. 2.3, 2.1.3, B.2

Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. The Virtual Stuntman: Dynamic Characters with a Repertoire of Autonomous Motor Skills. *Computer and Graphics*, 25(6):pp. 933–953, dec 2001b. 2.1.3

Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 417–426, jul 2003. 2.2.3

Pascal Glardon, Ronan Boulic, and Daniel Thalmann. A Coherent Locomotion Engine Extrapolating Beyond Experimental Data. In *Proceedings of Computer Animation and Social Agent*, jul 2004. 2.3.2

Michael Gleicher. Comparing constraint-based motion editing methods. *Graphical Models*, 63(2):107–134, mar 2001. 2.2.4, 5.3

Michael Gleicher. Motion Editing with Spacetime Constraints. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 139–148, apr 1997. 2.2.3, 4.1

Michael Gleicher. Retargetting motion to new characters. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 33–42, 1998. 2.2.3, 4.1, 5.3

Michael Gleicher and Peter Litwinowicz. Constraint-based Motion Adaptation. *Journal of Vizualisation and Computer Animation*, 9(2):65–94, 1998. 2.2.3

Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 47–57, 1984. 2.3.1

Hideo Hanafusa, Tsuneo Yoshikawa, and Yoshihiko Nakamura. Analysis and Control of Articulated Robot Arms with Redundancy. In *Proceedings of IFAC, 8th Triennal World Congress*, pages 1927–1932, 1981. 3.1.2

HAnim. *Humanoid Animation Working Group*. `http://www.hanim.org/`. 1.4.1, 3.2.1

Jessica K. Hodgins and Nancy S. Pollard. Adapting Simulated Behaviors For New Characters. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 153–162, 1997. 2.1.3

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating Human Athletics. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 1995. 2.1.3

H. V. Jagadish. Spatial Search with Polyhedra. In *Proceedings of International Conference on Data Engineering*, pages 311–319, 1990. 2.3.1

O. Jenkins and M. Mataric. Deriving action and behavior primitives from human motion data. In *Proceedings of International Conference on Intelligent Robots and Systems*, 2002. 2.3.1

Eamonn Keogh, Themistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing Large Human-Motion Databases. In *Proceedings of International Conference on Very Large Data Bases*, 2004. 2.3.1

Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, 2001. 2.3.1

Charles A. Klein and Ching-Hsiang Huang. Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, 13(3):245–250, mar 1983. 3.1.2

Hyeongseok Ko and Norman I. Badler. Animating Human Locomotion with Inverse Dynamics. *IEEE Computer Graphics and Applications*, 16(2):50–58, mar 1996. 2.1.3, 2.2.2, 4.1

D. H. U. Kochanek and R. H. Bartels. Interpolating Splines with Local Tension, Continuity and Bias Control. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 33–43, 1984. 4.2.2, 4.2.2

James U. Korein. *A Geometric Investigation of Reach*. MIT press, 1985. 3.1.1

Lucas Kovar and Michael Gleicher. Flexible Automatic Motion Blending with Registration Curves. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 214–224, 2003. 2.3.2, 5

Lucas Kovar and Michael Gleicher. Automated Extraction and Parameterization of Motions in Large Data Sets. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.3.1

Lucas Kovar, Michael Gleicher, and Fred Pighin. Motion Graphs. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 473–482, 2002a. 2.3.2, 4.3.2, 5, 6.2

Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate Cleanup for Motion Capture Editing. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 97–104, 2002b. 2.2.4, 4.3, 5.6, 5.1, 5.2, 5.3.1, B.2

Richard Kulpa, Franck Multon, and Bruno Arnaldi. Morphology-independent Representation of Motions for Interactive Human-like Animation. In *Proceedings of Eurographics*, pages 343–352, aug 2005. 2.2.4, 3.1.3, 5.4, 5.1, B.2

John Lasseter. Principles of traditional animation applied to 3D computer animation. volume 21, pages 35–44, July 1987. 2.1.1, 2.1, B.2

Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Limit Cycle Control and its Application to the Animation of Balancing and Walking. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 155–162, 1996. 2.1.3

Jehee Lee and Sung Yong Shin. A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 39–48, 1999. 2.2.4, 3.1.4, 5.1, 5.2, 5.1, 5.2, B.2

Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive Contol of Avatars Animated With Human Motion Data. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 491–500, 2002. 2.3.2, 5, 6.2

Alain Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems Man and Cybernetics*, 7(12):868–871, 1977. 3.1.2, 3.2.3

C. Karen Liu and Zoran Popovic. Synthesis of Complex Dynamic Character Motion from simple animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 408–416, 2002. 2.7, 2.2.3, 4.1, 4.1, 6.2, 6.4.1, B.2

Feng Liu, Yueting Zhuang, Fei Wu, and Yunhe Pan. 3D motion retrieval with motion index tree. *Computer Vision and Image Understanding*, 92(2):265–284, 2003. 2.8, 2.3.1, B.2

Anthony A. Maciejewski. Dealing with the Ill-Conditioned Equations of Motion for Articulated Figures. *IEEE Computer Graphics and Applications*, 10(3):63–71, may 1990. 3.2.4

Anthony A. Maciejewski and Charles A. Klein. Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments. *International Journal of Robotic Research*, 4(3), 1985. 3.1.2

Anthony A. Maciejewski and Charles A. Klein. Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations. *Journal of Robotic Systems*, 5(6):527–552, 1988. 3.1.2

Alberto Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., 1999. ISBN 0124906303. 2.1.4, 5

Stephane Menardais, Richard Kulpa, Franck Multon, and Bruno Arnaldi. Synchronization of interactively adapted motions. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, aug 2004. 6.2

Jean-Sébastien Monzani, Paolo Baerlocher, Ronan Boulic, and Daniel Thalmann. Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting. In *Proceedings of Eurographics*, 2000. 2.2.4, 5.1, 5.5, 5.2, B.2

David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics, 3rd ed. New York*. W. H. Freeman, 1999. 6.4.2

Tomohiko Mukai and Shigeru Kuriyama. Geostatistical Motion Interpolation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2005. 2.3.2

Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for Robot Manipulator Control. *Journal of Dynamic Systems, Measurement, and control*, 108:163–171, sep 1986. 3.1.2, 3.2.4

Yoshihiko. Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-Priority Based Redundancy Control of Robot Manipulators. *International Journal of Robotic Research*, 6 (2), 1987. 3.1.2

Michael Neff and Eugene Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 81–88, 2002. 2.1.3

Michael Neff and Eugene Fiume. Aesthetic Edits For Character Animation. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2003. 2.2.2

Michael Neff and Eugene Fiume. AER: Aesthetic Exploration and Refinement for Expressive Character Animation. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2005. 2.1.3

J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984. 2.3.1

Jack A. Orenstein. Spatial query processing in an object-oriented database system. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 326–336, 1986. 2.3.1

James M. Ortega and Werner C. Rheinboldt. Iterative solution of nonlinear equations in several variables. 2000. 3.1.2

Carol O'Sullivan, John Dingliana, Thanh Giang, and Mary K. Kaiser. Evaluating the visual fidelity of physically based animations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2003. 2.2.2, 4

Yi-Chung Pai and James Patton. Center of mass velocity-position predictions for balance control. *Journal of Biomechanics*, 30(4):347–354, 1997. 4.1

Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line Locomotion Generation Based on Motion Blending. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, 2002. 2.3.2, 5, 6.4.3.2, 6.13, 6.4.3.2, B.2

Woojin Park, Don B. Chaffin, and Bernard J. Martin. Toward Memory-Based Human Motion Simulation: Development and Validation of a Motion Modification Algorithm. *IEEE Transactions on systems, man, and cybernetics*, 34(3):376–386, may 2004. 2.3.2

Richard P. Paul. *Robot Manipulators: Mathematics, Programming and Control*. MIT press, 1981. 3.1.1, 3.2.2

Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1), mar 1995. 2.1.2, 5

Ken Perlin and Athomas Goldberg. Improv: A System for Scripting Interactive Actors in Virtual Worlds. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 205–216, 1996. 2.1.2

Nancy S. Pollard and Paul S. A. Reitsma. Animation of Humanlike Characters: Dynamic Motion Filtering with a Physically plausible Contact Model. In *Proceedings of Yale Workshop on Adaptative and Learning Systems*, 2001. 2.2.2, 4.1

Zoran Popovic and Andy Witkin. Physically Based Motion Transformation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 11–20, 1999. 2.2.3, 4.1

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, 2nd. edition*. Cambridge University Press, 1992. 3.2.3, 6.4.2, A.1

Davood Rafiei and Alberto Mendelzon. Similarity-based queries for time series data. In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 13–25, 1997. 2.3.1

Paul S. A. Reitsma and Nancy S. Pollard. Perceptual metrics for character animation: Sensitivity to Errors in Ballistic Motion. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 537–542, jul 2003. 2.2.2, 4

Paul S. A. Reitsma and Nancy S. Pollard. Evaluating Motion Graphs for Character Navigation. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004. 2.3.2

Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient Generation of Motion Transitions using Spacetime Constraints. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 147–154, 1996. 2.2.3, 4.1

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and Adverbs: Multidimensional Motion Interpolation Using Radial Basis Functions. *IEEE Computer Graphics and Applications*, 18(5):32–41, sep 1998. 2.3.2

P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley & Sons, 1987. 6.4.2, 6.4.2

Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing Physically Realistic Human Motion in Low-Dimensional, Behavior-Specific Spaces. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.2.3

Yasuhiko Sakamoto, Shigeru Kuriyama, and Toyohisa Kaneko. Motion Map: Image-based Retrieval and Segmentation of Motion Data. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004. 2.3.1

Philippe Sardain and Guy Bessonnet. Forces acting on a biped robot. Center of pressure-zero moment point. *IEEE Transactions on Systems Man and Cybernetics*, 34(5):630 – 637, sep 2004. 4.1

Lorenzo Sciavicco and Bruno Siciliano. A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators. *IEEE Journal of Robotics and Automation*, 4(4): 403–410, aug 1988. 3.1.2

Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The R -Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of International Conference on Very Large Data Bases*, pages 507–518, 1987. 2.3.1

Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics*, 20:67–94, 2001. 2.2.4, 3.1.4, 5.1, 5.1, 5.2, 5.6, 6.6

Hyun Joon Shin, Lucas Kovar, and Michael Gleicher. Physical Touch-up of Human Motions. In *Proceedings of Pacific Graphics*, oct 2003. 2.2.2, 4.1

Ken Shoemake. Animating Rotation with Quaternions Curves. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 245–254, 1985. 6.4.3.2

Bruno Siciliano and Jean-Jacques E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Proceedings of International Conference on Advanced Robotics*, pages 1211–1216, jun 1991. 3.1.2, 3.2.5

Scott. N. Steketee and Norman. I. Badler. Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 255–262, jul 1985. 4.2

Seyoon Tak, Oh-Young Song, and Hyeong-Seok Ko. Spacetime Sweeping: An Interactive Dynamic Constraints Solver. In *Proceedings of Computer Animation and Social Agents*, pages 261–270, 2002. 2.2.2, 4.1, 5.6

Deepak Tolani, Ambarish Goswami, and Norman I. Badler. Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs. *Graphical Models*, 62(5):353–388, 2000. 3.1.1, 3.1.4, 3.2.2

Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier Principles for Emotion-based Human Figure. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 91–96, aug 1995. 2.5, 2.2.1, B.2

Michail Vlachos, Marios Hadjieleftheriou, and Dimitrios Gunopulos and. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance. *International Conference on Knowledge Discovery and Data Mining*, 2003. 2.3.1

Allan Watt and Mark Watt. Advanced animation and rendering techniques - Theory and Practice. *Addison-Wesley*, 1992. 3.2.1, 3.2.3

Chris Welman. Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. Master thesis, Simon Frasier University, sep 1993. 3.1.2, 3.1.3

Daniel E. Whitney. Resolved Motion Rate Control of Manipulators and Human Prostheses. *IEEE Transactions on man-machine systems*, 10(2):47–53, jun 1969. 3.1.2

David A. Winter. Biomechanics and Motor Control of Human Movement. *Wiley, New York*, 2004. 2.1.3

A Witkin and M. Kass. Spacetime Constraints. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 159–168, 1988. 2.2.3

Andrew Witkin and Zoran Popovic. Motion Warping. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 105–108, 1995. 2.2.1, 4.1, 4.2.3, 5.3

Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 225–232, 1987. 4.1, 4.1

William A. Wolovich and Howard Elliot. A Computational Technique for Inverse Kinematics. In *Proceedings of 23rd Conference on Decision and Control*, pages 1359–1362, dec 1984. 3.1.2

W. L. Wooten and J. K. Hodgins. Simulating leaping, tumbling, landing and balancing humans. In *Proceedings of IEEE International Conference on Robotics and Automation*, apr 2000. 2.1.3, 4.1

K. Yamane and Y. Nakamura. Dynamics Filter-Concept and Implementation of On-line Motion Generator for Human Figures. *IEEE Transactions on Robotics and Automation*, 19(9):421–432, jun 2003. 2.2.2, 4.1

Victor Zordan and Jessica Hodgins. Motion Capture-Driven Simulations that Hit and React. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 89–96, 2002. 2.2.2, 4.1

Victor Zordan and Nicholas Van Der Horst. Mapping Optical Motion Capture Data to Skeletal Motion Using a Physical Model. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 245–250, 2003. 2.6, 2.2.2, B.2

Victor B. Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic Response for Motion Capture Animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2005. 2.2.2

Curriculum Vitae

Name	Benoît Le Callennec
Date of birth	January 14th, 1978, in Pabu (22), France
Nationality	French
Mother tongue	French
Languages	Good knowledge of English

Education

2001-2002

Post Graduation (Virtual Reality and Multimodal Interaction)

École Polytechnique Fédérale de Lausanne (Switzerland)

Thesis: An HANIM-compliant Inverse Kinematics Solver using a Task-Priority Strategy

2000-2001

Master of Science (Computer Science)

IFSIC, University of Rennes 1, France

Thesis: 3D models Registration

1996-2000

B. Sc in Computer Science

IFSIC, University of Rennes 1, France

Professional Activities

Fall 2001 - Winter 2005

Research assistant at the VRLab-EPFL, Lausanne, Switzerland.

Summer 2001

Development of 3DSMax plugins for 3D models generation.

Spring 2001

Master's training period: "3D models Registration" in computer vision domain.

Fall 2000

Teaching Assistant for a Turbo Pascal Programming course

Summer 2000

Design of a web site for E-business. (www.bretagne-specialites.com)

Summer 1999

Caller position in an administrative office.

Publications

- Benoît Le Callennec, and Ronan Boulic. Interactive Motion Deformation with Prioritized Constraints. *Graphical Models*. TO APPEAR.
- Hareesh Puthiya Veettilh, Ronan Boulic, Arun Sharma, Benoît Le Callennec, Kazuya Sawada, and Daniel Thalmann. An Intuitive IK Postural Control System for Anthropometric Digital Human Models. *In proceedings of the 11th International Conference on Virtual Systems and Multimedia*, October 2005.
- Manuel Peinado, Ronan Boulic, Benoît Le Callennec, and Daniel Méziat. Progressive Cartesian Inequality Constraints for the Inverse Kinematics Control of Articulated Chains. *In Proceedings of Eurographics, Short Presentation session*, Dublin September 2005.
- Marc Salvati, Benoît Le Callennec, and Ronan Boulic. A Generic Method for Geometric Constraints Detection, *In Proceedings of Eurographics, Short presentation session*. September 2004.
- Manuel Peinado, Bruno Herbelin, Marcelo M. Wanderley, Benoît Le Callennec, Ronan Boulic, Daniel Thalmann, and Daniel Méziat. Towards Configurable Motion Capture with Prioritized Inverse Kinematics. *In Proceedings of the third International Workshop on Virtual Rehabilitation*, September 2004.
- Benoît Le Callennec, and Ronan Boulic. Interactive Motion Deformation with Prioritized Constraints. *In Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, August 2004.
- Ronan Boulic, Benoît Le Callennec, Martin Herren, and Herbert Bay. Motion Editing with Prioritized Constraints. *In Proceedings of RichMedia*, October 2003.
- Ronan Boulic, Benoît Le Callennec, Martin Herren, and Herbert Bay. Experimenting Prioritized IK for Motion Editing. *In Proceedings of Eurographics, Slides & Video session*, September 2003.