Performance transfer : animating virtual charaters by playing and acting.

Maxime Garcia

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Maxime GARCIA

Thèse dirigée par **Rémi RONFARD**, Directeur de Recherche et Responsable Equipe, Communauté Université Grenoble Alpes

préparée au sein du **Laboratoire Laboratoire Jean Kuntzmann** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Transfert d'animation : animer des personnages virtuels par le jeu et le mime

Performance transfer: animating virtual charaters by playing and acting.

Thèse soutenue publiquement le **19 décembre 2019**, devant le jury composé de :

Monsieur REMI RONFARD
DIRECTEUR DE RECHERCHE, INRIA CENTRE DE GRENOBLE RHÔNEALPES, Directeur de thèse
Monsieur YIORGOS CHRYSANTHOU
PROFESSEUR, UNIVERSITE DE CHYPRE, Rapporteur
Monsieur JULIEN PETTRE
DIRECTEUR DE RECHERCHE, INRIA CENTRE RENNES-BRETAGNE ATLANTIQUE, Rapporteur
Madame SYLVIE GIBET
PROFESSEUR, UNIVERSITE BRETAGNE-SUD, Examinateur
Madame SARAH FDILI-ALAOUI
MAITRE DE CONFERENCES, UNIVERSITE PARIS-SUD, Examinateur
Madame JOËLLE THOLLOT
PROFESSEUR, GRENOBLE INP, Président

Abstract

During the past decades 3D animation widely spread into our everyday life being for entertainment such as video games or movies, or for communication more generally. Despite its common use, creating animations is still dedicated to skilled animators and not within reach of non-experts. In addition, the creation process traditionally follow a strict pipeline: after the initial storyboarding and character design phases, articulated characters are modeled, rigged and roughly positioned in 3D, at which point the layout of their relative poses can be established. Then, their actions and movements are broken down into keyframes which are interpolated to produce the final animation. Keyframing animations is a hard process during which animators need to spend time and efforts carefully tuning animation curves for each character's degrees of freedom and for each specific action. They also need to consistently sequence these actions over time in order to convey the desired character's intentions and personality. Some methods such as motion capture, aim at easing the creation process by directly transferring real recorded human motions to virtual characters. However, output animations still lack expressiveness and must be fixed by animators.

In this thesis, we propose a way of easing the process of creating animation sequences starting from a database of individual animations. We focus on animating virtual characters reproducing a story played with props such as figurines instrumented with sensors. The main goal of this thesis is to compute expressive and plausible animations from the data provided by the sensors in order to transpose the story told by the narrator and its figurines into the virtual world. To reach this goal, we propose a new animation pipeline analyzing and transcribing hands motion into a sequence of animations adapted to the user hand space-time trajectories and their motion qualities. We present a new translation, rotation and scale invariant motion descriptor allowing our system to perform action recognition as well as two different classifiers extracting Laban Effort motion qualities from an input curve. We also introduce a new procedural animation model inferring expressiveness fitting the narrator's hand motion qualities in terms of Laban Time and Weight Effort. Finally, we extend our system such that it can process multiple characters at a time, detecting and transferring interactions as well as making characters act with respect to

pre-defined behaviors, letting users express their narrative creativity.

Our system capabilities were tested through different user studies during which non expert users could follow a script or freely improvise stories with two figurines at a time. We conclude with a discussion of future research directions.

Résumé

Durant les dernières décennies, l'animation 3D s'est largement intégrée à notre vie quotidienne, que cela soit dans le domaine du jeu vidéo, du cinéma ou du divertissement plus généralement. Malgré son utilisation rependue, la création d'animation reste réservée à des animateurs expérimentés et n'est pas à la portée de novices. Par ailleurs, le processus de création d'animation doit respecter une pipeline très stricte : après une première phase de storyboarding et de design des personnages, ceux-ci sont ensuite modélisés, riggés et grossièrement positionnés dans l'espace 3D permettant de créer un premier brouillon d'animation. Les actions et mouvements des personnages sont ensuite décomposés en keyframes interpolées, constituant l'animation finale. Le processus de keyframing est difficile et nécessite, de la part des animateurs, beaucoup de temps et d'efforts, notamment pour la retouche de chaque courbe d'animation liée à un degré de liberté d'un personnage, et ceci pour chaque action. Ces dernières doivent aussi être correctement séquencées au cours du temps afin de transmettre les intentions des personnages et leur personnalité. Certaines méthodes telle que la motion capture rendent le processus de création plus aisé en transférant le mouvement de véritables acteurs aux mouvements de personnages virtuels. Cependant, ces animations transférées manquent souvent d'expressivité et doivent être rectifiées par des animateurs.

Dans cette thèse, nous introduisons une nouvelle méthode permettant de créer aisément des séquences d'animation 3D à partir d'une base de donnée d'animations individuelles comme point de départ. En particulier, notre travail se concentre dans l'animation de personnages virtuels reproduisant une histoire jouée avec des objets rigides, tels que des figurines, instrumentées par des capteurs. Notre principal objectif est de calculer des animations plausibles et expressives depuis les données retournées par les capteurs dans le but de transformer l'histoire racontée par le narrateur et ses figurines en une animation 3D. Afin d'atteindre cet objectif, nous proposons un nouveau pipeline d'animation, analysant et traduisant le mouvement des mains en séquence d'animations adaptées à leur trajectoire espace-temps ainsi qu'à leur qualité de mouvement. Nous présentons un descripteur de mouvement invariant par translation, rotation et passage à l'échelle, permettant à notre système de reconnaître des ac-

tions exécutées par l'utilisateur ; nous présentons également deux classificateurs reconnaissant les qualités du mouvement en tant qu'Effort de Laban. Nous introduisons un nouveau modèle d'animation procédural traduisant l'expressivité de la qualité de mouvement de la main en tant qu'Effort de la Laban Temps et Poids. Enfin, nous étendons le système afin de permettre la manipulation de plusieurs personnages en même temps, en détectant et transférant des interactions entre personnages tout en étant fidèle aux qualités de mouvement du narrateur et permettant aux personnages d'agir selon des comportements prédéfinis, laissant l'utilisateur exprimer sa créativité.

Les capacités de notre système ont été évaluées à travers plusieurs études utilisateurs durant lesquelles des novices ont pu suivre un script ou ont librement improvisé des histoires à deux figurines. Nous concluons avec une discussion sur les futures directions de recherche.

Thanks

First, I would like to thank my supervisor Remi Ronfard for his guidance and all the advises he gave me during my PhD. I would also like to express my gratitude to Marie-Paule Cani who helped me a lot for submissions, shared many ideas and gave me opportunities to participate in different research events and projects.

I also thank everyone who helped me for submissions. Laurence Boissieux provided me models, animations and ideas for demonstrating our methods. She also made many animation renderings for our videos. I am also grateful to Adela Barbulescu who gave me advises for making research posters and with whom I wrote my first research articles. I thank Gillian Borrell and Joelle Thollot for testing our system and acting two characters plays that were used to illustrate our methods. I also thank Youna Le Vaou and Ameya Murukutla for lending their voice in the videos accompanying the articles published during my PhD.

Within these three years, I also finished the work I stared during my Master internship dealing with the topic of inverse geological storytelling from a single 2D sketch. This work has been published in the Expressive 2018 conference proceedings thanks to the help of Estelle Charleroy, Claude Gout and Christian Perrenoud.

This PhD thesis also allowed me to start the biggest software project I ever worked on, namely the SkyEngine. Programming this game engine was really fun (and still is) and allowed me to collaborate with colleagues to implement research work and a wide range of standard 3D graphics algorithms. Therefore, I would like to thank everyone who contributed to the project: Julien Daval, Guillaume Cordonnier, Remi Colin de Verdiere, Pierre Casati and Valentin Touzeau.

A huge thank to my co-office colleagues, Debanga, Robin, Gregoire and especially Youna for all the the discussion we had, for bringing me support in the most difficult times and for the great time we had together.

It was a great pleasure to be part of the Imagine team for those 3 years and half, I really enjoyed the time I spend among all its members. I first thank all the permanent members of the team: thank you Stefanie, Melina, Jean-Claude,

Marion, Damien and Frederic. I would also like to thank everyone who is and who was part of the team: thank you Antoine, Ulysse, Pierre-Luc, Romain, Even, Tibor, Pablo, Harold, Sandra, Alex, Maguelonne, Thomas, Melanie, Qianqian, Moussab, David, Valerian, Marion, Paul-Elian, Nicolas (Nghiem), Nicolas (Donati) and Maud. A special thank to Amelie with whom I organized team activities such as the reading group, the doctoral consortium and the drawing sessions.

I give my thanks to everyone at the Anatoscope company, especially Romain, Elie, Elisabeth and Yves. We shared great times together, being at the Annecy festival, at the Grenoble Game Developer meetings (special thank to Arnaud Emilien for creating and leading this group) or running together during the Ekiden race.

I have a special thought for my friends who brought me support during the whole trip. Thank you Valentin, Noha, Auriane, Anthony, Julien, Benjamin, Raphael, Sonia, and Arnaud.

As my passion for Japanese animation grew during the last years, I also organized many Japanese cultural events and activities within the ANI Grenoble association I was presiding during this thesis. I would like to thank all the members of the association who helped me with the organization but also who allowed me to relieve all the stress I accumulated especially during karaoke sessions.

Finally, I would like to thank my family for supporting me until the very end of this thesis.

Contents

Chapter 1

Introduction

Playing with toys for creating stories is a very common thing to do as a child. Using figurines which have prominent aspects and traits highly stimulate the creativity of children who imagine diverse stories in their mind. The motivation behind this thesis, is to extract these stories created by playing with figurines and turn them into 3D animations. More precisely, this work aims at letting people draft 3D animations while expressing their narrative creativity using physical props such as toys.

As an illustration of this objective, figure 1.1 shows the very first scene of Toy's Story 1 by Pixar animation studio with its corresponding script. During this scene, we see a child, Andy, playing with toys while embodying each of them in turn. In the story created by Andy, he opposes Potato Head, the villain and Woody, the sheriff, in a western style scene and adds some extra elements such as a dinosaur and a dog casting force fields, letting his imagination going free.

This scene is very interesting as it shows that using only lifeless toys while embodying different characters as well as using clever camera motion and placement, already immerse spectators inside the story being played in front of them. This immersion is so intense that Pixar decided to give life to the toys as their were meant to be alive. This scene also shows how easy and inspiring it is to improvise a story using this kind of tangible objects.

Pixar went even further in Toy's Story 3, also in the introduction scene (figure 1.2), during which we can see Andy's toys as life-like characters in a very dynamic sequence of a western style chase full of anachronisms mixing steam train, force fields, dinosaurs and a pig shaped spaceship. At the end of this scene, the spectators are shown that everything they just saw was actually the mental projection of Andy.

ANDY (AS POTATO HEAD)
Alright everyone, this is a stick-up! Don't anybody move! Now empty that safe!

A GROUP OF TOYS have been crowded together in front of the "BANK" box.

Andy's hand lowers a CERAMIC PIGGY BANK in front of Mr. Potato Head and shakes out a pile of coins to the floor. Mr. Potato Head kisses the coins.

ANDY (AS POTATO HEAD)
Ooh! Money. Money. Money.
(kissing noises)

A porcelain figurine of the shepherdess, BO PEEP, is brought into the scene.

ANDY (AS BO PEEP)
Stop it! Stop it, you mean old potato!

ANDY (AS POTATO HEAD)
Quiet Bo Peep, or your sheep get run over!

The companion porcelain sheep are placed in the center of a Hot Wheels track loop.

ANDY (AS SHEEP)
Heeeeelp! BAAAAA! Heeeelp us!

ANDY (AS BO PEEP)
Oh, no! Not my sheep! Somebody do something!

WOODY, a pull-string doll cowboy, enters into the scene opposite the inanimate spud.

Andy's hand pulls on the ring in the center of Woody's back.

WOODY (VOICE BOX)
Reach for the sky.

ANDY (AS POTATO HEAD)
Oh, no! Sheriff Woody!!

ANDY (AS WOODY)
I'm here to stop you, One-Eyed Bart.

Andy's hand pulls out one of Mr. Potato Head's eyes.

ANDY (AS POTATO HEAD)
Doooooh! How'd you know it was me!

ANDY (AS WOODY)
Are you gonna come quietly?

Figure 1.1: Toy's Story 1 introduction scene in which Andy opposes Potato Head against Woody in a western style scene.

(a) Train Chase (b) Buzz Lightyear rescue (c) Pig spaceship entrance (d) Real world view

Figure 1.2: Toy's Story 3 introduction scene. This scene combines an old western style chase with very advanced technologies such as force fields and space ship which turns Andy toys' into more realistic characters. The viewer then realizes that everything he saw is the mental projection of Andy.

Our main goal can be seen as bringing life to real life toys, transforming children plays into 3D animated stories like Pixar did with virtual toys. In this thesis we describe new methods for analyzing performances and transferring them to character animations.

1.1 Performance Transfer

Real life puppets can have various aspects and ways of manipulating them. They can be fully rigid, articulated or deformable. Some puppets even combine several of these features. In addition, depending on their size and their shape they can be manipulated with rods, strings, with the fingers, the whole hand or with both hands (see figure 1.3). These different tangible interaction paradigms are really important as they influence the decisions the narrator might take while creating stories.

(a) Articulated mannequin (b) Rod puppet (c) String puppet (d) Hand puppet

Figure 1.3: Different type of puppets being articulated, manipulated with rods or strings or directly with one hand.

At a higher scale, acting with our whole body is also a mean to create and convey stories; acting and impersonating one character at a time has been done for centuries in theatrical plays. This kind of performance is usually done with several actors who have specific roles during the whole play.

We define performance transfer as the general process of mapping a performed gesture or action to a 3D animated character. Note that this process can be indifferently applied to puppet plays or to theatrical acting.

Transforming full body performances into 3D animations is more commonly referred as motion capture and is widely used in the cinema and video game industry for animating 3D characters, reproducing real actors motions.

In our context, we are deeply interested in transforming puppets performances into 3D animations as they represent an intuitive and powerful way to convey stories. Indeed, this kind of plays yield several advantages with respect to full body performances. These advantages lies in the fact that narrators can manipulate several characters at a time in an environment which can be of very different scales and whose elements can also be easily manipulated. In other words, building its own little world with puppets is very convenient and lets users create various kinds of stories.

Still, the task of transforming a puppet play into a 3D animation is particularly challenging as we attempt to recreate scenes that are inside someone's imagination from a simplified representation composed of physical props motions and the narrator's acting.

The use of puppets have become a topic of growing interest in research, especially for teaching narrative skills to children as presented in the work of [Hayes-Roth and van Gent, 1997], [Mayora et al., 2009] or [Cerezo et al., 2015] who propose systems for tracking puppets and triggering animated events as a response to the motion children perform with the puppets. Those systems seek to propose an interface which is intuitive enough to be manipulated by children and which transcribes their stories, intents and emotions into an output animation. Moreover, these works aim to generate animations with response to children hand motions. More precisely, the challenge addressed by these systems is twoflod: first, designing a motion recognition system which can be used by children, and secondly, transcribe the user's intents into plausible animations.

Those two problems are not completely independent as the way users can manipulate puppets, depending on their type for instance (figure 1.3), affects

the way the virtual characters will be animated. As an example, if users manipulate rod puppets supporting the arms, their motion will be directly mapped to the arms motion of the virtual character. The same could be said if users were using articulated or hand puppets. In the case of figurines, we observe that many systems involving rigid puppets as props for creating animated stories only transfer the figurines' rigid motion to virtual characters and limit motion recognition to few actions. To enhance the plausibility of generated animations, this first motion mapping must be augmented with automatically computed motions animating the rest of the virtual character body, which can be a difficult task to achieve.

In this thesis, we tackle the challenge of producing fully articulated animations while allowing narrators to use a diverse and expendable vocabulary of actions. This can be seen as proposing a new animation sequence authoring tool based on playing and acting with figurines.

1.2 Authoring Animated Stories

Building a system that allows users to author animated stories using puppets requires that we can track their rigid motion, as well as the narrator himself. We also need to map the puppets and the physical playground to a virtual replica in which the resulting animation will be played. Our main goal is then to create an animation sequence that best represents the narrator's mental projection of the story.

As mentioned above, this goal raises several challenges: each figurines' action has to be recognized according to the narrator's intention, animation sequences have to be played with the correct timing and at the right place in the virtual environment; interactions with objects existing only in the virtual world should be possible, and user emotions should also be perceptible in the resulting animations.

1.2.1 Animation authoring methods

Early works tackled those challenges in different contexts. We can distinguish three families of animation sequence generation techniques: sketch based methods, script based methods and acting based methods.

Sketch based authoring

Sketch based techniques attempt to interpret 2D trajectories to either make the character perform an animation sequence based on pattern recognition and interpretation, or to make parts of the body or the whole body itself move along the trajectory. In that first aspect, [Thorne et al., 2004] introduced the concept of *motion doodles* which are side-views 2D curves abstracting the main trajectory of an animation sequence. The input sketch is parsed into motion primitives and translated into a sequence of 3D animations stored in a library. Close-to-planar motion is used, and retiming is inferred from the detection of singular points in the geometry of the curve.

Script based authoring

Script based methods attempts to interpret a user provided scenario into an animation sequence which is guided by space-time constraints. They either let users specify the entire scene development using pseudo languages or let them specify a set of goals for virtual characters that the system will try to reach. This kind of authoring methods have the advantages of requiring no animation or acting skills, letting users focus on the storytelling aspect of the produced animation.

Acting based authoring

Some other methods are focused on translating users' acting either with their hand, using physical props, or their entire body to create animations. Many studies have been conducted using puppets as props to manipulate characters and object in a virtual scene. Illustrating these goals, [Held et al., 2012] proposed a system tracking figurines using a Kinect sensor which both recreates a corresponding 3D model of the object and transcript its rigid motion into a virtual scene. Users are provided with real-time feedback of the figurines motion and can compose several takes to create layered animation sequences. Likewise, in this work we focus on the use of physical props, like figurines, for creating animation sequences.

1.2.2 Authoring animation with figurines

When using figurines as control tools, two interaction paradigms can be used to build an animation system: the "Magic Mirror" and the "Offline Approach". the "Magic Mirror" metaphor provides the user with real-time visual feedback

of the transferred animation into the virtual world while the "Offline Approach" gives feedback only after the end of the recording. The Magic Mirror metaphor presents the advantage of making the user react to what is happening in the virtual world for a two-way physical-virtual interactivity. It can be especially useful to make montages with several animation layers. However, we emphasize that this additional feedback could distract the narrator, hence narrowing his creativity. On the contrary, the "Offline Approach" lets the user freely improvise with the props he has been provided with. In our work, we chose the "Offline Approach" to build our animation transfer system.

As a first approach, we proposed a system which allows users to create animations for two characters. Moreover, we were interested in transferring the narrators facial motion into virtual characters related to physical figurines the narrator plays with (see figure 1.4). As the narrator changes role over time, the main challenge is to figure out which character he embodies at a given frame of the animation. His facial expression is then transferred to the corresponding character. The tracking system FIGURINES [Portaz et al., 2017] is described in chapter 3. To tackle the role assignment problem, we proposed a classification approach using three different inputs: the gaze of the narrator, his voice pitch and the rigid movement of the figurines he holds. Using a naive Bayes classifier we evaluated how well each independent features performed for the task. The results are discussed in [Barbulescu et al., 2017]. Our system is able to identify the figurine the narrator embodies at a given frame and transfer his facial expressions to the corresponding virtual avatar while also transferring their rigid motions. This approach suffers from several limitation though. The major limitation comes from the fact that only rigid motions are transferred to the virtual characters' body and thus they do not perform any action outside their facial motions and rigid motions. Therefore the narrators intent is not fully transferred into the virtual avatar motions as they do not perform any actions.

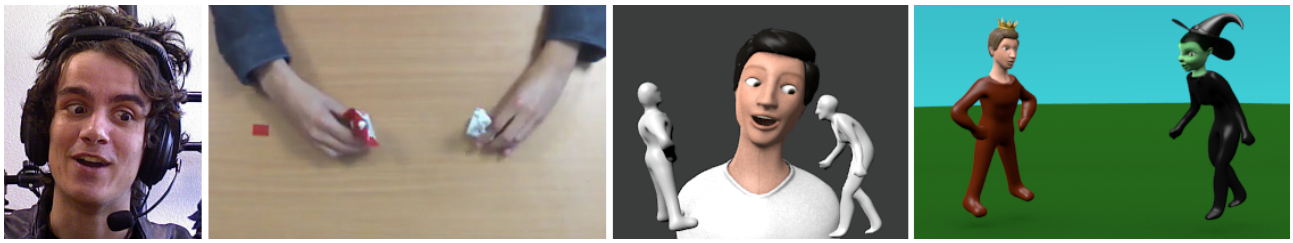


Figure 1.4: FIGURINES animation transfer system. Using two Kinect cameras, one extracting the narrator's facial features and one which combines with IMU's sensors placed in the figurines tracks their positions and orientations. Our system determines at each frame which character the narrator embodies and transfer his facial motion into the corresponding figurine.

Addressing the unsolved challenges of the FIGURINES system, we propose

a method to draft 3D character expressive animations using physical props like the one used in the FIGURINES system and extend the motion doodle paradigm. As our main recording device, we make use of the virtual display offered by the HTC Vive and of the accuracy of its controller tracking system to create animation sequences from the recorded trajectories at 60 fps rate. We record the motion of the controllers as *spatial motion doodles*, ie. trajectories of 6D rigid frames over time, and parse them into a sequence of actions which are translated into animations of articulated characters (see figure 1.5 and 1.6). This is done for the two controllers at the same time, allowing users to make characters perform synchronized actions and interact.

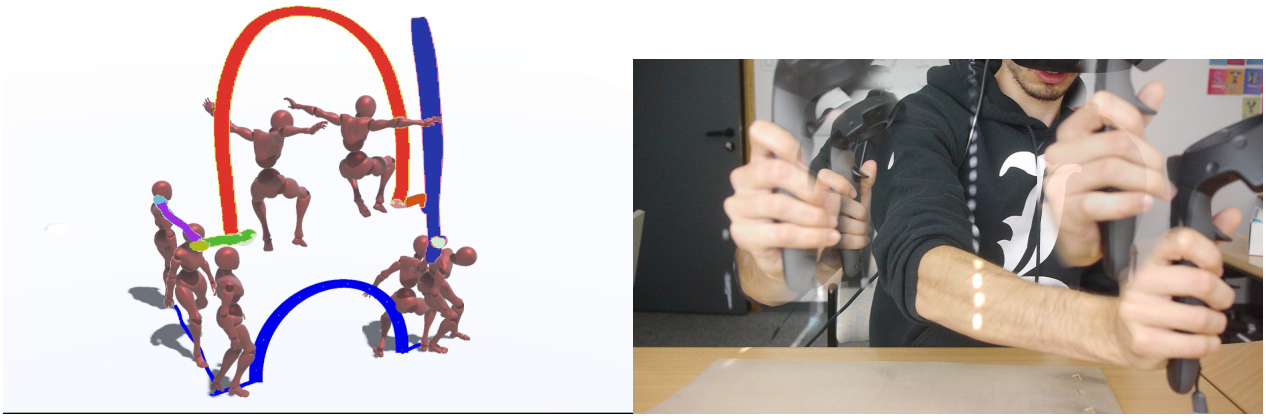


Figure 1.5: Main single character performance transfer example: the user (right) control a physical props whose 6D trajectory is recorded and transformed into an expressive character animation (left) following an adapted path computed from the input curve. This curve is shown on the left as colored segments corresponding to recognized actions performed by the user.

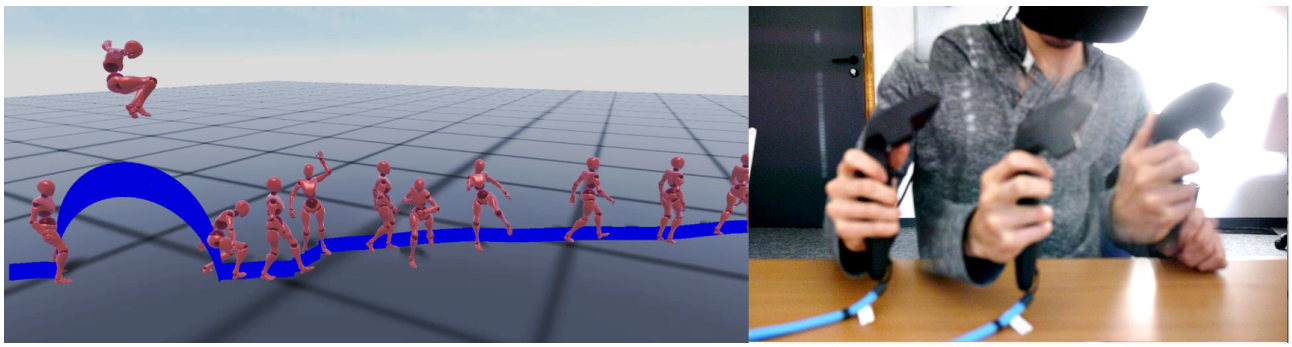


Figure 1.6: Another example of our performance transfer animation sequence output.

1.3 Overview

Our work is an intermediate method between sketching and acting animations, where we focus on creating animation sequences directly from spatial motion doodles extracted from the HTC Vive controllers motions. We are interested both in retrieving action sequences and motion qualities from these space-time trajectories aiming at creating expressive animation sequences. This

process generalizes for two characters plays, retrieving both independent action sequences as well as interactions between characters, allowing users to created animated stories featuring at least two characters.

1.3.1 Contributions

Our work presents several contributions in the field of action and interaction recognition from hand gestures, motion qualities recognition, procedural expressive animation generation and animation sequence generation.

Hand Gesture Action Recognition. In this work, we present a new position, rotation and scale invariant pattern extraction method used to detect actions. Our gesture recognition algorithm behaves like a compiler which operates on a vocabulary of motion tokens representing linear and angular velocity direction changes. Both character actions and interactions can be recognized from motion token regular expressions and from preconditions (especially for interactions). Finally, we also introduce a trajectory to regular expression learning algorithm, based on user training examples, enabling them to define their own gesture during the recordings. We conducted several user-studies to validate our action representation both on single action recordings and on action sequences. We also conducted a study evaluating the efficiency of our regular expression learning algorithm. This contribution is detailed in chapter 4 and 6.

Hand Gesture Motion Quality Recognition. Our work also aims at capturing user emotional intent through hand motion quality analysis. This expressiveness is interpreted as Laban Effort qualities for which we devised two classification algorithms. Using a carefully selected set of geometric features, we introduce a Naive Bayes classifier which recognizes Laban Efforts separately and which is further enhanced using feature selection for each Effort. We also introduce a Hidden Markov Model based classification algorithm which is best suited to classify temporal sequences like spatial motion doodles. We further discuss the classification results of both algorithms as well as the training examples and geometric feature choices. This contribution is presented in chapter 4.

Procedural Expressive Animation Generation. We introduce a new procedural Laban Time and Weight Efforts stylization algorithm which relies on animations segmentation into their five stages and on three animation modification operators, namely: motion scaling, retiming and reshaping operators. Those operators are independent and can be used on any skeletal animations. We conducted several users studies aiming at evaluating the efficiency of our

stylization with respect to neutral and gesture-retimed animations. This contribution is detailed in chapter 5.

Animation Sequence Generation. We propose a new method for assembling character animation sequences along the path of the input trajectories with suitable Laban motion qualities. We describe a method for computing appropriate trajectories that characters will follow from the input doodle. We also show how we compute smooth transitions between each animation based on well-known existing algorithms and on the animation stage decomposition. Additionally, we discuss several methods on how to compute these animations stages automatically. Finally, we show user experiments we conducted in different use cases, in particular in multi-character narrative plays. This contribution is presented in chapter 5 and 6.

1.3.2 Outline

This thesis is organized into seven chapters. Chapter 2 gives a review of related work, chapter 3 describes the experimental setups we built this during thesis. We then organize our contributions into three chapters: performance analysis, single performance transfer and storytelling performance transfer.

Chapter 4 describes a hand gesture study by first introducing our action recognition system as well as our regular expression learning algorithm. We then discuss the choice of geometrical feature sets we will use for performing Laban Effort classification before introducing our two classifiers, discussing and comparing their classification results.

We then detail, in chapter 5, how we compute adapted trajectories from user hand motion and how we altogether modify and sequence animation clips along them. We also introduce new procedural Laban Effort modifiers and discuss their efficiency through different user studies. For chapters 4 and 5 we will use figure 1.5 as our guideline example.

In chapter 6, we describe how our tool can be used to handle multiple characters at a time for creating animated stories. We further explain how we process interactions and context related behaviors.

We finally conclude and discuss future research directions in chapter 7.

Publications

1. Portaz Maxime, Garcia Maxime, Barbulescu Adela, Begault Antoine, Boissieux Laurence, Cani Marie-Paule, Ronfard Rémi and Vaufreydaz Dominique (2017).Figurines, a multimodal framework for tangible storytelling. *WOCCI 2017 - 6th Workshop on Child Computer Interaction at ICMI 2017 - 19th ACM International Conference on Multi-modal Interaction.*

2. Barbulescu Adela, Begault Antoine, Boissieux Laurence, Cani Marie-Paule, Garcia Maxime, Portaz Maxime, Viand Alexis, Heinish Pierre, Dulery Romain, Ronfard Rémi and Vaufreydaz Dominique (2017). Making Movies from Make-Believe Games. *6th Workshop on Intelligent Cinematography and Editing (WICED 2017).*

3. Barbulescu Adela, Garcia Maxime, Begault Antoine, Cani Marie-Paule, Portaz Maxime, Viand Alexis, Dulery Romain, Boissieux Laurence, Heinish Pierre, Ronfard Remi, Vaufreydaz Dominique (2017). A system for creating virtual reality content from make-believe games. *2017 IEEE Virtual Reality (VR).*

4. Garcia Maxime, Ronfard Rémi and Cani Marie-Paule (2019). Spatial Motion Doodles: Sketching Animation in VR Using Hand Gestures and Laban Motion Analysis. *Motion, Interaction and Games 2019 (MIG '19).*

Chapter 2

State of the Art

Authoring animation using sketches or physical props is not a new idea; our work follows the line of several existing works we will describe in the next sections. Moreover, this work took inspiration from Motion Doodle [Thorne et al., 2004] which propose an intuitive way of creating animation sequences using 2D sketches pattern matching. In this chapter, after introducing background related to expressiveness representation and inferring models, we review state-of-the art methods for authoring animations and recognizing and inferring expressiveness into input animations.

2.1 Background

Our work, as well as state of the art methods for recognizing and inferring expressiveness, require that we introduce definitions and concepts before going into further details. In this section, we introduce how motion expressiveness can be represented through different representation models and how then can be transcribed into animations. Moreover, we will introduce the Laban Motion Analysis which is commonly used to describe and qualify motions.

2.1.1 Emotion Based Models

As emotions are highly relevant expressive representations which are known to everyone, their are quite often chosen to characterize motions. They also constitute a large vocabulary of expressiveness descriptors. Six of them are often used for dramaturgic exercises as they convey very diverse feelings and are referred as basic emotions as defined by Ekman ([Ekman, 2003]): wrath, disgust, fear, joy, sadness and shock. While being simple and understandable by everyone, this model is not quantitative and only represents emotions in their strongest form, moreover it cannot represent "a little bit/very angry" motions. Addressing this problem the Russell Circumplex Model (RCM) [Russell, 1980] represents emotions in a 2D continuous space along the valence and arousal axis. When stylizing animations, this representation has the advantage of excluding

interpolations of completely opposed emotions which cannot be combined in practice to the benefit of alike emotion interpolations. While being very accessible in the sense that animation experts as well as non experts can label animations with emotions, they limit the description of motions as they qualify them in an overall fashion. This limitation led researcher to choose lower-level motion quality representations to stylize their animations such as Laban Motion Analysis which we will now introduce.

2.1.2 Laban Motion Analysis

Laban Movement Analysis (LMA) is a method for characterizing quality in human movements in terms of timing, space used, posture and intention. It was introduced during the 1940s by Rudolf Laban ([Laban, 1950]), a Hungarian choreographer. He introduced five categories to fully describe and classify motions, namely Effort, Shape, Phrasing, Body and Space. We will now define each of these categories.

Effort describes the dynamic quality of motion and is based on the 4 following dimensions (with the associated extreme values): Space (Direct - Indirect), Time (Sustained - Sudden), Weight (Light - Strong) and Flow (Bound - Free) making the Effort category isomorphic to $[-1; 1]^4$.

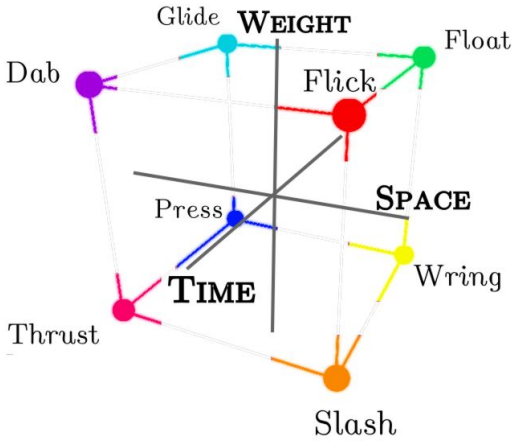


Figure 2.1: Cube representing Laban's Effort category: the three axes are Space, Time and Weight. Each corner is an association of the 3 effort parameters.

Expert studies on Laban effort expression [Bishko, 2014] observed that Space, Time and Weight are often sufficient to describe actions, either two or three of them being active at the same time. Motions involving one descriptors are defined as Factors (characterized by two opposite Elements), those involving two descriptors are defined as States and those involving three of them as Drives. Figure 2.1 illustrates the Laban Effort dimensions of Space, Time and Weight, omitting the Flow axis. In this work, we study the hand gesture motion qualities for the three Laban Effort but further restrict the animation quality transfer to the two dimensions of Time and Weight, which best capture the input gesture styles (see section 4). This restriction results in a 2D representation of the Time/Weight space represented in figure 2.3.

Shape characterizes body posture during motion. This feature directly influences the movement's convex hull. Similarly to Effort, it is isomorphic to a $[-1;1]^3$ cube and evolves according to 3 factors: Horizontal (Enclosing - Spreading), Vertical (Sinking - Rising) and Sagittal (Retreating - Advancing) as shown in Figure 5.8.

As reported in [Durupinar et al., 2016] and [Bishko, 2014], Effort and Shape can be combined together, as each feature from the first category has expressiveness affinity with the second. For maximal expressiveness, [Bishko, 2014] suggests to associate Light effort with a Rising shape, Strong effort with a Sinking shape, Sustained effort with an Advancing shape and Sudden effort with a Retreating shape. In chapter 5, we use this coupling between Effort and Shape to emphasize Weight (Strong/Light) and Time (Sustained/Sudden) qualities by changing the shapes of the 3D character in the corresponding animations.

(a) Preparation (b) Anticipation (c) Execution (d) Follow-Through (e) Transition

Figure 2.2: The five main steps of motion: preparation (intention of action), anticipation (energy gathering), execution (energy release through main action), follow-through (follow-up motion induced by the execution), transition (movement either back to hold pose or to connect to next action).

Phrasing describes the relative durations of the five main stages in human movements (Figure 2.2). Among them, *Preparation* is when the character mentally prepares to execute the movement; *Anticipation* is an energy accumulation phase during which the character moves in the opposite direction from the main, subsequent motion; *Execution* corresponds to this main motion; *Follow-through* represents all the extra movements at the end of the main action; *Transition* is the end stage, which leads either to a rest pose or to another action. In practice, some of these stages may be removed or masked depending on the motion and its context.

Body describes which parts of the character are involved in the motion as well as how different body parts motion relate to each other. Moreover, body parts motion structural organization is described through six patterns: Breath, Core-Distal, Head-Tail, Upper-Lower, Body-Half, Cross-Lateral. Breath suggests a continuous motion connecting all body parts with the rhythm of our breath; training motions like breath exercises and Katas exercises practiced in combat

sports follow a Breath configuration. Core-Distal characterizes motions linking the chest or pelvis to the extremities of the body. Head-Tail involve mainly the spine from which motion is transmitted from one extremity to the other. Upper-Lower characterizes which part of the body maintains the overall balance. Body-Half describes configurations in which one side of the body(left or right) is free to move while the other maintains stability. Finally, Cross-Lateral characterizes motion in which left and right sides are moving in opposite directions.

Space describe the 3D space covered by a motion. It can be interpreted as the 3D convexhull of the body a given moment of a motion and as the overall body convexhull during the entire motion.

Those five motion descriptor categories are quite exhaustive as they qualify motion dynamics, timing, global posture, motion phases, motion's space coverage, body parts motion links and so on.

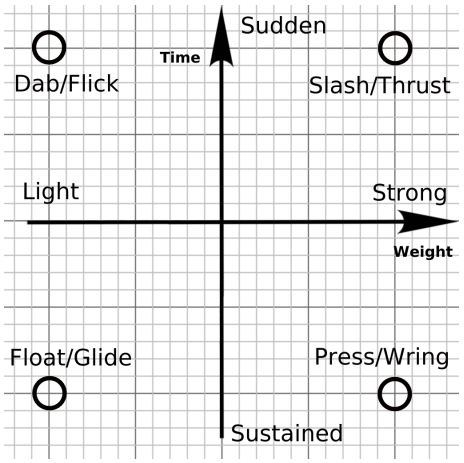


Figure 2.3: Laban's Effort Drives projected in the Time and Weight axes.

Challenge: LMA is a theory used by choreographers who convey dancing directives and advises, using it as a dancing vocabulary. Converting this theory into a mathematical representation is still quite a challenge that has been tackled by researchers in the character animation field but remains open.

In our work, we use Laban Effort, Phrasing and Shape categories to classify and infer expressiveness into animations.

2.1.3 Animation Style Transfer

As we will discuss further in section 2.5.2, animation style transfer can be operated through two different kind of techniques: procedural stylization, which applies modification rules to each skeleton joints during the whole animation with respect to an input style, and data-driven stylization which relies on a database of already stylized animations to either interpolate and extrapolate between styles or apply a given style to an animation which is not part of the database. While procedural stylization algorithms are quite diverse, most of the data-driven stylization methods relies on regression models for interpolat-

ing and extrapolating between styles. Let us explain the general principle of regression-based models and their uses in our animation case study.

Lets consider an expressiveness space $P = \mathbb{R}^n$ and a set of labeled stylized animations A_i, S_i. Lets assume that the provided styles are "extremes" among the expressiveness space. Given an animation A , n stylizations are stored and represented by the $(p_0, .., p_n) \in P^n$ points. For each style we know the value of each skeleton joint DOF θ at frame t . As we would like to interpolate and extrapolate between styles for the A animation we have to compute a regression model meaning that for each joint DOF we want to find a function f such that $\forall p_i, f(p_i) = \theta_i$ and which interpolates/extrapolates "well" in the continuum of P. Multiple Linear Regression is often chosen as it is fast to compute and its efficiency can be controlled by choosing basis regression functions. It is usually formulated the following way $\forall i \in [1, n], \theta_i = \sum_{j=1}^{p} \beta_j \phi(p_i, p_j)$, which is a linear combination of parameters β_j weighted by some function basis ϕ that might dependent of the input points (which is the case in our example). The goal of the regression is to find the values of β_j minimizing the square distance between the θ_i and the related value computed with the regression model which can be expressed as a Least Square optimization problem $\min_{\beta_j}(\sum_{i=1}^{n}(\theta_i - \sum_{j=1}^{p} \beta_j \phi(p_i, p_j))^2$. Choosing the notations $\Phi = \{\phi(p_i, p_j)\}$, $\beta = \{\beta_j\}$ and $\Theta = \{\theta_i\}$, the solution to this problem (computed through nullification of the partial derivative) is $\beta = (\Phi^T \Phi)^{-1} \Phi^T \Theta$. Finally, the last unknown to remove is the choice of the basis function *phi* which depends on the input problem that has to be solved. A typical choice is the Radial Basis Functions which are functions depending on a metric $||.||$ such that $\phi(p, c) = \phi(||p - c||)$. In our animation study case, Gaussian RDF are a common choice $\phi(p_i, p_j) = \exp^{-(\epsilon||p_i - p_j||)^2}$. It is worth mentioning that as a regression model has to be computed for each joint DOF at each frame, the computation time can be quite high that is why some work fits higher level properties which can be global over time or compute much smaller auto regressive models for instance.

In this work we introduce a new procedural method to infer Laban Time and Weight Efforts to input animations as described in chapter 5.

2.2 Spatial Interaction

While animators create 3D animations using professional software, repetitively clicking on joints and modifying their values, methods involving spatial interactions for animation creation have been a topic of research for years.

Moreover, motion capture which is now a standard for producing video games and CGI movies has been widely studied and is still a topic of interest in research. In this section, we will review two kinds of spatial interaction techniques, namely puppet manipulation methods and body-based acting methods which transfer human body motion into 3D animations.

2.2.1 Puppet Manipulation

The motivation behind our work is to create 3D character animation by playing with figurines or toys. This idea has been the focus of a wide range of puppetry systems. Moreover, the use of tangible interfaces has brought very different virtual manipulation paradigms [Ishii and Ullmer, 1998].

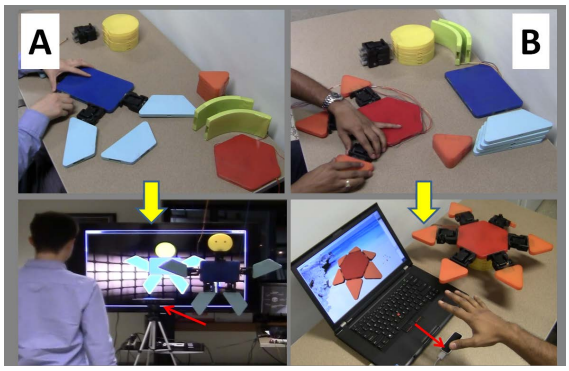


Figure 2.4: PuppetX ([Gupta et al., 2014b]) framework. Users first design their articulated puppets. A virtual replica of the assembled puppets is automatically created and is manipulable via its physical version.

One popular puppetry paradigm is the Magic Mirror metaphor we described in chapter 1. Character joint manipulation systems like [Oore et al., 2002], [Jacobson et al., 2014] or [Gupta et al., 2014b] allow users to intuitively pose characters with immediate feedback as the tracked physical props represent parts of the manipulated body. In their approach [Oore et al., 2002] allow users to animate 3D characters by layers using bamboo tubes equipped with 6 DoF sensors whose motion are mapped to a character's joint. The user successively animate all character joints while being provided with immediate visual feedback of the modifications he brings. In a similar way, [Gupta et al., 2014b] introduces a system for creating articulated puppets together with a related virtual avatar which possesses the same degrees of freedom as its corresponding physical prop. Likewise, the user is provided with visual feedback in which the virtual puppet's motions are coordinated with the physical one's. [Slyper et al., 2015] also propose an interface for animating both a robot and a related virtual character by recording keyframes of the robot limb motions tracked using a webcam and whose articulations are controllable from their software.

More recently, [Jacobson et al., 2014] designed both the hardware for a physical skeleton system constituted of pluggable angular sensors and an algorithm to relate this skeleton to a 3D mesh, either computing automatically a rig with appropriate weights or mapping the virtual representation of the physical skeleton to an already existing rig. The user can then manipulate the characters'

articulations using the physical skeleton at real-time frame rate either for posing or animating. Compared to previously introduced methods, [Jacobson et al., 2014] generalize to any kind of tree like skeletons with any underlying mesh. However it is still difficult to animate characters directly in a real-time performance manner as there are many DoFs to manipulate at a time.

Using a different approach [Nitsche and McBride, 2018] designed a virtual puppetry system, allowing user to manipulate three types of marionettes which are simulated in a virtual reality environment and controlled using HTC Vive controllers. In this system, users can switch between string, rod and hand marionettes.

While all previously presented systems were focused on animating articulated characters, some systems only transfer puppets rigid motion to the benefit of manipulating several of them at the same time. [Held et al., 2012] 3D puppetry system transfers rigid motion of figurines into corresponding virtual avatars in real-time using a Kinect sensor. Using the Kinect, the system first creates a scanned 3D mesh of the figurine used during the play while also texturing it. Then, during the performance, the system is able to recognize which figurine is being used and determines its current position and orientation by computing image-based correspondences. They demonstrate the efficiency of the Magic Mirror paradigm by allowing users to make montage from successive recordings. Closely to [Held et al., 2012], [Gupta et al., 2014a] designed a montage system which also transfers physical props rigid motion into the 3D virtual world. The system allows the user to record several takes of a given sequence and interactively rate his satisfaction over time for each takes. The system finally compute an optimal montage of the takes with respect to the user rating.

Some systems also uses Augmented Reality like paradigms to improve user immersion for animation content creation. [Ziola et al., 2011] designed an interactive system which let users play with LEGOs while extra elements are added to the real world using projectors. Moreover, their system, LEGO Oasis, is equipped with a camera performing LEGO and gesture recognition adding content like tracks on the playground table as well as playing animation when triggers occur like fire truck spreading water on a building which is on fire. This approach is quite interesting as it let the user freely improvise stories while augmenting the immersion with non invasive projections. While this system does not create 3D animation directly, it could be used to compose and mix them in a physical playground, like an Augmented Reality system could do.

[Sreenivasa et al., 2009] proposed an interesting approach suggesting that a robot can be controlled by only moving its head. Their demonstrated their

technique for walking, crouching and standing actions and for which the robot makes its move when its head leaves a control circle and is able to determine which foot to move. This technique can be translated to 3D virtual characters by providing the user a mean to control its head (with a tracked rigid object for instance) and make the character behave the same way as the robot described in their work.

As mentioned in chapter 1, we also designed designed a puppetry system [Barbulescu et al., 2017, Portaz et al., 2017], as on offline paradigm system which lets user freely improvise stories with two IMU's tracked figurines whose rigid motion is transferred into the virtual world while the narrator facial motion is re-targeted to the figurine he embodies at each frame.

On a side note, it is worth mentioning that creating animated content is now also a topic of interest for companies which invest in professional tools such as Quill [1] or MasterpieceMotion [2] to animate characters and other objects in VR.

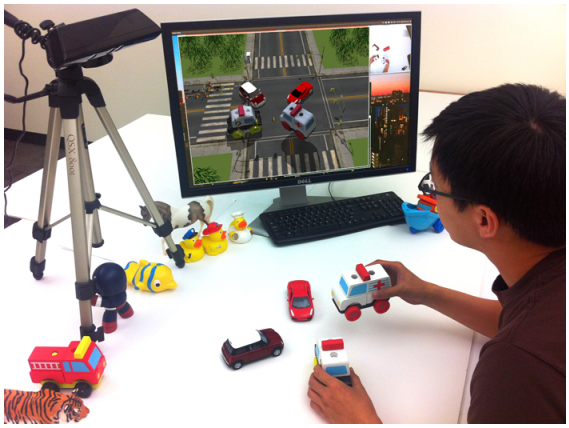


Figure 2.5: 3D Puppetry recording session. The user first scanned the figurines using the Kinect sensor to create corresponding textured 3D meshes. Figurines are then tracked in a real-time fashion using image-based feature correspondences and their rigid motion are transferred to the corresponding 3D models. We can notice that the user is provided with real-time feedback which is useful for making animation layers.

We reviewed an non exhaustive list of puppetry systems which are focus either on animating articulated characters and manipulating one at a time or on the contrary some of them transfer only rigid motions of tracked puppets but allows users to control several tangible objects. We can also highlight the fact that a majority of those system are using the Magic Mirror metaphor which raises the question of user preferences against offline recordings. In this thesis, we record HTC Vive controllers and tracker as an offline process, users can only see the result of their manipulation after recording.

2.2.2 Acting Transfer

Creating animations using registered human body motion has two main advantages, it produces very realistic animations as they come from real motion data and once the acquisition system is set up users can create a wide range of animation in a very short time.

[1] Quill VR by Facebook: https://quill.fb.com/
[2] Masterpiece Motion: https://www.masterpiecevr.com/motion

[Dontcheva et al., 2003] was one of the first work allowing users to create and edit motion captured animation at interactive rates. The main innovation of this work featured a layered animation system in which the user can refine part of the body animation at a time using different mapping strategies. Three mapping paradigms are available: absolute mapping which is strait forward from actor to character motion, relative mapping editing joint motion with respect to its parent motion, which is very useful to edit character body part of complex poses, and additive mapping which, as it suggest, adds the actor motion to the already existing animation.

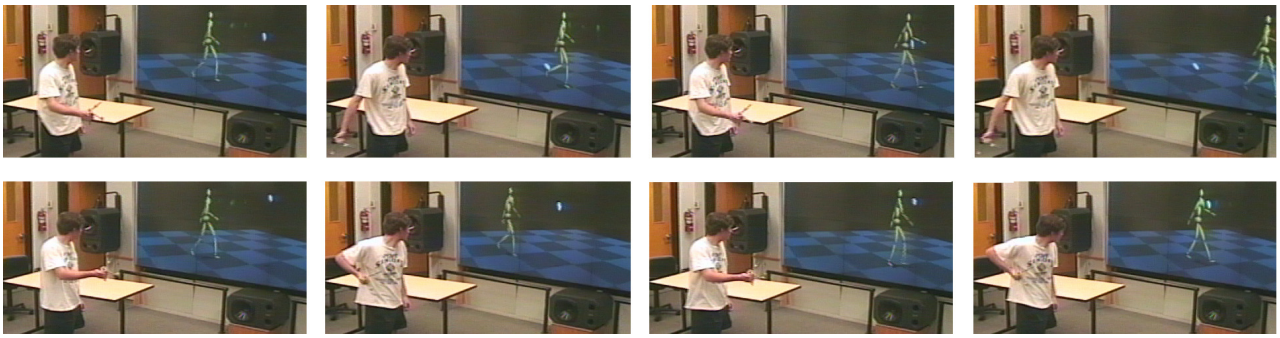


Figure 2.6: [Dontcheva et al., 2003] layered animation acting, after recording the whole body motion, the user re-act the right arm animation over the previously recorded animation.

Most of the motion capture tools are used for strait forward mapping from human or animal registered motion to a highly similar or identical skeletons. On the other hand, some work focus on skeleton re-targeting applications which allows users to animate characters with highly different skeletal structure and morphologies. [Leite and Orvalho, 2012] proposes a Kinect tracking based system which maps users skeleton to shadow puppets, i.e. silhouette of a character from a single point of view, of various kind like dragon, humanoid or birds. Users can manipulate these puppets with their bodies adopting more front or side view motion depending on the puppet they manipulate. The system performs skeleton re-targeting allowing user to control the dragon tail with their leg for instance. As a side note, [Lu et al., 2011] also focuses on animation creation using shadow puppets in which users design and control them using two wireless sensors.

More recently, [Lamberti et al., 2019] designed a multimodal character control system which uses three inputs: user's body motion, user voice and scripts. Body motion tracking is used to perform action recognition which is then interpreted by the system and converted into a character animation. As the controlled character is not necessary human, this method avoid animation transfer problems that can result from re-targeting. Animations can also be trigger using voice recognition, which can be used for motions which are difficult to mimic in the recording space. Finally scripts are directing the goal of the character, imposing its trajectory for walking motions for instance.

2.3 Motion Quality Recognition

Motion qualities are feature traits of motions that are independent of their function or objective. Throwing an object can be done in various intents while satisfying its goal. Finding the motion qualities of a given gesture is actually an ill-formed problem as it is dependent on the quality space we want to place it in. In section 2.1.2, we introduced several motion quality spaces like the basic emotion space defined by Ekman ([Ekman, 2003]) or the LMA categories and pointed out that these representations lacks of formal definitions and that finding a formulation that generalizes for all kind of motions remains a difficult task. In this section, we review motion quality classification methods which relies on an analysis of geometrical properties of the body joints motions.

[Senecal et al., 2016] designed a system able to continuously recognize actors emotions as expressed in the Russell Circumplex Model solely based on the body motion (containing speed, acceleration and jerk of the different body joints). To achieve their goal, they trained a 10 layers neural network which outputs RCM coordinates using LMA properties as input which are extracted from their training data of professional actor performing different gesture in all different emotions.

In their work, [Bouchard and Badler, 2007] aim at segmenting motion capture data with respect to motion LMA properties. In their work they focus on recognizing Laban Effort qualities and use them to perform motion capture data segmentation. Each effort is quantified using one neural network (four in total) which was feed with kinematic feature such as velocity, acceleration, curvature and torsion of the right arm and the sternum. Their segmentation results shows that LMA effort are a good candidate for semantic representation of all kind of human motion.

In addition, work like [Camurri et al., 2003] devised a framework (Eyesweb) which account for multimodal inputs such as motion and music to classify expressiveness intent. Moreover, they introduce a method to design expressiveness classification algorithm from multimodal data using several abstraction layers from low-level features to high-level representations. They demonstrate the efficiency of their system in the context of Laban Effort classification on dancer motions retrieved from video data. Like [Senecal et al., 2016], they use a full body motion data to perform Laban Effort classification. Interestingly, they use the duration of the pauses in the motion as well as tempo changes as features to recognize Sudden from Sustained motion which are quite different from other Effort classification systems as their are directly linked to a more musical representation of the motion.

While [Bouchard and Badler, 2007] performs Laban Effort classification using features of the right arm and the sternum, some other work like [Fdili Alaoui et al., 2017] or [Mentis and Johansson, 2013] focus on achieving the classification using only data from the hand motion, which is really close to our case study (see chapter 4).

Moreover, using a Kinect sensor to track the wrist motion, [Mentis and Johansson, 2013] proposes to classify Light against Strong motion depending on the wrist position with respect to the hip (below means Strong while above means Light). The Time Effort is recognized by identifying acceleration peaks for Sudden motions against continuous velocity for Sustained motions. Finally, Space classification is achieved by computing the deviation from the segment defined two points in the wrist trajectory. Their approximation approach was evaluated by a LMA expert and was pretty efficient in their configuration (65% of successful classification) which does not generalize to all kind of motion (if all wrist actions are done above the hip, it compromises the Weight classification for instance). One interesting finding is that the expert sometimes could not identify one of the effort for a given motion and often quantify each effort rather than strictly classifying them, which suggest that it may be useful to add a Neutral category to each Effort classifier.

In a similar case, [Fdili Alaoui et al., 2017] study the effect of three different type of feature on the classification rate: positional data represented by the distance between the arm and the chest, dynamic data which corresponds to the norm of the jerk of the wrist and physiological data represented by muscle activation around the wrist. The intuition behind those features is that position data will help classifying the Space axis, dynamic data the Time axis and physiological data the Weight axis. To evaluate the influence of each feature, they ask a LMA professional to performs motion in different Drives and trained 5 different HMM: three of them contains data from a single feature only, the forth combines all the features and the fifth one also adds speed and acceleration. Their classification results highlight several facts especially relevant for us: the positional does not help dissociating Direct from Indirect motion while jerk is highly relevant for classifying in the time axis and physiological data is weakly relevant for Weight classification. However, the combination of all the features (with acceleration and speed) performs better in recognizing Drives but the recognition rate remains around 25% which shows that Drives classification based on hand motion only is a difficult problem.

In this thesis we contribute to address this challenge by considering a wider set of features and by performing feature selection which improves classification accuracy (see chapter 4).

2.4 Sketching Animation

While creating 3D models using sketches has been explored for years, it also has been explored for animation creation and edition. More specifically, sketches have been used both for posing and animating characters' body parts as well as creating full animation sequences.

2.4.1 Character Posing

Several works involve posing characters using sketch. Moreover, the work of [Guay et al., 2013] has been quite innovative, introducing a method for posing characters using Line Of Actions (LOA). The LOA refers to the line going through two extremities of the body, called a bodyline, which best represent the overall posture at a given moment of a motion. The main idea of this work is to make bodylines fit the different LOA drawn by the user and repeat this operation several times so that one can pose the character at a given keyframe of an animation. This technique is based on a bone chain to LOA mapping where bones joints angles are modified from root to leaf in order to fit the LOA as much as possible. It reveals to be efficient and intuitive to pose characters in a way that experts as well as beginners can use it at their level. It is notably quite useful for posing characters with long bodyline like dragons or part of characters like tails. More generally it is very efficient for posing long bodylines which are more difficult to handle using the traditional editing way.

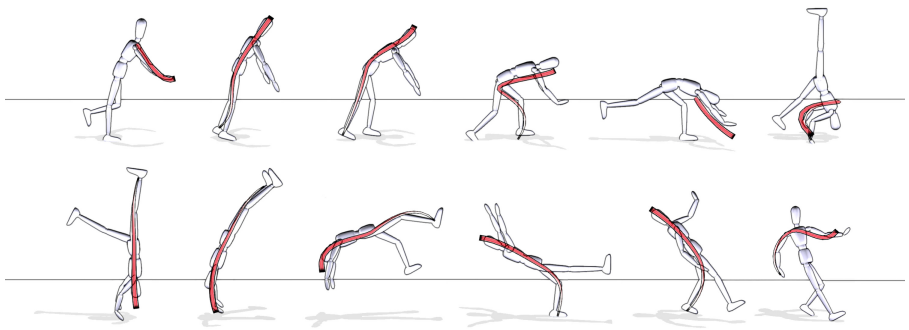


Figure 2.7: Line Of Action [Guay et al., 2013] animation posing example. Using a single view in this case, the user selects characters' bodylines and draw LOAs for editing their poses. Once the user is satisfied with the current pose, he saves it as a keyframe of the animation.

Using a different approach [Choi et al., 2012] designed a system retrieving 3D posture from stick figure drawing searching for a corresponding 3D skeleton pose in a database of animations. Unlike [Guay et al., 2013] this posing method is limited to the animation present in the database and therefore can restrict the artistic expressiveness, it can however be used as a rough posing method. In a similar fashion [Bessmeltsev et al., 2016] propose an interface to pose a rigged 3D model using 2D vectorized gesture drawings of the underlying model. As opposed to [Choi et al., 2012], this technique relies solely on geometric analysis

of the 2D sketch and computes pose correlations with the 3D skeleton and each of its bones influence on the model.

2.4.2 Animation Creation

Extending the LOA method for creating animations [Guay et al., 2015] also proposed a sketch-based method to make characters bodylines follow space-time trajectories allowing users to create character animations. Moreover, those space-time curves named dynamic line of actions (DLOA) animates joints positions, making them following the sketch curve at the same speed as it was drawn. As a consequence each joint of the animated bodyline is sequentially stretched during fast sections before going back into its default size, inducing a cartoonish style into the resulting animation. In addition, the user can impose LOA constraints along the generated DLOA, allowing him to pose the character at specific moment of the animation while the animated bodyline moves along the DLOA. While this method as been proved usable on characters having only one principal bodyline like the dragon in Figure 2.8, it has yet to be extended to human-like characters which have very different moving behaviors and shorter bodylines.

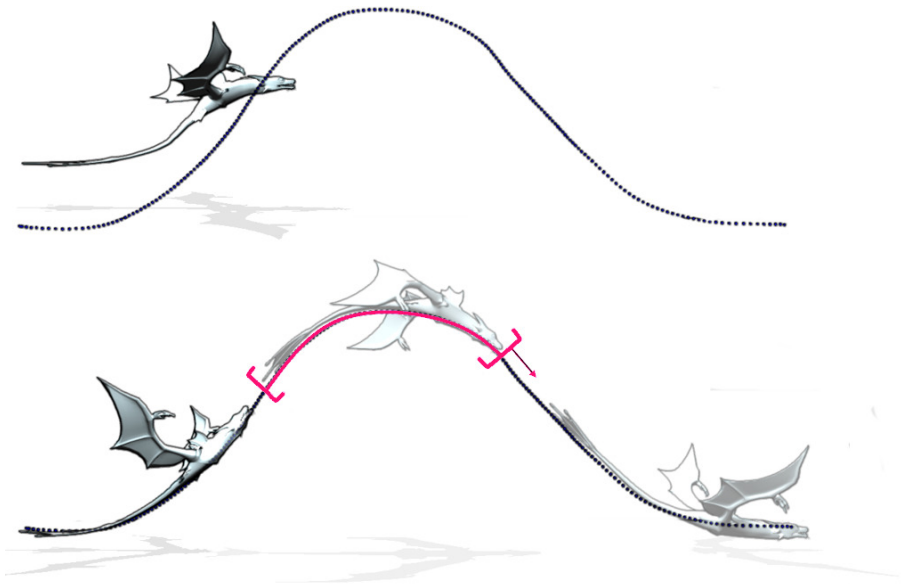


Figure 2.8: Dynamic Line Of Action animation sketching [Guay et al., 2015]. The user draws a space-time curve which is then followed by the character's bodyline. As some joints of the bodyline might move faster than the others it induce squash and stretch behaviors. The user can also refine other parts of the body using line of actions at any frame.

On the other hand, [Ciccone et al., 2017] proposes a method to animate motion cycles using several space-time curves examples sketched by users. Moreover, animating one bone or IK controller after an other, the system compute a mean cycle based on a computed average of the training examples. This computation is based on a curve descriptor [Mori et al., 2005] which is used to make correspondences between each point of the training curve and compute

the average cycle.

2.4.3 Animation Edition

While previous work focused on creating animations from scratch using sketches, some techniques were introduced to efficiently and intuitively edit existing animations using 2D sketches.

[Terra and Metoyer, 2004] proposes a system for editing animations timing using sketch curves as retiming reference. Moreover, the system first find correspondence between the sketched curve and the animated object motion path, then it applies user input timing to the input animation using curve chunk correspondences.

[Choi et al., 2016] proposes a complete sketch-based animation edition system. Moreover this system propose to directly edit bones trajectory using three different paradigms: global space, local space as well as dynamic space edition. The third one is particularly interesting as it defines a new way of editing animation sequences in which animated objects stays relatively in place. Moreover, dynamic space allows motion paths editing in the camera plane: motion paths are stretched and projected into the camera plane making the horizontal axis representing time while the vertical axis represents the values. This feature is especially efficient for editing pelvis trajectory of characters as well as extremities of the body.

More recently [Ciccone et al., 2019] proposed a new sketch-based animation edition system which aims to edit animation using only FK. More precisely, as IK interpolation can be difficult to control, they propose a new method for professional animators to edit animation motion paths using space-time curves which compute for each key an optimized animation curve in terms of values and tangents (assuming that animators work with Bezier curves). This workflow is quite innovative as it allows animators to animate their character without using IK to create trajectories and only use it for posing or for applying strong position constraints. Moreover, this method reduce the size and complexity of animation rigs used to pose characters and thus significantly decrease the rig evaluation time.

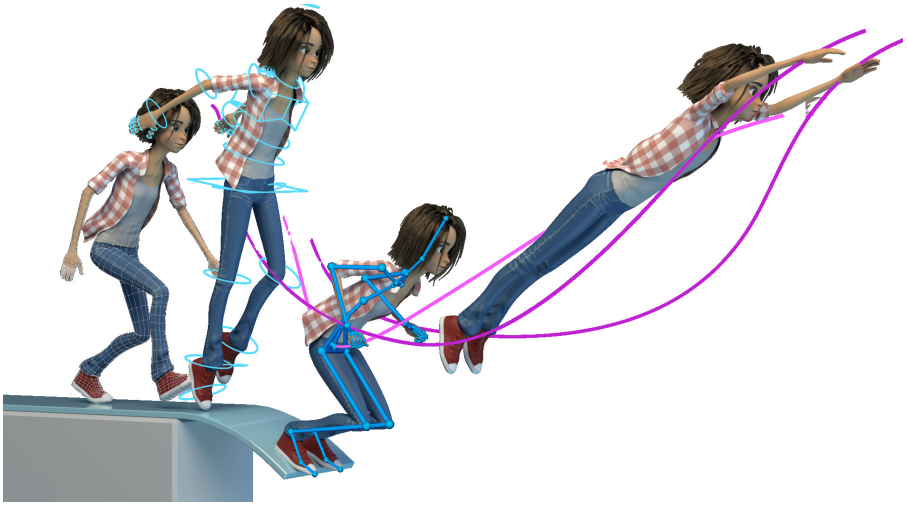


Figure 2.9: In [Ciccone et al., 2019], users can modify joints trajectories while their system optimize animation curve in a FK fashion to match the new trajectories.

2.4.4 Authoring Animation Sequences

While all previously described methods can easily be used to create or edit parts of an animation they do not tackle the problem of creating an entire animation sequence.

Although few sketch-based methods try to solve this problem, the method proposed by [Thorne et al., 2004] addresses this issue introducing the concept of motion doodle which we will extend in this thesis. The idea behind motion doodles is the following: given an input 2D stroke, it is segmented in terms of directions between horizontal, vertical and obliques chunks. Segmentation points are found using corner detection algorithm [Chetverikov, 2003].Then, directions are computed with respect to the value of the angle between two corners and the horizontal direction. As shown in figure 2.11, this decomposition classify stroke directions into 6 bins which are assigned to a token. Then, each animation stored in the database is attributed with a regular expression in terms of these direction tokens. Finally each input strokes is converted into a token sequence which is parsed in the chronological order, converting the input stroke into an animation sequence. Each generated animations are parametrized with respect to the duration of the stroke as well as its start and end point. Animation such as jumps, also take the apex of the stroke as input for trajectory beautification. Each generated animation then follows an ideal trajectory which is generated thanks to these input parameters. It is important to note that although this technique being bound to 2D motions, depth can be inferred by projecting detected corners to the ground after the sketch interpretation step. In chapter 4, we further extend this technique in 6 dimensions and make use of a moving frame to extend the efficiency of this technique.

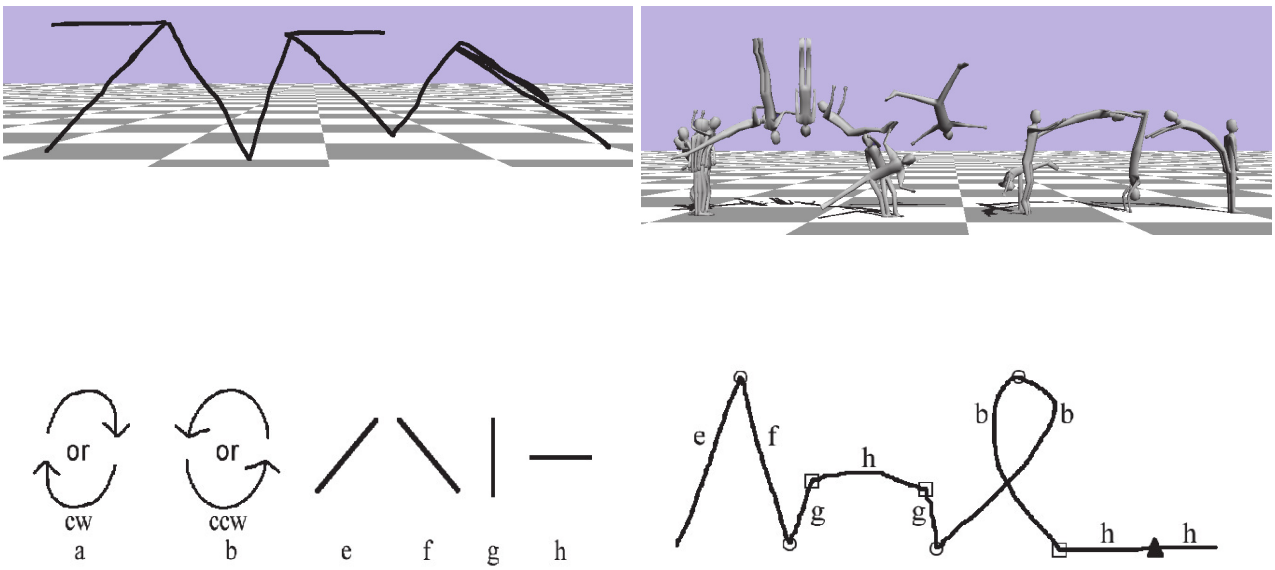


Figure 2.11: Motion Doodle example [Thorne et al., 2004]. Users sketch 2D patterns which are segmented and interpreted as an animation sequence. Each animation is defined as a regular expression of an underlying representation of the curve in terms of directions tokens.

[Hyun et al., 2016] also tackled the challenge of creating animation sequences from sketch. More precisely, the goal of the work is to generate multi character animation sequences from sentences recognizable by defined motion grammars. Moreover, these context free grammars allow users to define action sequences for each character while adding spatio-temporal constraints using semantic rules. It is especially useful to control start and end points of characters motions as well as synchronization between several characters. They showed the efficiency of their system by designing a sketching interface which converts a 2D sketch of a basketball plan into a motion grammar sentence. Once the sequence as well as the semantics are correctly parsed, a multi-level Markov Chain Monte Carlo algorithm samples animation clips from a database which satisfy plausibility constraints such as avoiding interpenetration between character or insuring smoothness in the concatenation of successive animation clips. Figure 2.12 shows the process from sketching to animation generation with an intermediate sentence conversion step. Interestingly, they highlighted the fact that human motion cannot be generated/recognized using regular expressions. In our work we also share this assertion as we notice limitations using only regular expression for action recognition and use geometrical analysis for style recognition.

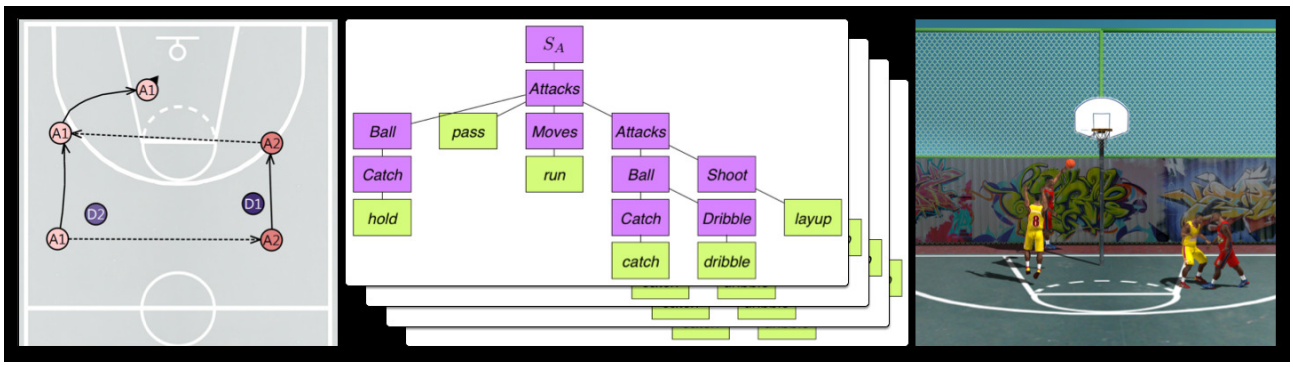


Figure 2.12: Motion Grammar workflow [Hyun et al., 2016]. Users sketch a 2D diagram of a basketball game. This sketch is then converted into an action sequence resulting from a successive application of grammar rules which are then used to sample animation clips from a database. They are finally assembled to create an animation sequence representing the basketball play drawn by the user.

2.5 Stylization

Creating expressive and stylized animations from neutral ones has always been a challenge as it involves very complex behaviors for any given intent. Moreover, despite having been studied for many years, the question of finding a generic model allowing to modify an animation into a given intent or emotion is still an open question. Like we mentioned in section 2.1.3, this problem has been studied through two different kind of approaches namely, procedural and data driven techniques.

2.5.1 Procedural Stylization

[Chi et al., 2000] introduced the EMOTE model, which procedurally modifies input animation with respect to Laban Effort and Shape features. Effort and Shape are applied using three operators, a joint re-positioning operator mostly used for the Shape features, a retiming operator which modifies the duration of anticipation, execution and follow-through phases and a flourishing operator applying additional operations such as wrist bends and arm twists. This system was further extended by [Durupinar et al., 2016] who brought two major contributions: the system provides a LMA feature to low-level animation parameters which was build with LMA experts as well as a Personality Factors to LMA features. The personality factors are modeled with the OCEAN model consisting in five orthogonal personality traits, openness, conscientiousness, extroversion, agreeableness and neuroticism. Combined together, the mappings allow users to stylize animations with respect to the personality of a character.

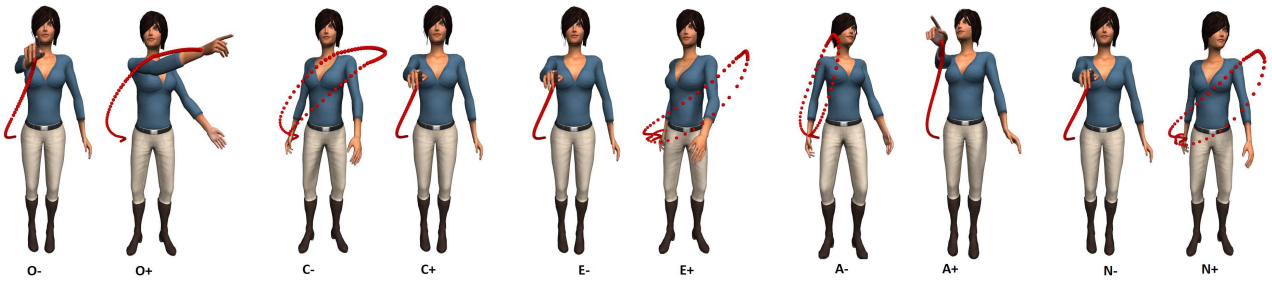


Figure 2.13: PERFORM [Durupinar et al., 2016] OCEAN based stylization. This personality based stylization relies on a two step mapping, first with Laban features which are then used to alter skeleton joint properties as established by LMA experts.

Likewise [Su et al., 2007] designed a system that generates stories and stylizes animations with respect to characters personalities. In contrast with [Durupinar et al., 2016] they model character personality using the Abridged Big Five Circumplex which extend the OCEAN model by considering pairing of personality traits as 2D continuous spaces. Interestingly, in their approach not only global posture and animation are stylized with respect to the character personality but also action performed by the character are influenced by his traits.

[Neff and Fiume, 2005] proposes a system which iteratively lets animators build animation sequences using animation script as well as sketches. Moreover, the animator can first define character properties like joint tensions and then establish an animation plan using a script. The system then interpret the input script into a resulting animation and lets the user modify it until he is satisfied. Stylization is provided by a set features like global posture, motion rhythm or joint tensions.

Stylizing motion using LMA has also be studied in the case of expressive robot motion generation. [Masuda et al., 2010] proposed a simple yet effective Laban stylization method, modifying robot joint position, velocity and acceleration depending on Laban Effort and Shape parameter. Moreover they correlated the Time axis with velocity, Weight with acceleration and Space with relative trajectories between face and body extremities. More recently, [Desai et al., 2019] created an interface to design expressive robot motions based on high level semantics like basic emotions [Ekman, 2003]. They tackle the problem using a data-driven approach to identify the role of each low-level parameter of the robot motion with respect to the six emotions. To ensure the scalability of the low-level to emotion mapping, a wide variety of motion were used for computing a regression which was possible thanks to crowdsourcing which was used to rate parametrized robot motion with respect to the basic emotions. Finally a linear regression was performed to relate low-level parameters to emotions.

2.5.2 Data-driven Stylization

One of the first notable data-driven stylization system was introduced by [Rose et al., 1998]. In their methods they classify each animation as a verb while stylized version are referred as adverbs which are spread among an adverb space. All animation clips and their respective stylizations are stored in a library. Adverbs can represents any kind of stylization space in which the users can navigate in a continuous manner. Moreover each joint DOF can be stylized in inbetween adverbs or even adverbs outside the convex-hull of the stored example using linear approximation and Radial Basis Function as regression model (see sectio 2.1.3).

In a similar way [Xia et al., 2015] designed an online regression algorithm to stylize unlabeled input animations. Moreover, given an input pose and style, the system performs a k-nearest neighbors (KNN) search inside a database of pre-computed animations (containing animations in different styles) finding the k poses closest to the input pose. The system then computes a Mixture of Auto Regressive models (MAR) for each degree of freedom accounting for previous poses and style changes. More precisely, a local regression model is computed for each combination of action and style in the database, the k nearest poses to the input pose are then used in the MAR which compute a weighted sum of the nearest pose associated regression models and interpolate/extrapolate the related poses resulting into the new stylized pose.

While this methods is able to handle heterogeneous motions, it shows limitations when stylizing animations which are not similar to the database. Addressing this limitation [Yumer and Mitra, 2016] tackle the animation stylization problem using a spectral domain approach. In this work, they noticed that for two distinct actions, the difference between the neutral and stylized magnitudes in the spectral domain are highly similar. Thus, given an input action in a given style and a target style, the system choose an already registered animation in their database both in the source and target style, for each joins computes their magnitude difference and subtract it to input action. This results in the magnitude of the desired style for the input action. Computing an inverse Fourier transform for each joins gives results in the desired stylized animations. It is worth mentioning that the idea of computing joint magnitude

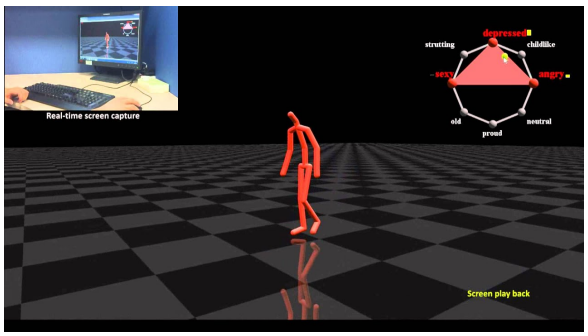


Figure 2.14: Xia et al. [Xia et al., 2015] style transfer example. Given an input animation, The user can interpolate across several styles at a time, the system find closest stylized animations in its database, computes a regression model on the flow and stylize the input animation.

differences between styles and apply it to another animation was first explored by [Amaya et al., 1996] who applied this same principle but in the time domain, which necessitates manual labeling of cycles or meaningful period for both animations in the database and for new animation inputs.

More recently [Smith et al., 2019] proposed a new data-driven stylization based on three neural networks, whose results are on par with [Xia et al., 2015] but is much less computational costly and fast enough to run on mobile devices, which was not the case of all previous methods (especially [Xia et al., 2015] who necessitate GPU parallelism).

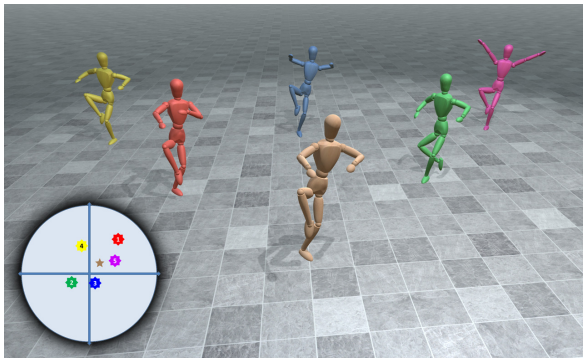


Figure 2.15: Aristidou et al. [Aristidou et al., 2017] Laban features dance stylization. The stylization takes RCM coordinates as input as shown in the bottom left disk. A mapping model between those coordinates and Laban features was computed before hand. The final stylization is based on a regression model.

Interestingly [Aristidou et al., 2017] proposed an hybrid approach for stylizing animations with respect to the Russell's Circumplex Model (RCM). Moreover, they defined a set of LMA features, classified into the Effort, Body, Shape and Space categories and which are used to modify input animations. Body features are represented by inter joint distances as well as hip-ground distance, Effort features are represented by extremities velocities, acceleration and jerks while Shape and Space are represented by the volume occupied by the body. A two-way mapping between RCM coordinates and LMA features is learned by recording professional dancer performances which are labeled relatively to emotions. The final mapping is computed using a Radial Basis Function regression. This data-driven pre-computation is then used to procedurally modify any input animation with respect to any RCM coordinate. It is achieved by applying heuristic rules which modifies the pelvis, hands, head positions as well as the animation framerate such that the new animation LMA features fit the values learned in the mapping. This approach also highlighted relevant LMA features that were highly varying across the different emotion without being dependent to the dancer.

In our work we tackle the stylization problem using a procedural approach, considering that building a animation database of various style is difficult to achieve for all the 30 animations used in our examples.

2.6 Multi-Character Animation

Multi-character animation adds another layer of complexity with respect to single character animation generation. Indeed, more constraints have to

be taken into account such as synchronicity, contacts between characters and global/relative positioning. Additionally, designing a system which gives intuitive and quick control over several characters is really challenging as we usually manipulate one or two objects at a time. Several works tackle this challenge essentially through sketching, scripting and behavior simulation.

[Perlin and Goldberg, 1996] designed a complex real time authoring system decoupling characters motions and behaviors. Moreover, a Behavior engine takes decisions about which action should be executed by the character at each frame. Behaviors are inferred through scripts in which users can specify a wide variety of decision rules and parametrized action sequences such as triggers to position or orientation, stochastic decisions, layered actions and low level body part displacements. Multi-character authoring is dealt by treating all characters as being a single one and by adding rules for one characters to change others cues.

While this approach covers a wide range of very complex scenarios, behaviors lacks re-usability for long sequences as users have to write all the steps before reaching the desired goal. In addition, creating scenario variations can be tedious as this it is not part of the system design. [Kapadia et al., 2011] address those two issues by putting an additional layer of abstraction between the user and the resulting animation. Moreover, using states, actions, costs and constraints as input, they designed a heuristic algorithm searching for a animation sequence plan. Additionally they let the user specify personal goals for each character as well as a final global goal that must be reach to end the animation sequence. Actions and states are correlated in the sense that some actions can be executed in certain states and some actions change states of characters. Variants can easily be generated by using a set of modifiers which changes states or cost properties. In summary, they propose an animation sequence planner based on cost minimization with respect to personal and global goal reachability which as the advantage to be more flexible than [Perlin and Goldberg, 1996] to the cost of losing some precise control of the generated animation sequence. This method was further exploited by [Shoulson et al., 2013] modifying the [Kapadia et al., 2011] planner into an event-centered one. The new algorithm search in the event space rather than the personal character action set which highly diversify the generated animation sequences as interactions can replace characters individual actions while still achieving their personal goals as well as global goals if the user specifies any.

[Shoulson et al., 2014] generalize the two previous methods in the more complex system which let users customize either the behavior algorithm used

for planning, the trajectory planer system (navigation mesh for instance) and the animation coordinator which takes the decision of which and how to play animation clips with respect to actions and trajectories given by the two other modules. This system is in the line of the end-to-end solutions like [Perlin and Goldberg, 1996] while being highly flexible.

The methods we just presented are focused on generating consistent multi-character animation sequences by taking scripts, goals, states, action and constraints as inputs. However, with such systems it can be difficult to relate those inputs with a movie script which purely contains dialogues and action and stage directions. In addition, they do not provide users with an interactive and intuitive interface that let them see the results of changes in the scenario. [Kapadia et al., 2016] and [Marti et al., 2018] propose two visual authoring tools that let user manage their scenarios either by providing partial story board or by interpreting their script using Natural Language Processing. Users are interactively provided with several kind of visual feedback being characters action time-lines, 2D top-view objects and characters displacement or 3D views of the underlying scene which are automatically generated by these systems. Those systems rely on the notion of smart objects and affordances which define objects/characters that can perform actions or which notify other actors on how they can be used.

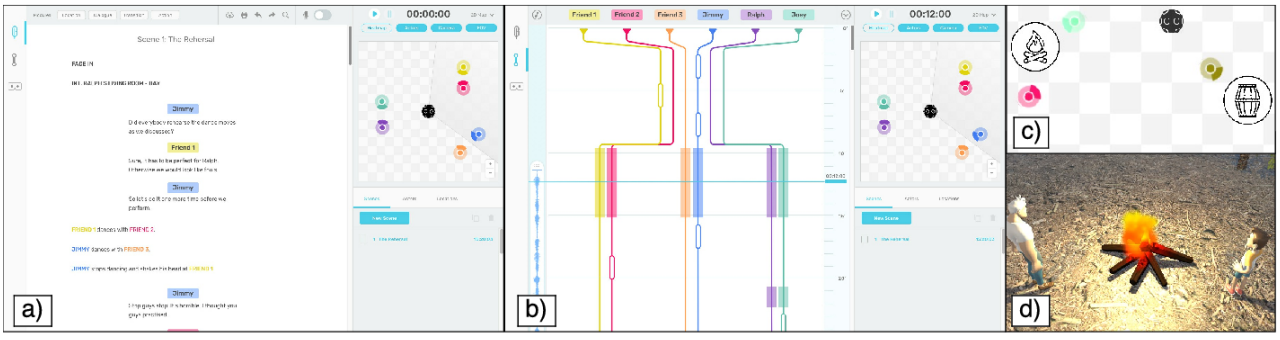


Figure 2.16: Cardinal framework [Marti et al., 2018] interpreting user provided script and generating a 2D and 3D visualization of the scene.

Finally works like [Kim et al., 2009] focus on multi-character animation edition rather than generation which rely on character trajectories and timing constraints handling rather than editing high level representation of the sequence. Moreover, users can sketch new trajectories, specify relative character locations, action duration and synchronization. All timing and spatial modification are handled in an as rigid-as-possible way, smoothing the whole edition process and avoiding abrupt discontinuities.

In our work we propose two approaches to multi-character authoring. The

first one is a direct extension of the spatial motion doodle which accounts for character interactions. We also propose a scripting approach which generates perfectly idealized doodles using Action/Interaction regular expressions as well as space-time optimization showing that our system can also be generative with respect to desired animation sequences.

2.7 Conclusion

Animation authoring tools have been studied through a wide variety of methods; from using the human body, physical props and sketches to pose or animate characters, to procedural and scripting approaches to either stylize animations or even control several characters at a time. However, we showed that there is still room for improvement especially for authoring animation sequences using physical props and for stylizing animations without relying on an animation database. In the next chapters, we address these topics by proposing new techniques to recover actions and motion qualities from hand motions (chapter 4) and to create an adequate expressive animation sequence which translates users intents (chapter 5) and allow them to create animated stories (chapter 6).

Chapter 3

Performance Acquisition Setups

In this work, we used three different setups to acquire user narrative performances namely the FIGURINES system, The Slate 2 by ISKN [1] and the HTC Vive controllers and trackers. Those acquisition systems all return position and orientation data about the manipulated objects. However, they have different features as well as different pros and cons. In this chapter we describe those setups and discuss their properties and uses.

3.1 FIGURINES Setup

The first recording system we used for capturing storytelling performances of one narrator was the FIGURINES system we designed. This system aims at letting one user play with rigid figurines that are tracked using one top view Kinect sensor and Interial Motion Units (IMUs), returning acceleration data. IMUs are placed inside the support of each figurine. The combination of the two sensors avoid losing the figurine tracking when the narrator's hands occlude them from the Kinect video while guaranteeing an acceptable precision. Figurines are tracked up to 200 FPS using the IMU sensors allowing us to recover their entire 6D trajectories (position and orientation).

Additionally, one front view Kinect sensor records the narrator facial motion, capturing his head position and orientation, his gaze direction as well as facial expression features (48 face feature positions, see figure 3.2). Using the commercial software Faceshift, we are able to recover an animated facial mesh of the narrator oral performance. The narrator is also equipped with a microphone used for recording his voice during the play. The entire system is presented in figure 3.1.

The performance data processing is done in an offline fashion in order to improve precision in the 6D path reconstruction. It also allows us to correctly

[1] ISKN: `https://www.iskn.co/`. Note that the Slate has been replaced by the Repaper.

synchronize the narrator's voice and facial motion with the figurines' motion.

Like explained in chapter 1 this acquisition setup allows us to transfer each figurine rigid motion to a corresponding rigged 3D model and to identify which character the narrator is impersonating at a given frame. Using this information and the facial motion data, we are also able to animate the 3D characters faces by transferring the narrators facial expressions to the embodied character at each frame of the output animation.

Figure 3.1: FIGURINES setup: figurines are tracked using a IMUs sensors paired with a top Kinect sensor. The frontal Kinect records the narrator's facial expressions. The play stage is the table in the middle and covers a surface of $70 \times 70cm^2$.

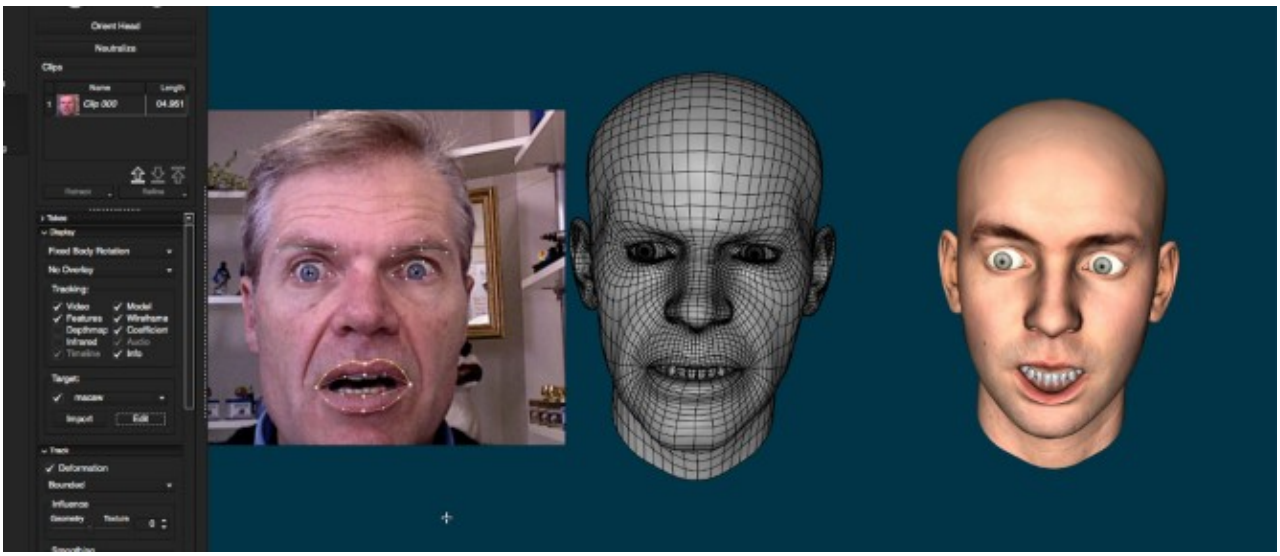


Figure 3.2: Faceshift Software tracking narrator facial expression and transferring them to a 3D character facial mesh.

While recovering both the figurines' and the narrators' motion data as well as his voice, this system still presents some accuracy issues mainly due to physical and magnetic perturbation of the IMUs sensors (contact with the ground and other figurines). In addition, 6D path recovering from acceleration and video data can take some time to reconstruct. Finally, during this thesis, we were forced to drop the narrator tracking as the Faceshift software is not available to the public anymore. Alternative solutions can easily be used to account for this missing feature like the FaceTracker existing in the Kinect SDK, but this has been left to future work.

3.2 The Slate by ISKN

During this thesis, we collaborated with a French company ISKN located in Grenoble which develops a pen tracking device digitizing sketches drawn on physical paper laying on a tablet containing sensors, namely, the Slate. More precisely, the tablet tracks a ring shaped magnet which can be attached to any pen and retrieves both its 3D position in space and 2 orientation coordinates excluding the twist (along the pen axis). In our case, we attached the ring to a LEGO brick which plays the role of the manipulated object for narrative performance. The figurine is tracked at 144 fps and can be manipulated over a $15 \times 21 cm^2$ surface with a maximum height of approximately $12cm$. A SDK is available on their web site which allows us to integrate this recording device both in online or offline applications. In addition, this device has the advantage of being cheap (around 175$) and easy to carry. Figure 3.3 shows the Slate device and how we use it in our context.

(a) The Slate 2 composed of 5D sensors inside the tablet tracking the ring attached to the pen.

(b) We adapted the Slate 2 device by attaching the tracked ring to a LEGO brick.

Figure 3.3: ISKN The Slate 2 device. Originally used for tracking pen motion for digitizing sketches while drawing them, we use this device on a LEGO brick as it gives us back the position and two orientation component (excluding the twist)in 3D relatively to the Slate censors. This device tracks the magnetic field produced by the ring magnet and give real time feedback (144 fps)

However, this setup comes with important drawbacks. First, the twist orientation cannot be retrieved by any means and is set at zero by default. Secondly, this system cannot track several rings at a time which limits the use of this system for multi-character scenarios. Thirdly, this system is limited in terms of playing area and can be perturbed by many electronic devices.

As it is, we mainly use this device for development and quick prototyping in the one character scenario case as it is easy to carry and to interface with different applications. In the future, it would be interesting to investigate further how to use this system for online applications like video games.

3.3 HTC Vive

As VR and AR are evolving at a high pace and are more and more used as an application interface, we also experimented performance capture using the HTC Vive setup. It tracks a head mounted display (HMD) and two controllers both in position and orientation. In our experiments, we use the controllers as figurines the narrator can play with. Both the HMD and the controllers are tracked at 90 fps at high precision which is very suitable for creating stories with two characters. The controllers are light and are $20cm$ tall which fits well in the hand. The sensors covers a $5 \times 5m^2$ area, which is more than sufficient for our application. In addition, they are highly resistant to occlusion. Several library like OpenVR and SteamVR allow us to interface the HTC Vive with different applications. Moreover, we use the OpenVR plug-in for Unity 3D to record the narrator's performance inside a virtual playground. Figure 3.4 shows an example of the HTC Vive setup.

Figure 3.4: HTC Vive setup composed of two controllers and a HMD providing a stereoscopic display. The tracking is done at 90 fps at high precision.

During our experiments we observed that using the controllers as recording device sometimes caused discomfort when making high turns (see chapter 5 and 6). To cope with this issue and to increase the number of character we can play with at a time we attached HTC Vive trackers to 3D printed figurines of Commedia Dell'arte characters like shown on figure 3.5.

.

Figure 3.5: Commedia Dell'arte 3D printed figurines attached to HTC Vive trackers. For our experiment we printed a wood mannequin version of Arlequin, Pentalone, Pierrot and Colombine.

So far, this setup is the most appropriated system we tested for our performance acquisition process. All the animations and trajectories shown in the experimental results of the next chapters were acquired with this setup mainly using the controllers.

Chapter 4

Performance Analysis

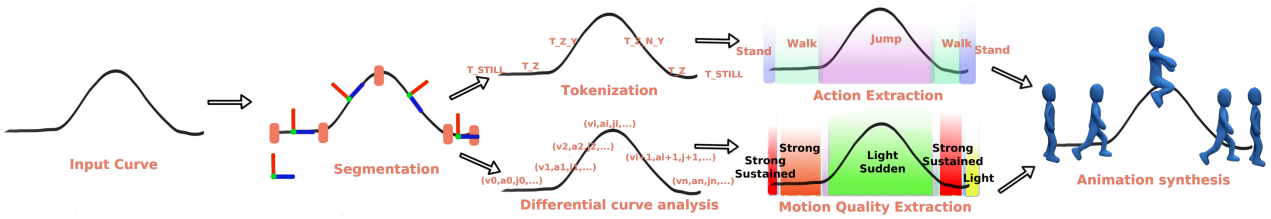


Figure 4.1: Spatial motion doodle pipeline: We first segment the input curve (spatial component of the SMD) based on changes on motion direction using a moving local frame, and label the segments with motion tokens. We then recognize actions as regular expressions over motion tokens, together with their motion qualities. Finally, we synthesize corresponding animations following the input SMD trajectory, timing and motion qualities.

In our quest of creating expressive animations from figurine narrative performances, we resort to trajectory analysis for extracting an action sequence as well as motion qualities from the figurines' rigid motion. As mentioned in chapter 1, we extract hand motion qualities in terms of Laban Efforts while we propose our own algorithm for action recognition.

We first introduce our trajectory model and define the different types of action we can recognize (section 4.1). Then, we describe our action recognition algorithm which relies on a new position, orientation and scale invariant descriptor we devised. We also present an action learning algorithm which enable our system to let users perform their own gestures during recordings (section 4.2). In a second part, we explain how we retrieve Laban Effort qualities from users' hand motions. Moreover, we use two different machine learning techniques to perform Laban Effort classifications: a Naive Bayes classification and a HMM-based classification (section 4.3). We show the efficiency of our action recognition algorithm and our Laban Effort classifiers through a series of experiments. We also discuss the limitation of our hand motion analysis system. The whole performance analysis pipeline is depicted in figure 4.1.

4.1 Trajectory and Action Models

4.1.1 Spatial Motion Doodle

In this work, we aim at recovering storytellers intent using mainly their hand motion as input. In chapter 3 we presented the different setups we use for acquiring figurines trajectories in terms of position and orientation.

We define those 6D curves as Spatial Motion Doodles or SMD which are sets of points $\{p_i\}$, where $p_i = (x_i, y_i, z_i, \theta_i, \phi_i, \gamma_i)$ containing the position and orientation over time of the center of mass of the tangible object in a global world frame and where y denotes the vertical direction.

One main challenge we tackle is to recognize both actions and motion qualities using only this information as input. This is done under the constraint that we allow users to execute any action from any starting point, orientation and at any amplitude. In other words, we allow then to stage their scenes as they wish. This constraint leads us to characterize motion in terms of local displacement with respect to a current position. In figure 4.2, we show how the tangible object and the character frame relate to each other where x, y and z axis respectively represent the LEFT, UP and FRONT directions.

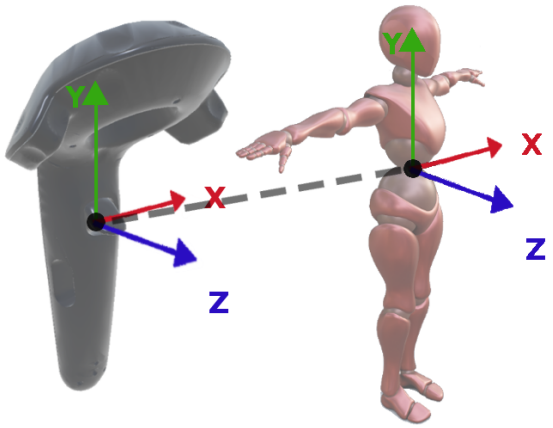


Figure 4.2: Controller and Character Moving frames. Z axis points towards the front direction, Y points axis up and X points left.

Consequently, for each input SMD, we compute its related local displacement curve $\{(\vec{v_i}, \vec{\omega_i})\}$ using the character's moving frame at each point p_i:

$$\vec{v_i} = R_i(p_i - p_{i-1}) \tag{4.1}$$

$$\vec{\omega_i} = (\theta_i - \theta_{i-1}, \phi_i - \phi_{i-1}, \gamma_i - \gamma_{i-1}) \tag{4.2}$$

$$R_i = R_z(\gamma_i)R_y(\phi_i)R_x(\theta_i) \tag{4.3}$$

$$R_x(\theta_i) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_i & -\sin\theta_i \\ 0 & \sin\theta_i & \cos\theta_i \end{bmatrix} R_y(\phi_i) = \begin{bmatrix} \cos\phi_i & 0 & \sin\phi_i \\ 0 & 1 & 0 \\ -\sin\phi_i & 0 & \cos\phi_i \end{bmatrix} \tag{4.4}$$

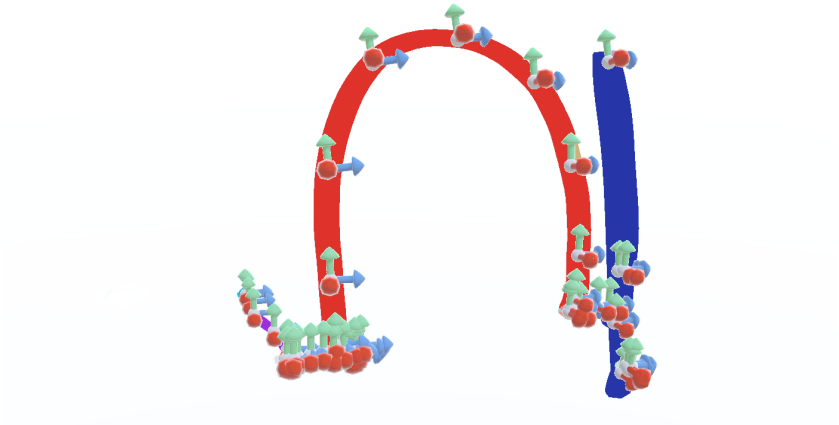
$$R_z(\gamma_i) = \begin{bmatrix} \cos\gamma_i & -\sin\gamma_i & 0 \\ \sin\gamma_i & \cos\gamma_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

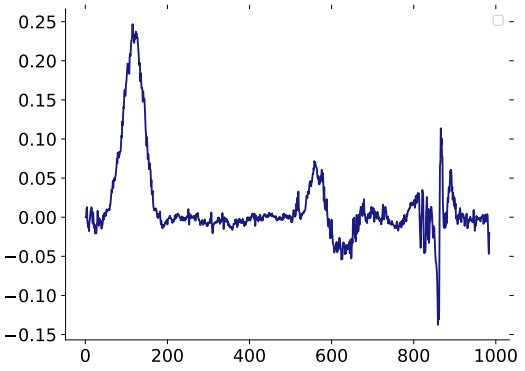
$\vec{v_i}$ represents the local linear velocity of p_i while $\vec{\omega_i}$ represents its angular velocity and R_i the rotation matrix of p_i. Thanks to this representation we are able to describe each figurine motion in terms of local direction description; we can easily deduce if one character moves forwards, backward, to one side or if it turns at any moment. In the next sections, we explain how we use the $\{p_i\}$ and $\{(\vec{v_i}, \vec{\omega_i})\}$ curves as well as their derivatives to perform both action and motion quality recognition. In figure 4.3, we show the SMD of our main example and its corresponding local velocities curves. In this figure we can easily relate the local y and z linear velocity variations between the frames 500 and 700 to the jump (parabolic shaped) of the input SMD; with $\vec{v_y}$ increasing at the beginning of the jump and becoming negative when falling down while having a positive $\vec{v_z}$ translating the fact that the character moves forward during that period.

Note that before we carry out any computation on trajectories, we first de-noise and smooth the input SMD. During this thesis, we tested two de-noising algorithms: a third order Savitzky-Golay method [Savitzky and Golay, 1964]) and a Laplacian smoothing. In practice we noticed that the Savitzky-Golay algorithm sometimes introduces undesirable behaviors near the extremities, instead we chose the Laplacian smoothing which proved sufficient for our needs.

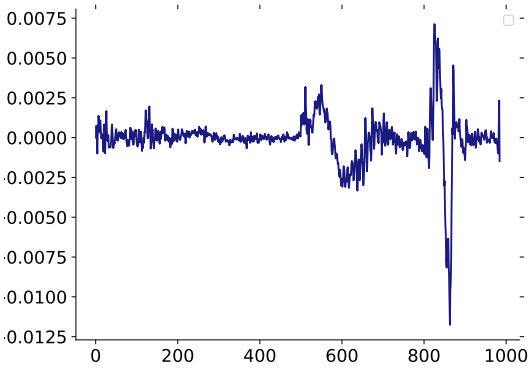
4.1.2 Action Types

When animating characters experts usually differentiate cyclic actions from the rest of as they only need to animate one cycle and then repeat it as much as they want in their final results. This is especially the case of Walk and Run animations which contains few keyframes that are repeated over and over along a trajectory. In our work, we also categorize actions between cyclic and non-cyclic and also distinguish them between in-place and not in-place (follow a trajectory) actions. This aspect plays an important role when transferring the doodle into the actual trajectory the character will follow (see chapter 5). For

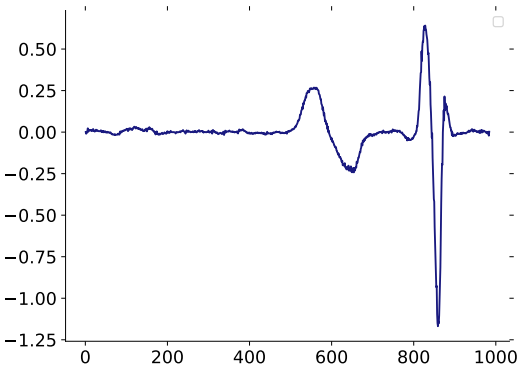




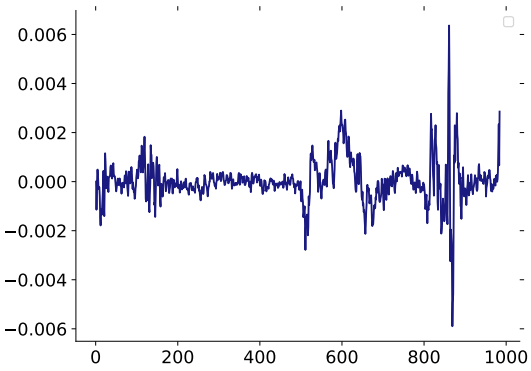
(a) Local linear speed x coordinate curve $\vec{v_x}$.



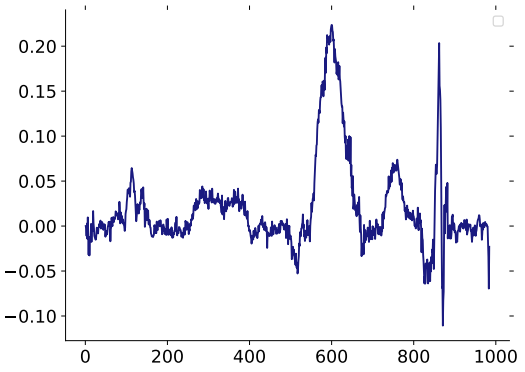
(b) Local angular speed x coordinate curve $\vec{\omega_x}$.



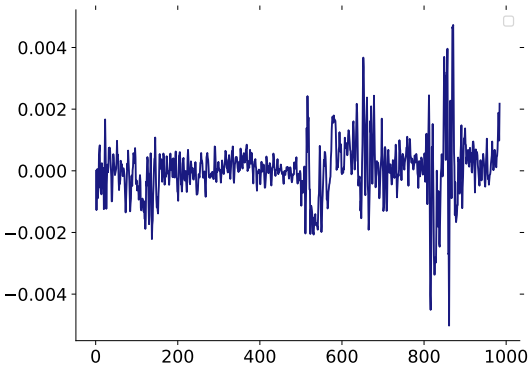
(c) Local linear speed y coordinate curve $\vec{v_y}$.



(d) Local angular speed y coordinate curve $\vec{\omega_y}$.



(e) Local linear speed z coordinate curve $\vec{v_z}$.



(f) Local angular speed z coordinate curve $\vec{\omega_z}$.

Figure 4.3: Spatial Motion Doodle of our main animation transfer example. Note that several frames are displayed along the trajectory showing the orientation of the figurine at those specific points. Its related local velocities shows that the motion sequentially followed the local x axis then the local z and y axes for the rest of the trajectory.

the same reasons, in our context we also dissociate actions that make characters jumping out of their starting position from the rest of the actions.

By combining these three aspects we can classify actions into 8 categories being cyclic or non-cyclic and in-place or not and making a jump or not. Table 4.1 shows this classification for a subset of our action library.

Action	Cyclic	In-place	Jumping
Walk	✓	✗	✗
Jump	✗	✗	✓
Punch	✗	✓	✗
Idle	✓	✗	✗
Stomp	✗	✓	✗

Table 4.1: Classification of a subset of action present in our library in terms of cyclic, in-place and jumping aspects.

In this chapter only the cyclic aspect of actions intervenes in the action recognition process while the other two will play a role in chapter 5 when creating the animation sequence out of the input SMD.

4.2 Action Recognition

Action recognition is a well studied topic both from videos and from human motion data [LaViola and Keefe, 2011]. Traditionally action recognition from 3D trajectories is done in global space and relies on properties such as speed and acceleration. Similarly to [Bribiesca, 2007], we propose a curve partitioning representation which has the advantage of containing an additional orientation data from which we compute motion features with respect to a moving frame like explained in section 4.1. In this section we describe how we perform action recognition from an input SMD. Moreover, we further detail how our system segments the input SMD into a sub representation in terms of translation and orientation direction changes and how actions are represented for performing both partial and full matching.

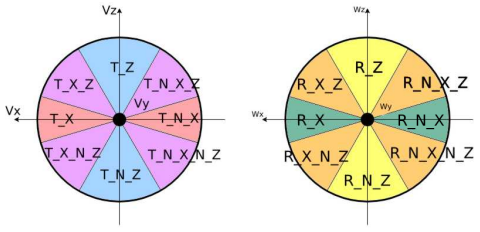
4.2.1 Recognition Process Overview

Our action recognition system is similar to a compiler and is directly inspired by [Thorne et al., 2004] who encode actions as regular expressions of a language whose tokens represent 2D strokes orientations (see chapter 2). In this work we extend this idea to 6D, classifying curve segments into speed and rotational

speed direction changes. We call this first step tokenization, which consists in converting the input trajectory into a sequence of motion tokens. The tokenization process is performed by a state machine, the Segmenter, which plays the roles of a lexer (which traditionally recognizes floats, integers, strings an so on when parsing code). Then, another state machine, the Matcher, reads the token sequence in the chronological order and checks for partial and full regular expression matches with the animation expressions database. In an analogous manner, the Matcher performs a syntax analysis of the token sequence (checking that token sequences follows construction rules when coding). The final output of this process is a sequence of action labeled curve segments representing the action sequence which will be played by the related character.

4.2.2 Tokenization

To be able to consistently recognize actions (eg. Jump versus Kick), we rely on the $\vec{v_i}$ and $\vec{\omega_i}$ variations at each point p_i of the input curve which give information on local position and orientation changes.



(a) Plane of XZ linear velocities. (b) Plane of XZ angular velocities.

Figure 4.4: We partition the space of linear and angular velocities into 27 spherical bins of width $\frac{\pi}{6}$ (Only the X and Z components are shown in the figure).

More precisely, the Segmenter partitions the $\{(\vec{v_i}, \vec{\omega_i})\}$ curve, computed from the input SMD, into *spatial motion tokens*. The latter represent short snippets of 6D motion where local velocities remain in the same bin in terms of direction. Moreover, token values represent bins partitioning the 6D space of local linear and angular velocity directions (see Figure 4.4 for a 2D illustration and figure 4.5 for a 3D representation). The $T\_translation\_or\_rotation$ token indicates that the motion is in a given bin of velocity direction, with respect to the current orientation of the character.

For instance, the T_Z token corresponds to a forward character motion (positive $\vec{v_z}$ direction) while T_N_Z token corresponds to a backward motion. In our implementation, we use 27 bins in translation and 27 bins in rotation.

Our segmentation algorithm works as follows: for each new point p_i, the Segmenter computes its related motion token T_i and store it in a stack. If the first token of the stack is different from T_i, the stack is emptied and is filled with T_i. Then, when the stack size is greater than α (sensitivity), it is emptied and the token T_i is declared as recognized and assigned to the curve segment it represents. α depends on the frequency of the device used for recording the

SMD, the higher the frequency the greater α is. We tested different α values for the HTC Vive controllers and empirically found 5 to be the best α value. For other devices we simply take the HTC Vive α value as reference and propose the following formula $\alpha_d = 5\frac{F_d}{F_{HTC}}$. This process further de-noises the input curve and is especially useful for removing small unintended gestures which often occur when holding a 3D device.

To extract T_i, we compute the signs of the six local linear and angular velocities $\{(\vec{v_{xi}}, \vec{v_{yi}}, \vec{v_{zi}}, \vec{\omega_{xi}}, \vec{\omega_{yi}}, \vec{\omega_{zi}})\}$ with respect to each axis, identifying the right bin. Evaluating the sign of these velocities enables us to partition the SMD into segments with constant local velocity directions.

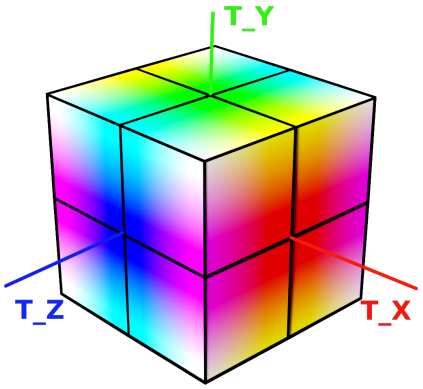


Figure 4.5: Linear velocity space segmentation. Each edge intersection represents a segmentation bin among the 27 bins. The angular velocity space is segmented the same way (replacing $T\_$ by $R\_$).

Indeed, for a velocity vector $\vec{v_i}$ and an axis $\vec{x}$, the Segmenter determines that $\vec{v_i}$ contributes to $\vec{x}$ if $dot(\vec{v_i}, \vec{x}) \geq 0.5$ $(cos(\frac{\pi}{3}))$. Additionally, we restrict the space of possible bins to 54 which do not consider simultaneous translations and rotations. This decision is justified by the fact that when rotating the recording device, the moving frame rotates the same way as the prop and consequently influence any linear velocity direction occurring. Thus, for each pair $(\vec{v_i}, \vec{\omega_i})$ the Segmenter takes the decision if the current motion is a translation or a rotation by evaluating $||\vec{v_i}|| \geq \theta||\vec{\omega_i}||$. In practice we take $\theta = 300$. θ can be evaluated by asking user to perform a minimal translation only gesture and a minimal rotation only gesture, assigning θ to $\frac{\min_i ||(\vec{v_i}||}{\min_j ||(\vec{\omega_j}||}$

Finally, we identify segments along the SMD where $||(\vec{v_i}, \vec{\omega_i})||^2 \leq \epsilon$ and assign them to the T_STILL token symbolizing immobility. After many recordings we noticed that the ϵ value depends on the user but it usually stays in the $[0.02 : 0.05]$ interval. This value can also be computed by asking users to move at their minimal estimated speed and assign ϵ to $\min_i ||(\vec{v_i}, \vec{\omega_i})||^2$.

4.2.3 Action Matching

Once the SMD is converted into a sequence of tokens, the Matcher preforms an approximate matching of the regular expressions associated with actions in our database. Each action is represented as a regular expression of spatial mo-

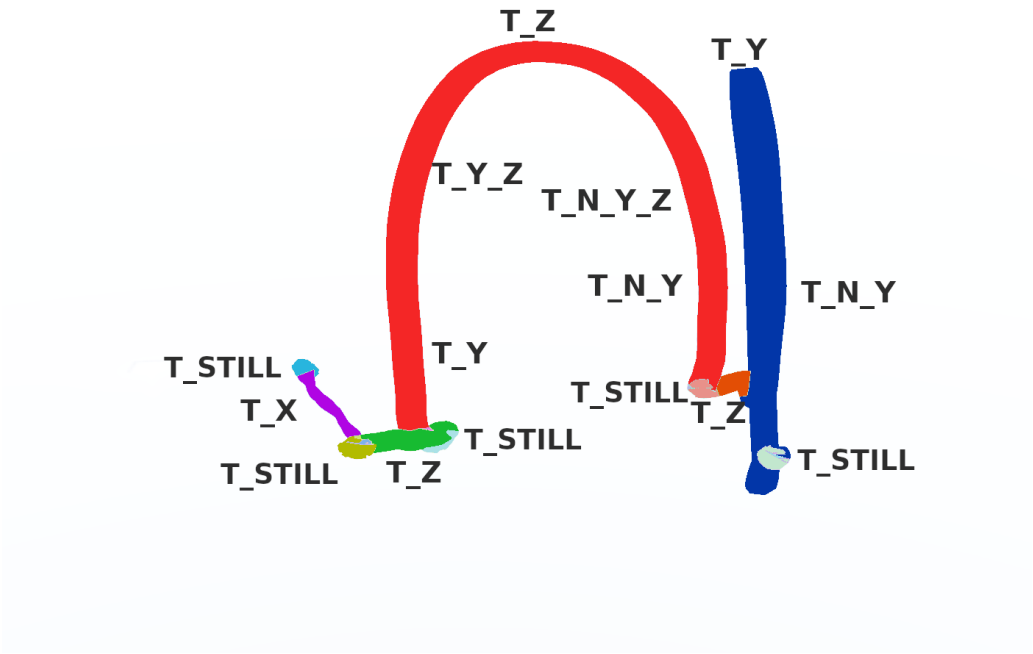


Figure 4.6: Tokenization process of our main animation transfer example (left to right).

tion tokens containing only $*$, $+$ and $|$ operators, where $*$ means zero or more occurrences, $+$ means one or more occurrences and $|$ is the logical *Or* operator. Figure 4.13 shows a subset of actions of our database together with their related regular expressions. The Matcher parses the token sequence in chronological order, store them in a stack, and check for approximate matches with respect to the regular expressions of our database. Once an expression matches, the Matchers save the related action as the longest matching expression but still continues to parse the token sequence without emptying the token stack. Indeed, as regular expressions contains $*$ and $+$ repetition operators, matching the current longest sequence can still go on after the first match. Secondly, as regular expressions of our database can overlaps between themselves, the Matcher keep the longest regular expression matched as the detected action. In practice, this decision proved efficient as confusion between overlapping actions occurs very few times in our user studies (see chapter 5).

We use approximate matching, rather than exact matching, to allow small errors in the spatial motion tokens drawn by the user. This is done by searching for the action with the smallest Levenshtein distance to the current token string, using a variant of Levenshtein automata [Schulz and Mihov, 2002]. Noting that regular expressions in our database are insensitive to repetitions of atomic tokens, we remove all $+$ from action regular expressions. Then, we enumerate all possible sub-expressions of non-repeating tokens by traversing all $*$ and $|$ symbols in the regular expressions. This allows us to customize very compact

and efficient Levenshtein automata suitable to actions in our database. For instance the Jump expression of figure 4.13 can be transformed into the following Levenshtein expression:

$$(T\_YT \| T\_Y\_Z) + T\_Z + (T\_N\_Y \| T\_N\_Y\_Z) +$$
$$\Downarrow$$
$$(T\_YT\_ZT\_N\_Y \| T\_YT\_ZT\_N\_Y\_Z \| T\_Y\_ZT\_ZT\_N\_Y \| T\_Y\_ZT\_ZT\_N\_Y\_Z)$$

In practice, the Matcher also search for exact matching if any occurs, they take priority over approximate matching. The full Matcher algorithm is written in Algo 1.

Our action detection method brings several benefits. It is translation and rotation invariant as all computations are performed in a moving frame (in contrast with [Thorne et al., 2004] where the drawing plane frame is used). In addition, the method is also scale invariant in space as each token of regular expression is followed by a + or a $*$ making them infinitely repeatable. These properties give more freedom to users: whatever the current position and rotation of the prop, they will be able to execute any recognizable action; furthermore each action can be executed at their preferred size, ie. either within a small or a wide area. It is also worth noticing that our action formalism can easily represent an infinite number of actions regular expressions without bringing confusion in the detection process: for instance, considering a set of n actions, we can assign $\forall i, A_i = (T\_YT\_N\_Y)^i$ (up and down motion) repeated i times for each action. As the Matcher recognizes the longest matching regular expression with respect to an input token sequence each A_i is detected by our system when users execute the $T\_YT\_N\_Y$ sequence i times.

However our method also comes with some limitations. First, even if we deal with overlapping actions, confusion can still occur because of the proximity of several regular expressions, this is something to be aware of when assigning regular expression or learning them (see section 4.2.5). Secondly, for some actions users might want use the same gesture but with different traits like speed (Walk and Run for instance) which is not something currently analyzed by our system.

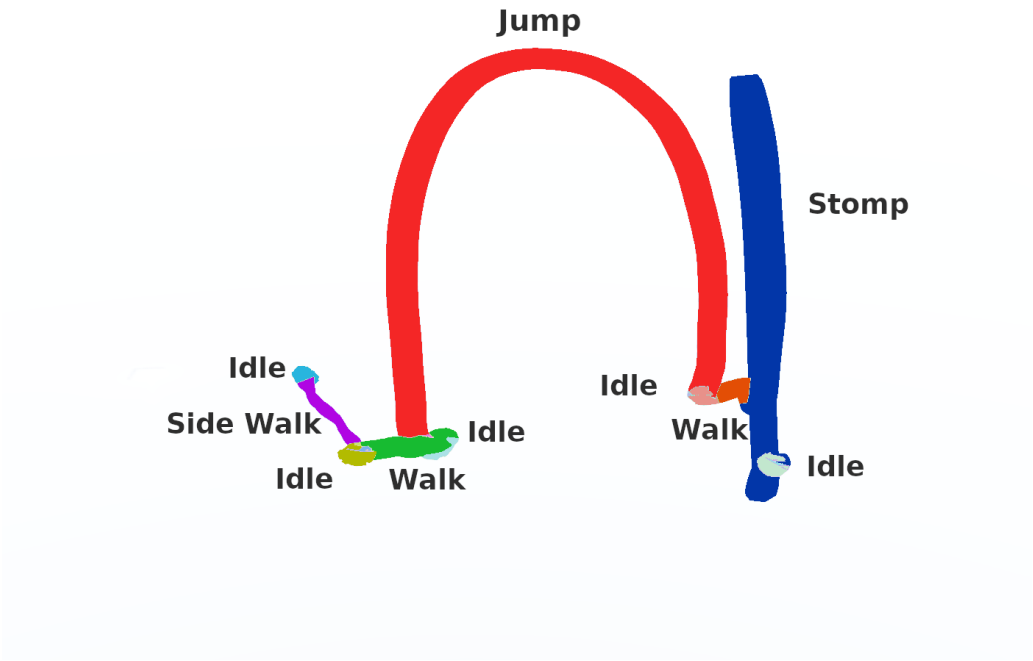


Figure 4.7: Action matching process of our main animation transfer example (left to right). Reading the output tokens successively and using the regular expressions of table 7.1, the Matcher detected the following action sequence: Idle, Side Walk, Idle, Walk, Idle, Jump, Idle, Walk, Idle, Stomp.

Data: Input 6D Curve
Result: Sequence of actions with their corresponding curve segment
set current token sequence as empty;
$leven\_threshold = 2$;
while *points to read* **do**
  read current point;
  compute direction token and add it to current sequence;
  compute matches(current sequence);
  if *no matches* **then**
    if *best match* $\neq$ *unknown action* **then**
      push best match
    else
      for *action in Actions* **do**
        Compute Levenshtein Distance(current sequence,action
        Leveinstein expr)
      end
      Store the closest action from current token sequence as $Clos_{Act}$;
      if $LevenshteinDistance(Clos_{Act}) \leq leven\_threshold$ **then**
        push best match ($Clos_{Act}$);
      else
        push best match (unknown action);
      end
    end
    reset current token sequence;
  else
    if *matches* **then**
      if *last action is cyclic && last action is matched* **then**
        add read input to action curve segment
      else
        store matches as biggest match;
      end
    end
    if *partial matches && no matches* **then**
      if *last action is cyclic && last action is partially matched* **then**
        add read input to action curve segment
      end
    end
  end
end

Algorithm 1: Action detection algorithm

4.2.4 Checking Regular Expression Overlapping

As mentioned in section 4.2.3, actions regular expression can overlap. While proposing to users non intersecting regular expressions that are as close as possible to the initial ones is a very difficult problem as users might want to preserve their initial gesture, notifying them of such intersections can be done with existing methods. In this section, we describe a method to check for regular expressions intersection, which is a well known algorithm in formal language theory. For two regular expression R_1 and R_2 , the principle of the method works as follows:

- Compute the two related non deterministic automata (NFA) N_1 and N_2 using Thompson [Thompson, 1968] algorithm
- Compute their deterministic equivalent (DFA) D_1 and D_2 and their complementary D_1^c and D_2^c
- Revert $(D_1^c \cap D_2^c)^c$ into a regular expression to obtain the intersection of the two regular expressions

Let us give more explanations on a concrete example, $R_1 = T\_Z*T\_Y+T\_X*$ and $R_2 = T\_Z*T\_Y+$ restricting ourselves to the language $\{T\_ZT\_YT\_X\}$.

We first build their corresponding NFAs using Thompson algorithm producing the automata of figure 4.8.

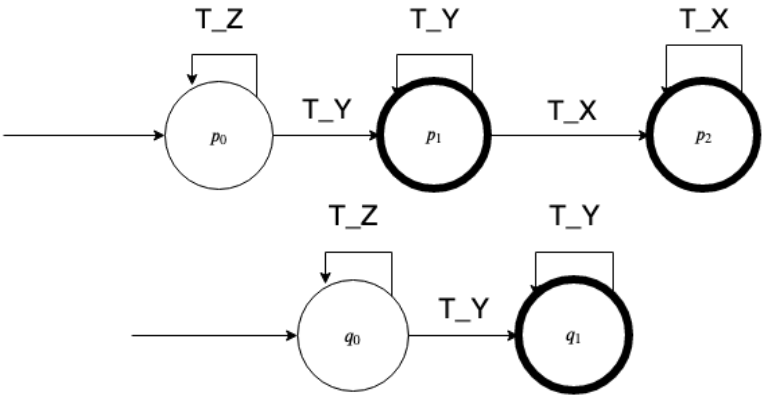


Figure 4.8: Non deterministic automata N_1 and N_2 of R_1 and R_2 regular expressions.

Those automata are still not completely deterministic as not all states possesses transitions for all tokens. Thus, we need to add the remaining transitions, connecting them to additional non reachable states. The resulting deterministic automata are depicted in figure 4.9.

Computing the complementary of DFAs is an easy operation as it consists in inverting all finishing on non finishing states. They are depicted in figure 4.10.

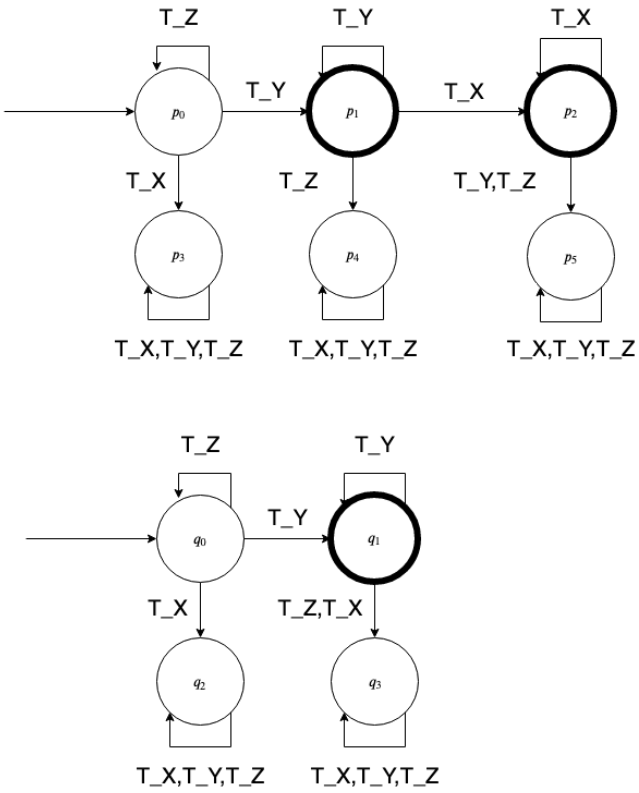


Figure 4.9: Deterministic automata D_1 and D_2 of R_1 and R_2 regular expressions.

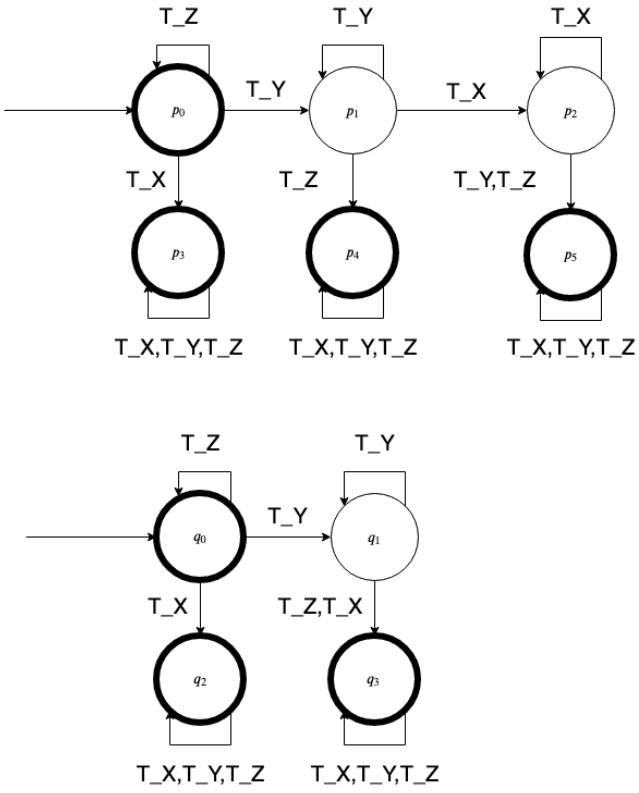


Figure 4.10: Complementary of the Deterministic automata D_1^c and D_2^c of R_1 and R_2 regular expressions.

The union of DFA is obtained by computing the product automata of R_1^c and R_2^c. This union is represented in figure 4.11.

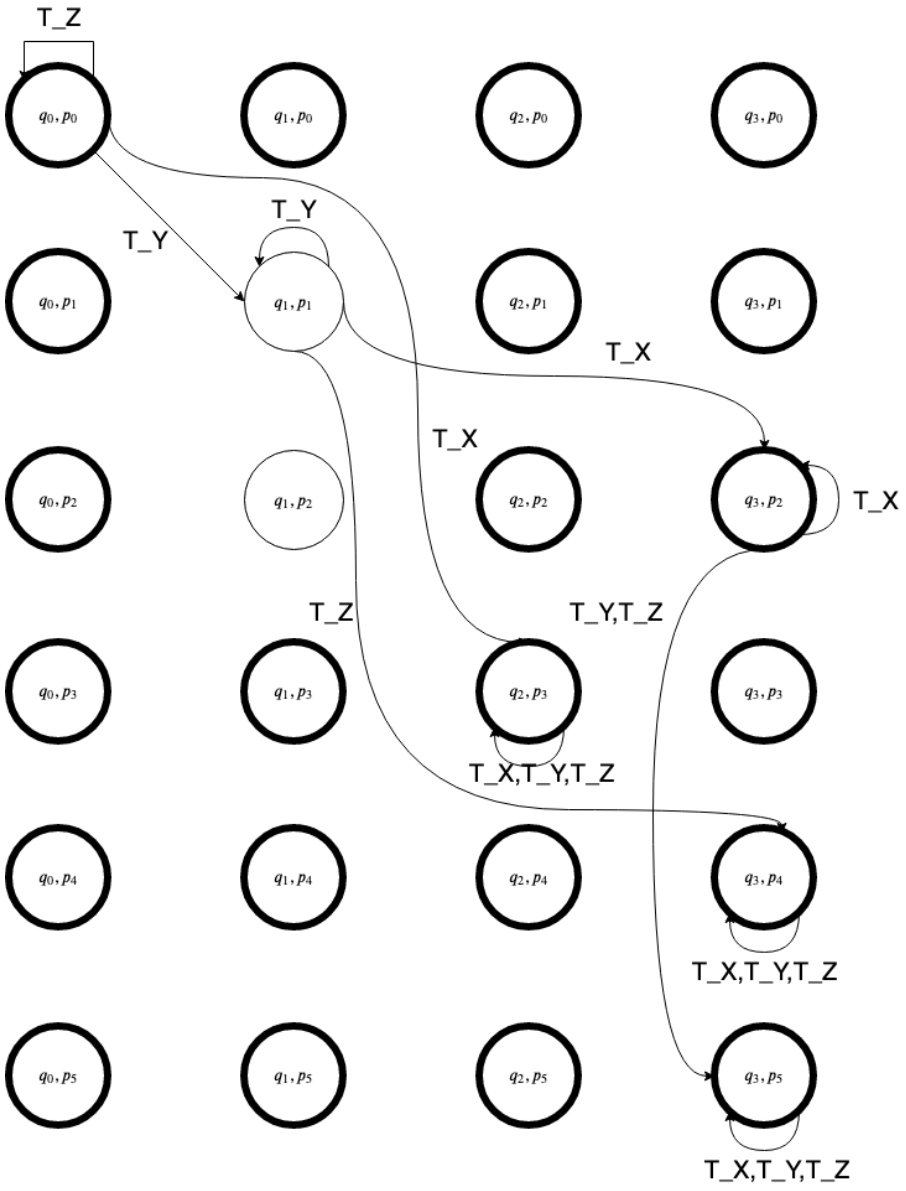


Figure 4.11: Union of complementary deterministic automata ($D_1^c \cap D_2^c$) of R_1 and R_2 regular expressions.

We finally compute the complementary of the union DFA in figure 4.12.

The output regular expression R from the resulting automata can easily be computed, as most of the states are not reachable, leaving only two reachable states (q_0, p_0) and (q_1, p_1). Thus, in our example $R = T\_Z * T\_Y+$.

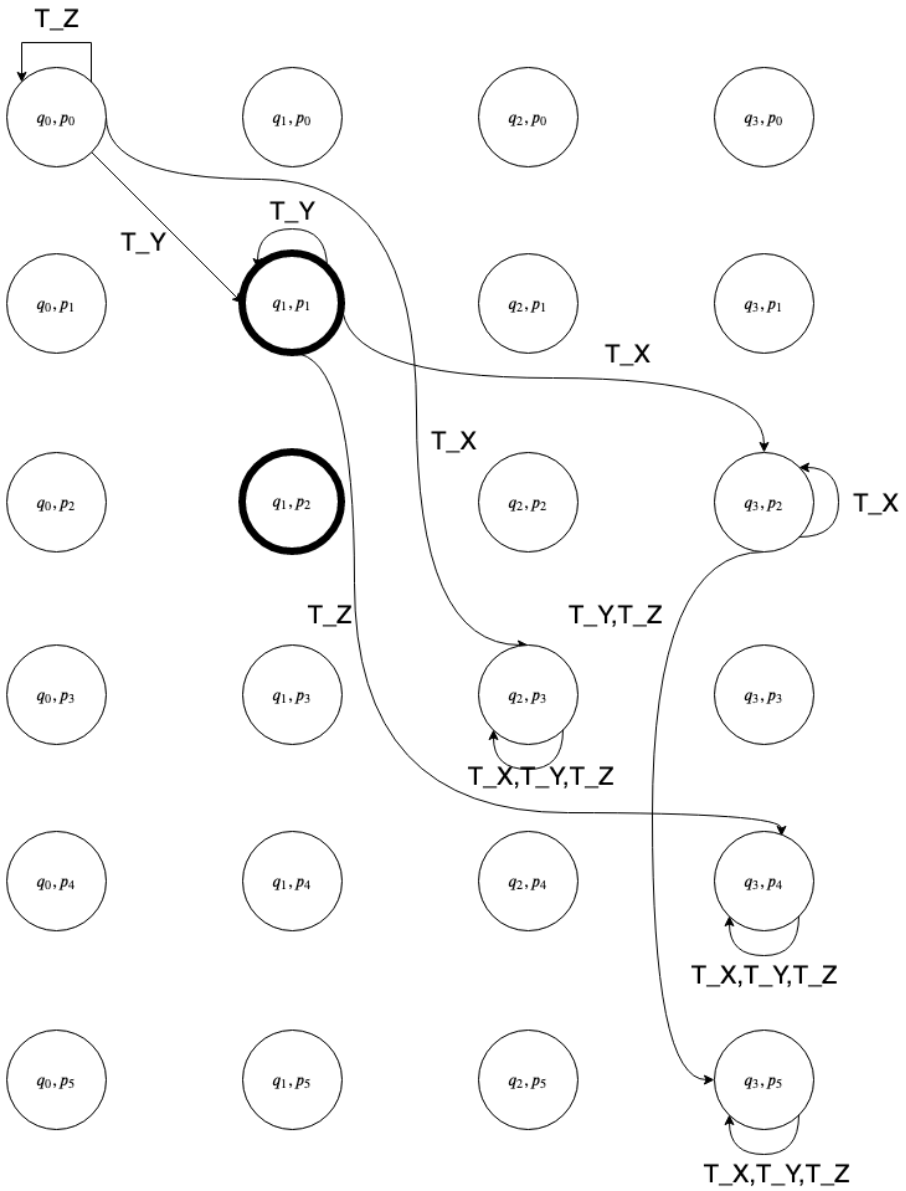


Figure 4.12: Complementary of the union of the complementary deterministic automata $(D_1^c \cap D_2^c)^c$ of R_1 and R_2 regular expressions.

4.2.5 Learning Action Expression

As defining action regular expression by hand can be tedious and is not intuitive, we propose a user-centered action regular expression learning algorithm.

Moreover, we propose a heuristic approach to learn user dependent action regular expressions from a small number of spatial motion doodle examples. Most of the time, three examples are sufficient. Learning a regular expression from examples is an ill-posed problem which can accept the composition (or operator) of all the training examples as a solution. In our approach, we build a compact regular expression which recognizes both examples as well as inputs that are very similar to them. This method can be seen as a special case of [Van Zaanen, 2000] which learns grammars from input sentences by aligning common words and derives grammar rules that can generate the inputs.

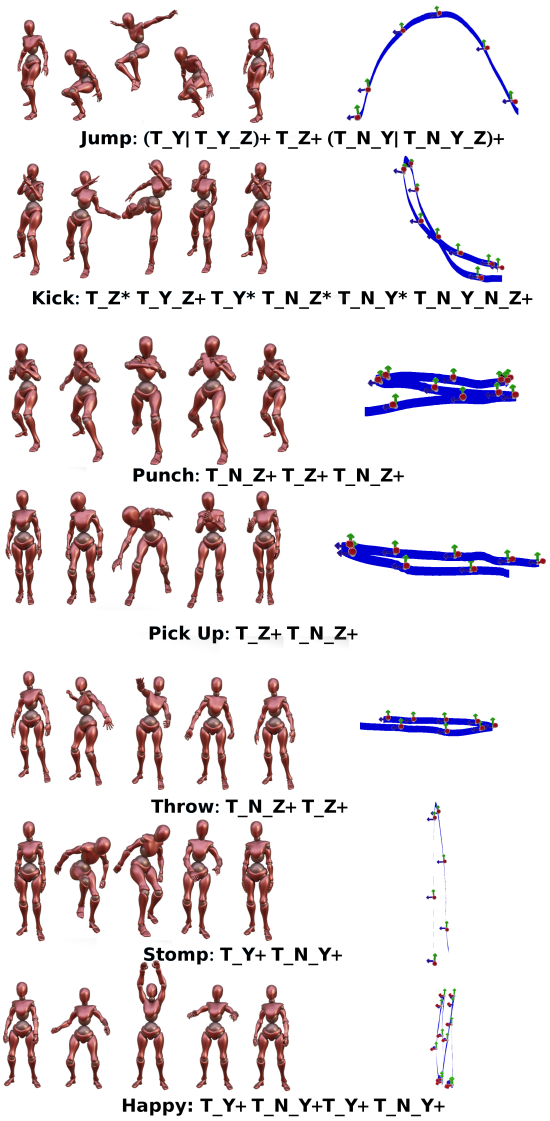


Figure 4.13: Subset of actions in our database with their related regular expressions

Given n examples representing the same action, we compute an average space time doodle using dynamic time warping with l_2 norm as distance function, following an approach first proposed by [Ciccone et al., 2017]. Using this technique, we find point correspondences between the input examples and then compute the average doodle by taking the mean of each correspondences. We then convert the n input doodles and the average doodle into $n + 1$ token sequences $\{X_i\}$.

Our learning algorithm is aimed to align tokens that are repeated more than twice for each sequences, considering them as mandatory in the final learned results (translated as a $+$ in a regular expression). Other tokens are considered as optional or unintentional gestures from the user (translated as a $*$ in

a regular expression). Thus, from each sequence $X_i = T_{0i}...T_{ni}$ we extract two subsequences : $M_i = m_{0i}...m_{pi}$ composed of mandatory tokens (preserving the original order) and $O_i = o_{0i}...o_{li}$ composed of optional tokens. For each o_i, we store the index of the next mandatory token in the X_i sequence as well. Then, our system parses mandatory sequences in parallel and build the first part of the final regular expression $R = r_0 + ...r_n+$ where $r_i = (m_{i0}|...|m_{in})$. Note that each mandatory sequence may have different length, consequently if $i \geq \min_j length(M_j), r_i$ will be treated as optional and r_i+ is replaced by r_i*. Finally, we add the remaining optional tokens into the final sequence using the following method: before each token r_i we insert the optional sequence $\widetilde{o_i} = ((o_{k_i0}...o_{(k_i+p_i)0})|...|((o_{k_in}...o_{(k_i+p_i)n}))*$ where $(o_{k_ij}...o_{(k_i+p_i)j})$ is the subsequence of X_j composed of optional tokens located between the $m_{(i-1)j}$ and m_{ij} mandatory token. Note that if all subsequences are equal then this optional token sequence is treated as a quick intentional (mandatory) gesture. Finally, we emphasize that, by construction, each training example is contained in the final regular expression R.

To be more familiar with how our learning algorithm works, lets consider the following Backward Jump action example (Figure 4.14) and mean curve.

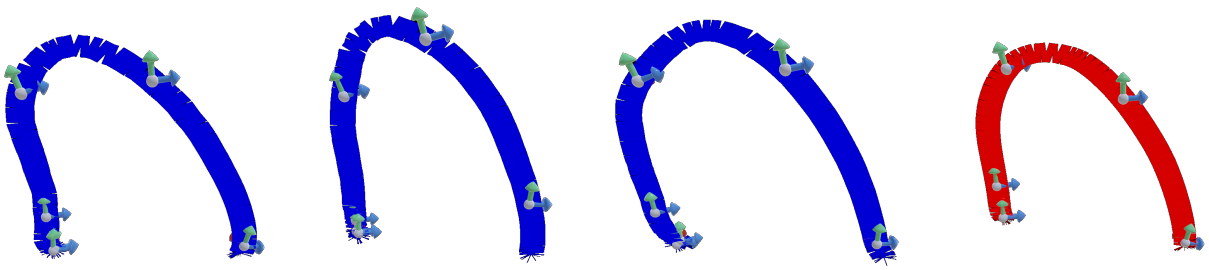


Figure 4.14: Backward Jump action example (blue) and mean (red) curves.

Our algorithm segments each input curve into its corresponding token sequence and remove all token successive occurrences that exceed 2 repetitions, clearly highlighting mandatory token from optional tokens. For instance the T_YT_Y pair from the first curve token sub-sequence (see below) is the result of the compression of the $T\_YT\_YT\_YT\_Y$ sequence. Below, we show each input curve related token sequence in which we highlighted mandatory tokens surrounded by a red frame. We then align and compose them while completing the inbetween part with the corresponding optional sequences.

T_YT_Y T_Y_N_ZT_Y_N_Z $T\_N\_ZT\_N\_Y\_N\_Z$ T_N_YT_N_Y
T_YT_Y T_Y_N_ZT_Y_N_Z $T\_N\_ZT\_N\_Y\_N\_Z$ T_N_YT_N_Y $T\_X\_N\_YT\_N\_X$
$T\_Y\_N\_ZT\_Y$ T_Y_N_ZT_Y_N_Z T_N_ZT_N_Z $T\_N\_Y\_N\_Z$ T_N_YT_N_Y
T_YT_Y T_Y_N_ZT_Y_N_Z T_N_ZT_N_Z $T\_N\_Y\_N\_Z$ T_N_YT_N_Y

$$\Downarrow$$

$((T\_Y\_N\_Z)*(T\_Y)*)$ $(T\_Y\_N\_Z \| T\_Y)+$ $(T\_N\_Z \| T\_Y\_N\_Z)+$ $((T\_N\_Z)*(T\_N\_Y\_N\_Z)* \| (T\_N\_Y\_N\_Z)*)$
$(T\_N\_Z \| T\_N\_Y)+$ $((T\_X\_N\_Y)*(T\_N\_X)* \| (T\_N\_Y\_N\_Z)*)(T\_N\_Y)*$

In the resulting regular expression we can notice that the last $T\_N\_YT\_N\_Y$ of the fourth expression has been translated to $(T\_N\_Y)*$ because the three other expressions contains only three mandatory tokens.

In section 4.2.6 we evaluated the efficiency of our algorithm which really proved efficient in practice. Moreover, such an algorithm greatly expend the usability of our system since each user can author animation sequences with his own gestures which also make action patterns easier to memorize.

4.2.6 Evaluation

We evaluated the efficiency of our recognition algorithm and learning algorithm through several experiments. More precisely, we evaluated the capacity of our to system to recognize users intended actions.

As a first experiment, we asked users to execute predefined action patterns 10 times, after showing them the recorded physical props motion of a typical example. We computed the success rate per action category, measured as the percentage of executions that indeed matched the target regular expression. Results are summarized in Figure 4.15.

These results show that our system achieves good recognition results with respect to the user intent. Moreover, walking and jumping actions were always recognized as intended. More interestingly the Punch and Throw, which were the actions with the most recognition failures, still have more than 75% of recognition rate. Given that the regular expression for Throw is a sub sequence of the regular expression for Punch, this shows that our system is not confused by action similarity and that the policy we follow for overlapping actions proved efficient.

In a second experiment, we evaluated the efficiency of our regular expression learning algorithm by asking 5 users to train our system for 3 different actions

(each user had a different action set to train). Users were then asked to test the system thrice performing short SMD containing only 2 or 3 actions, and several times with longer sequences (at least more than 5 actions). On simple doodles, our system achieved 94% of correct recognition rate. This number decrease to 90% on longer sequence which is still high. This small drop rate is to be expected as it is more difficult for users to avoid doing mistakes on longer sequences. The study revealed that motion involving rotations were more difficult to recognize, especially when users rotate the controllers away from their center of gravity, which induces unintended translations.

Overall, these experiments showed the usability of our system which best performs when users use their own gestures when recording. In addition, we also observed that it is still the case when using both hands (see chapter 6). It also highlights that the ergonomic aspect of the props must be taken into account when choosing action patterns, that is why in our case only few gestures contains rotation tokens.

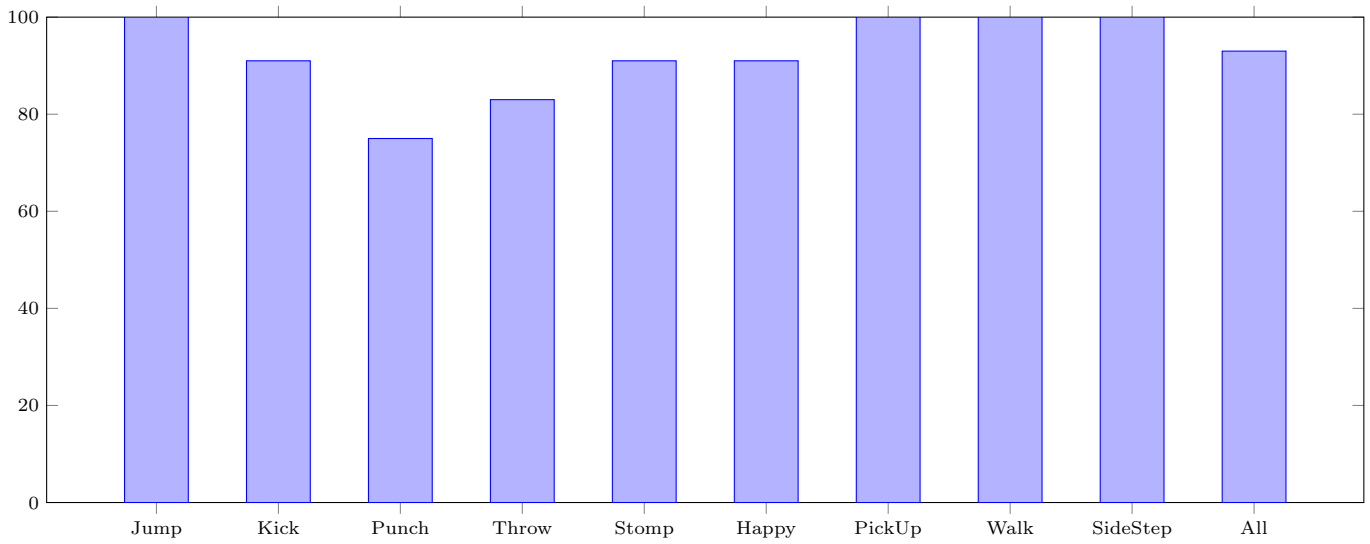


Figure 4.15: Percentage of actions correctly recognized during our user study.

4.2.7 Conclusion

We presented a new algorithm performing action recognition from hand gesture trajectories. Our algorithm behave like a compiler using lexical and syntax analysis based on a motion token vocabulary. The motion descriptor we introduced is translation, rotation and scale invariant which gives staging freedom to users. We also proposed an action regular expression learning algorithm which let users use their own gesture when recording and which proved very efficient at recovering their intents. This method also comes with some limitations for dealing with overlapping actions as well as actions whose regular expressions

are very close. In the future, it would be interesting to further limit the confusion due to overlapping by enhancing the approximate Levensthein expression matching using a token distance which is more relevant of their spatial representation.

In section 4.3 we describe how we performs motion quality recognition, extracting additional geometrical features from the SMD to perform Laban Effort classification (section 4.4).

4.3 Motion Quality Recognition

In section 4.2, we presented an algorithm which recognizes action sequences from figurines rigid motions. Another challenge we tackle to achieve our main goal from chapter 1 is to recognize expressiveness from the same rigid motions. In our work, we qualify motion expressiveness in terms of Laban Effort qualities divided into their Space, Time and Weight Factors. In this section, we introduce a new method for recognizing Laban Effort qualities from the same trajectories we use for action recognition.

As mentioned in chapter 2, identifying Laban Effort from animations has already been studied, especially for full body animations. [Fdili Alaoui et al., 2017] also focused on using only data provided by sensor tracking the wrist velocity, acceleration, jerk and muscle activity to classify motion into the Laban Effort space. However, their results show that there is still room for improvement as recognizing Drives remains challenging. They also showed that using muscle activity in addition to kinematic properties, greatly improved their classification results. To cope with this missing data and hoping to improve the Drives classification rates, we propose to study a larger number of kinematic features from the input trajectories. Using this larger set of features we present two different classification methods and discuss their recognition rates. We first present the features we extract from the input trajectories before studying Laban Effort classification using a Naive Bayes classifier and an HMM-based classifier.

In the same manner as [Masuda et al., 2010] and [Fdili Alaoui et al., 2017] we based our feature choice on a priori assumptions of which feature will give information on which Effort quality. Moreover, we related Torsion τ and Curvature κ to the Space dimension, Speed (linear and angular) V_e, Acceleration A_e and Jerk J_e to the Time dimension and Equi-affine Speed V_a and Acceleration A_a to the Weight dimension. Let us detail each feature and explain these choices.

Time features. Time opposes Sudden to Sustained motions which present two major differences: Sudden motions are globally quicker than Sustained motions and they also are characterized by a great impulse (jerk peak) at the beginning of the motion while Sustained motion do not present any impulse during the entire motion. We then relate this difference in behavior to variation of speed $V_e = (v_e, \omega)$, containing linear and angular velocity information (ω defined as in section 4.2.2 and $v_e = \frac{p_i - p_{i-1}}{t_i - t_{i-1}}$), as well as its two successive derivatives $A_e = (\dot{v_e}, \dot{\omega})$ and $J_e = (\ddot{v_e}, \ddot{\omega})$ computed using finite differences. Those features are expressed in the global world space. Furthermore, we also

consider speed expressed in the moving frame and its derivative $v, \dot{v}, \ddot{v}$ (with $v = R_i(p_i - p_{i-1})$ like in section 4.2.2) as being part of our feature set as they provide orientation invariant information.

Space features. The Space Effort opposes Direct to Indirect motions whose differences can be simply illustrated by two paths which go from a starting point A to a finishing point B. As illustrated in figure 4.16, the Direct motion from A to B is a strait line while the corresponding Indirect motion takes detours before reaching B. Geometrically speaking, those two curves have very different curvatures (and torsion in the case of 3D) all along the path. Curvature κ and torsion τ are mathematically defined as :

$$\kappa = \frac{|v_e \times \dot{v}_e|}{\|v_e\|^3} \qquad \tau = \frac{[v_e, \dot{v}_e, \ddot{v}_e]}{\|v_e \times \dot{v}_e\|^2}$$

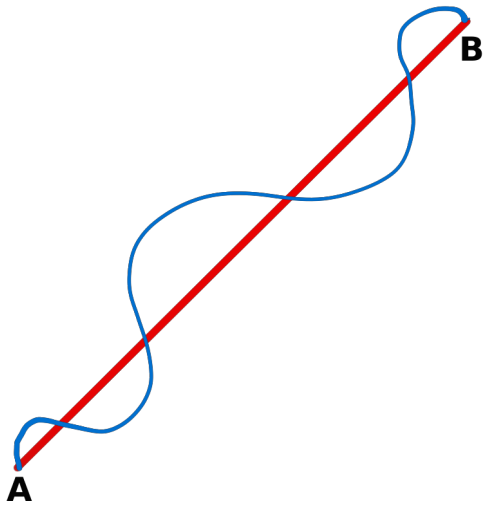


Figure 4.16: Direct (red) and Indirect (blue) paths from A to B.

The curvature of the path at a given point can be interpreted as the inverse of the radius of the circle which best approximate the curve at this point while torsion gives information about the amount by which the curve is leaving the plane which best contains it at the given point with respect to the plane normal. Consequently, each curvature and torsion variation implies that the trajectory is deviating, to some extend, from its current shape which we relate as an Indirect behavior. In figure 4.16, the Direct trajectory has a null curvature and torsion while the Indirect path presents high curvature variations.

Weight features. Weight opposes Strong to Light motions which mostly differ in the uniform aspect of motions as well as their amplitude. In our case, in which we only have figurine rigid motion, we focus on quantifying the uniformity of the motion. It turns out that the equi-affine velocity and its variation gives us such information. In 2D, the equi-affine velocity of a point moving along a planar trajectory is defined as $V_a = |v_e \times \dot{v}_e|^{\frac{1}{3}}$ and is related to the Euclidean velocity with the formula $V_a = v_e \kappa^{\frac{1}{3}}$. Equi-affine velocity is important in movement studies because it has been shown that human subjects spontaneously draw with constant equi-affine velocity. This is known as the 1/3 power law [Pollick and Sapiro, 1997]. In 3D, equi-affine velocity is defined in a similar fashion using the triple scalar product $V_a = |v_e, \dot{v}_e, \ddot{v}_e|^{\frac{1}{6}}$ which is again related

to the Euclidean velocity with the formula $V_a = v_e \kappa^{\frac{1}{3}} \tau^{\frac{1}{6}}$. Similarly to the $2D$ case, recent work has shown that human subjects describe spatial movements performed with equi-affine velocity as uniform [Pollick et al., 2009] and this is known as the 1/6 power law. Using this definition, we relate Light motions as having a constant equi-affine velocity unlike Strong ones. Therefore, in addition to the equi-affine velocity, we also compute equi-affine acceleration as the first derivative of V_a.

In the next section, we describe how we perform Laban Effort classification using these geometric properties.

4.4 Laban Effort Classification

We studied Laban Drive classification through two different machine learning methods with two different dataset of labeled SMDs. We first trained and evaluated the classification rates of a Naive Bayes classifier in section 4.4.1. Then, we also studied Laban classification using a Hidden Markov Model based classifier (section 4.4.2). Eventually, we compared the efficiency of the two methods relatively to each other in section 4.4.3.

4.4.1 Naive Bayesian Classification

Naive Bayes classification is a commonly used machine learning method which relies on the Bayes theorem and the assumption that all features used for classifying are independent. Given two events A and B the Bayes theorem can be written:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{4.6}$$

In our Laban Drive classification case the event A corresponds to $C(T) = d$, the class of an input trajectory T is equal to a specific Drive d. The B event corresponds to $F = f_1, ..., f_n$, the features considered for classification have the values computed from the input trajectory. Equation 4.6 can be rewritten:

$$P(C(T) = d|f_1, ..., f_n) = \frac{P(C(T) = d)P(f_1, ..., f_n|C(T) = d)}{P(f_1, ..., f_n)} \tag{4.7}$$

This expression can be further extended by noticing that $P(f_1, ..., f_n|C(T) = d)$ can be rewritten $P(f_1|C(T) = d)P(f_2, ..., f_n|C(T) = d, f_1)$ using the definition of conditional probability ($P(A \cap B) = P(A)P(B|A)$). By reiterating the operation $n - 1$ times we obtain:

$$P(f_1, ..., f_n|C(T) = d) = P(f_1|C(T) = d)P(f_2|C(T) = d, f_1)...P(f_n|C(T) = d, f_1, ..., f_{n-1}) \tag{4.8}$$

This equation is still difficult to compute as it holds with a lot of conditional dependencies. This is where the hypothesis of feature independence intervenes for $(i, j) \in [1 : n]$ with $i \neq j, P(f_i|C(T) = d, f_j) = P(f_i|C(T) = d)$. Thus equation 4.8 can be simplified as:

$$P(f_1, ..., f_n|C(T) = d) = \prod_{i=1}^{n} P(f_i|C(T) = d) \tag{4.9}$$

Having simplified the first equation 4.7, we must choose a method to actually classify the input curve into one of the Laban Drive with respect to the probability of a given input to belong to a given class. In this work, we chose the maximum a posteriori method consisting in choosing the maximum probability among all the classes' a posteriori probability:

$$C(T) = \arg\max_{d} P(C(T) = d) \prod_{i=1}^{n} P(f_i|C(T) = d) \tag{4.10}$$

Note that $P(f_1, ..., f_n)$ is ignored as it is a constant shared by all the classes. The remaining task is to determine $P(C(T) = d)$ (a priori probability law) and all $P(f_i|C(T) = d)$ (likelihood). It is important to point out that we actually use three Naive Bayes classifiers, one for each Effort axis (*Space* axis (Direct - Indirect) *Time* axis (Sudden - Sustained) and the *Weight* axis (Light - Strong)), and combine their classification results in order to perform Drive classification. Thus for each classifier we choose a uniform probability law as the a priori law, resulting in:

$$P(C(T) = d) = \frac{1}{2} \tag{4.11}$$

as there is two classes per Effort axis.

Finally, we chose Normal distribution laws as likelihood probability laws resulting in:

$$\forall i \in [1 : n], P(f_i|C(T) = d) = \mathcal{N}(f_i, \overline{f_i^m}, \sigma^2_{f_i^m})) = \frac{\exp^{\frac{(f_i - f_i^m)^2}{2\sigma^2_{f_i^m}}}}{\sqrt{2\pi\sigma^2_{f_i^m}}} \tag{4.12}$$

where $\overline{f_i^m}$ is the mean of the feature f_i over all the training examples whose class is d and $\sigma^2_{f_i^m}$ the corresponding standard deviation.

The three classifiers were trained using a dataset of spatial motion doodles labeled with Laban Space, Time and Weight qualities. We used a total of 200

training examples, captured by asking 5 users to move a rigid object in the eight different combinations of the *Space Time* and *Weight* qualities. This was done five times for each combination.

Feature Selection. In order to improve classification results and to validate our choice of features, we propose to extend the Naive Bayes classification with a feature selection process. This process aims to remove features that brings confusion and redundancy when classifying between n classes (in our case $n = 2$). To choose which features to keep, we proceed to a feature ranking a keep the features of higher ranks. Several feature ranking methods exist in the literature [Novakovic, 2010] but, to be consistent with our feature independence hypothesis we choose the relevance factor proposed by [Pink and Ramanathan, 2000]. This relevance factor relies on computing the intra-class and inter-class variances of each feature and use their ratio as ranking factor.

For a given feature f, its intra-class variance $\sigma_{intra}(f)^2$ indicates how much the feature varies within all classes. Having a high intra-variance is not desirable as it shows that the feature f is not representative for all the classes. On the other hand inter-class variance $\sigma_{intra}(f)^2$ represents the variability of f between all the classes and more precisely it quantifies how much all classes are separated from each other with respect to f. Thus a higher value is preferable as it shows how well this feature separate the different classes. Statistically speaking, the intra-variance represents the mean of all the class variance with respect to f while the inter-variance is the variance of all the class means with respect to f. The final relevance of the feature f is computed as:

$$Rel(f) = \frac{\sigma_{inter}(f)^2}{\sigma_{intra}(f)^2} \tag{4.13}$$

The higher $Rel(f)$ is the more relevant the feature f is. Those two variances must be computed with all feature values being normalized beforehand (meaning that all f values are in $[0 : 1]$).

Thus, considering n classes and m_i examples in the i^{th} class, we compute for each feature f:

$$\overline{f} = \frac{\sum\limits_{i\in[1:n]} \sum\limits_{j\in[1:m_i]} f_{ij}}{\sum\limits_{i} m_i} \tag{4.14}$$

$$\forall i \in [1, n], \overline{f_i} = \frac{\sum\limits_{j\in[1:m_i]} f_{ij}}{m_i} \tag{4.15}$$

followed by the corresponding inter-variance and intra-variance:

$$\sigma_{inter}(f)^2 = \frac{\sum\limits_{i\in[1:n]} m_i(\overline{f_i} - \overline{f})^2}{\sum\limits_i m_i} \tag{4.16}$$

$$\sigma_{intra}(f)^2 = \frac{\sum\limits_{i\in[1:n]} \sum\limits_{j\in[1:m_i]} (f_{ij} - \overline{f}_i)^2}{\sum\limits_i m_i} \tag{4.17}$$

Using our training data, this feature selection enabled us to evaluate which features of the SMDs are the most relevant for Laban classification and remove confusing features. For those three Naive Bayes classifier we used a set of 50 features extracted from the geometric feature described in section 4.3. Moreover we computed the mean and standard deviation of $v, \dot{v}, \ddot{v}, \|V_e\|, \|A_e\|, \|J_e\|, V_a, A_a$, κ and τ.

We computed all those 50 features for all labeled training examples, and incrementally trained classifiers for each task using the M most relevant features for that task with M ranging between 1 and 50. We then chose the classifier with the best accuracy for each task, resulting in 7 features for recognizing Space, 4 features for recognizing Time and 2 features for recognizing Weight as reported in table 4.2.

Space relevance	Time relevance	Weight relevance
Torsion Mean	Local Accel Up Mean	Equi-Affine Speed Mean
Acceleration Mean	Local Jerk Up Mean	Equi-Affine Accel Mean
Speed Mean	Acceleration Stdev	
Equi-Affine Speed stdev	Local Speed Up Mean	
Local Accel Up stdev		
Local Speed Up stdev		
Local Jerk Up stdev		

Table 4.2: Relevance (decreasing order) of SMD features for characterizing the Space, Time and Weigh dimensions of Laban's Effort.

The relevance results presented in table 4.2 is consistent with the feature-effort association assumptions we made in section 4.3. Moreover, Torsion is the most relevant feature for Space, Speed Acceleration and Jerk are highly relevant for Time and Equi-Affine Speed and its derivative are also relevant for Weight. On the other hand, we notice that the other relevant feature of the Space Effort are not coherent with our a priori definition Direct and Indirect qualities. This observation can be the consequence of some bias that is present in the training

examples. In addition, the Curvature is unexpectedly not present in the selected feature set of the Space dimension which can also reveal the presence of bias in our training set which may not include enough examples with varying curvatures.

Naive Bayes classification results. We tested our three Naive Bayes classifiers independently as well as their combination for Drive classification. The tests were evaluated using 24 curves which are not part of the training examples and are labeled in the 8 different Drives. In figure 4.17 we compare the results obtain with and without feature selection and notice the efficiency of performing feature selection. Indeed this additional process increased significantly the classification rates both for individual Efforts and Drives. It is worth noticing that as highlighted by [Fdili Alaoui et al., 2017], Drives classification is an hard problem as we only perform 35% of successful recognition at most. On the contrary, our individual classifiers performs quite well with all recognition rates above 74%.

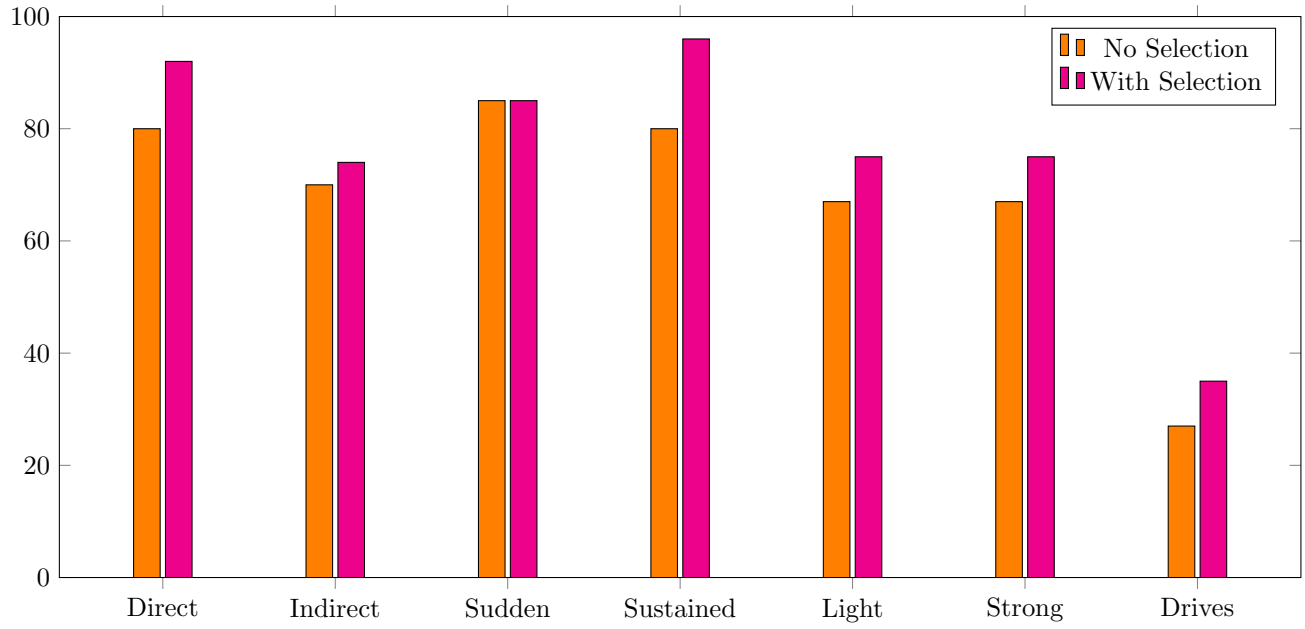


Figure 4.17: Naive Bayes Laban Effort classification accuracy in percentage before and after feature selection.

We also observed that classification results were worse when dealing with action trajectory segment of SMDs. Moreover Drives classification drops back to 27% even with feature selection. This questions the scalability of the Naive Bayes classification as well as the set of training and testing examples we chose for this experimentation. Those two curve sets might also presents some bias as they were recorded by non LMA experts. Overall, the low recognition rate of the Drives classification can also reflect a limitation to recognize Laban Drives with only single trajectory data. In the next section we describe how we perform

Laban Effort classification using HMM based classifier, which copes with the scalabity problem we faced. We also introduce new training and testing sets that are more consistent that the ones we used for this first experiment.

4.4.2 HMM Classification

In this section we describe how we perform Laban Effort classification using Hidden Markov Model (HMM) which is also a well known statistical model best suited to represent temporal sequences' behaviors. As SMDs are temporal sequences of 6D points, choosing HMM to perform Laban Effort classification is an appropriate decision.

The structure behind HMM classification is different from the Naive Bayed classifiers we presented in section 4.4.1. Sequential sequences behaviors are represented as statistical automata, with n states $\{S_i\}$ and transition between them $\{a_{ij}\}$ which are bound to probability of occurrence such that: $\forall i \in [1 : n], \sum_j a_{ij} = 1$.

Each state is associated with a probability law which gives statistical information (usually mean and standard deviation of features extracted from the input sequences) of the modeled behavior at this state. In our case this probability law will be very similar to equation 4.9 using also Normal distributions. Given each states probability distributions and transitions as well as an input sequence and its related feature vectors, it is possible to compute the likelihood that the input follows the underlying behavior model. For instance, let's suppose that we can segment input SMD into different distinctive parts (for instance start/middle/end), we relate each part to one state characterizing behavior changes. When training an HMM we must observe these behaviors changes over time in the related feature sequences. If we take the examples of SMDs, it is a reasonable assumption to consider that most of them have an acceleration phase, a more constant phase and a decelerating phase. They can thus be modeled with 3 states.

Let's further illustrate SMD HMM classification by considering that the states have only left-to-right transition, meaning that one state can only transit into itself or its right neighbor. Finally, we study SMD behavior only based on the Speed norm $\|V_e\|$. The corresponding HMM looks like:

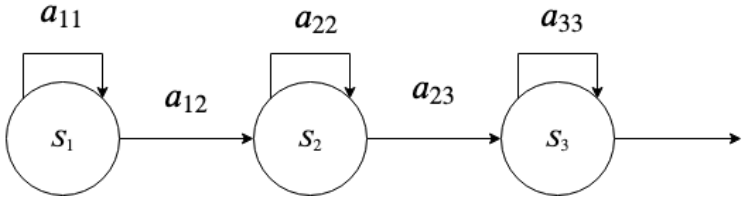


Figure 4.18: Left to Right 3-States HMM representing SMDs' behavior. State's probability distributions and transitions are unknown.

We then need to determine all transition probabilities as well as probability laws of each states. This is done using training data and its related feature data, let's consider the three following SMD speed sequences examples:

$$SMD1 : 0.01\ 0.5\ 1.1\ 2.4\ 2.5\ 2.7\ 2.5\ 2.0\ 1.2\ 0.3\ 0.002$$
$$SMD2 : 0.1\ 0.4\ 1.0\ 1.7\ 2.3\ 2.8\ 2.7\ 2.8\ 2.4\ 1.5\ 0.3\ 0.003$$
$$SMD3 : 0.01\ 0.8\ 2.0\ 2.8\ 3.0\ 2.7\ 1.9\ 0.4\ 0.0001$$

Notice that each SMD starts with a speed near zero, accelerates until a more stable phase in the middle, and finally ends with a deceleration phase. Furthermore, we can initially decompose each of those sequences into 3 parts of equal sizes, giving a first position of each state in the three sequences:

$$SMD1 : 0.01\ 0.5\ 1.1\ 2.4|\ 2.5\ 2.7\ 2.5\ 2.0|\ 1.2\ 0.3\ 0.002$$
$$SMD2 : 0.1\ 0.4\ 1.0\ 1.7|\ 2.3\ 2.8\ 2.7\ 2.8|\ 2.4\ 1.5\ 0.3\ 0.003$$
$$SMD3 : 0.01\ 0.8\ 2.0|\ 2.8\ 3.0\ 2.7|\ 1.9\ 0.4\ 0.0001$$

Then, using statistical inferring methods such as Baum-Welch algorithm [Baum and Petrie, 1966], the border between state will be rectified such that speed values that are outliers with respect to their current state group values, are placed in the best matching state, where values are close to each other, resulting in:

$$SMD1 : 0.01\ 0.5\ 1.1|\ 2.4\ 2.5\ 2.7\ 2.5\ 2.0|\ 1.2\ 0.3\ 0.002$$
$$SMD2 : 0.1\ 0.4\ 1.0|\ 1.7\ 2.3\ 2.8\ 2.7\ 2.8\ 2.4|\ 1.5\ 0.3\ 0.003$$
$$SMD3 : 0.01\ 0.8|\ 2.0\ 2.8\ 3.0\ 2.7\ 1.9|\ 0.4\ 0.0001$$

Taking a closer look to the values in the SMDs first states, we observe that 3 values are marking the transition with the second state while the other 5 are not adjacent to the border. We can then deduce that the probability of transitioning between the first state and the second one is the same as reading one of those three values among the other ones part of the first states $\Rightarrow a_{12} = \frac{3}{8} = 0,375$.

By repeating the same operation on the two remaining state, we obtain the following updated HMM:

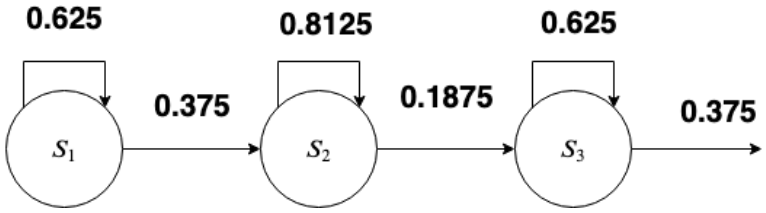


Figure 4.19: Left to Right 3-States HMM representing SMDs' behavior with computed transitions.

Finally, we can retrieve the probability laws (as normal laws) for each states by computing the mean and standard deviation of all values in each groups:

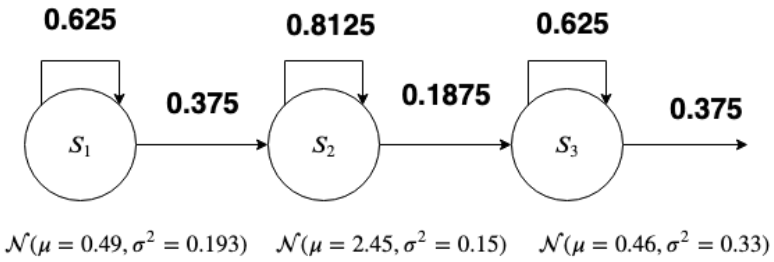


Figure 4.20: Left to Right 3-States HMM representing SMDs' behavior with computed transitions and State probability laws.

Finally, if we consider a new SMD speed input 0.01 1.7 2.8 0.1, we can estimate its likelihood with respect to the HMM model (with the hypothesis that the initial state is S_1):

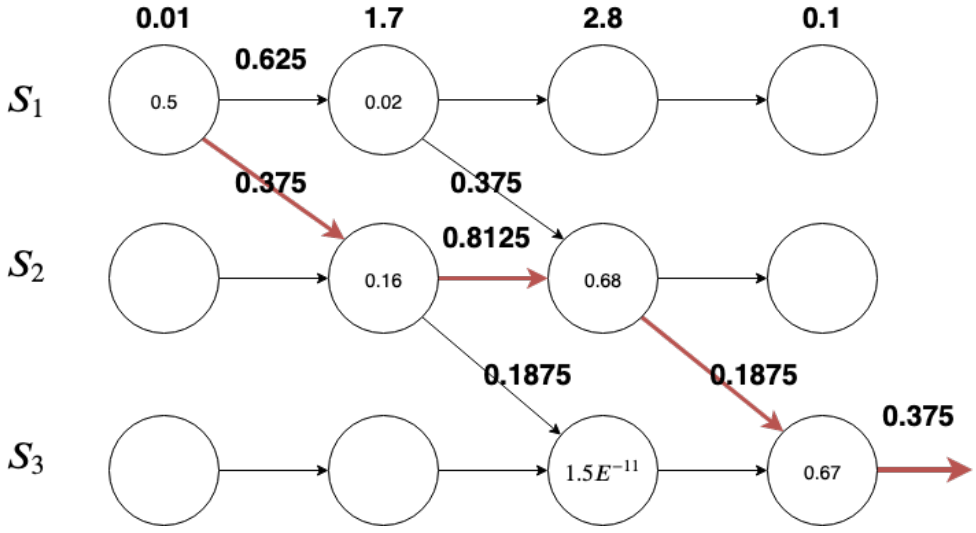


Figure 4.21: SMD likelihood computation using the forward algorithm.

The likelihood is obtained by taking the path with the highest probability resulting in $0.5 * 0.375 * 0.16 * 0.8125 * 0.68 * 0.1875 * 0.67 * 0.375 = 0.0007808$; this is known as the forward algorithm. This likelihood gives an estimation

that the input follows the SMDs' behavior with used for training.

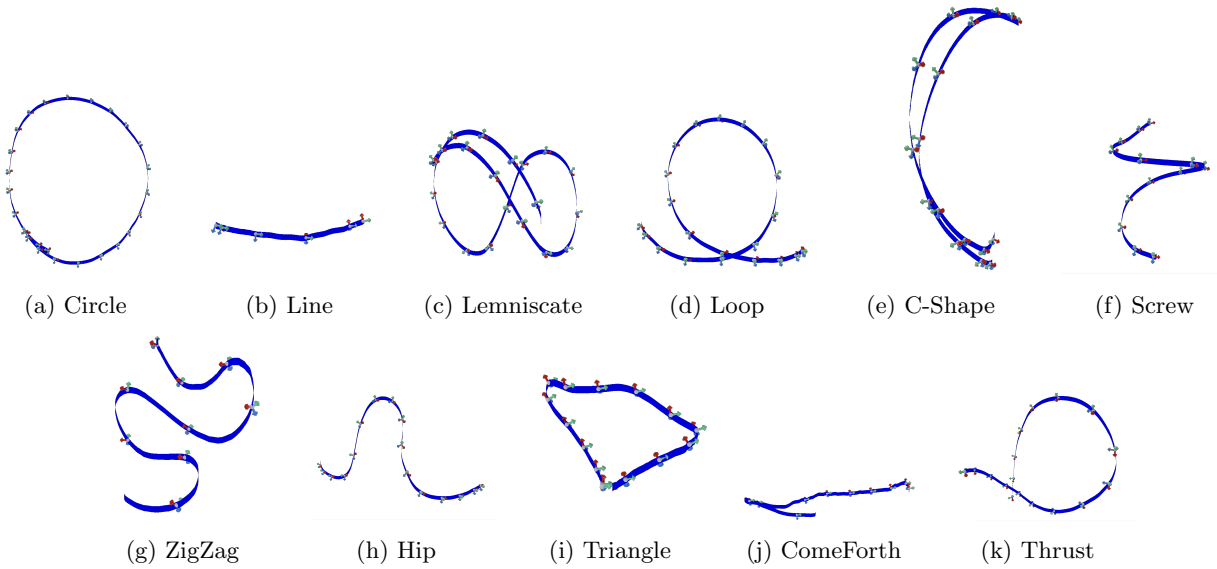
Furthermore, HMM classification can be done by training one HMM per class and as in section 4.4.1 we consider the maximum of likelihood as the class of the input SMD.

In this thesis, the HMMs used for classification have 6 left-to-right states and the feature used for learning the models were also $v, \dot{v}, \ddot{v}, \|V_e\|, \|A_e\|, \|J_e\|, V_a, A_a, \kappa$ and τ. Those features are grouped into 3 different sets as referred in table 4.3. The Euclidean set contains Speed, Acceleration and Jerk norms in the global space while the Moving Frame set contains local Speeds, Accelerations and Jerks. The Equi-Affine set is composed of the Equi-Affine Speed and Acceleration as well as the Torsion and Curvature.

Euclidean Set	Equi-Affine Set	Moving Frame Set
$\|V_e\|$	V_a	v
$\|A_e\|$	A_a	$\dot{v}$
$\|J_e\|$	κ	$\ddot{v}$
	τ	

Table 4.3: Feature sets used for Laban Effort classifications using HMMs.

Data Registration. For this new Laban Effort classification process, we build a more consistent dataset performed by 3 people including 2 LMA experts. A total of 104 curves has been gathered in which 88 are predefined pattern expressed in the 8 different Drives, and 16 are free style Drives curves. This choice strengthen recognizable action pattern coverage as well as over-fitting and bias avoidance thanks to the presence of freestyle curves. The 11 predefined patterns are presented in figure 4.22. For classification results, 56 (7 per Drive) curves have been used for training and 48 for testing. Furthermore, we enhanced our training dataset using data augmentation by sub-sampling each training curve 16 times for a total of 840 curve set or 105 per Drives. Note that each training and testing curve is re-sampled uniformly such that each curve has the same point number (512 in our case).



(a) Circle (b) Line (c) Lemniscate (d) Loop (e) C-Shape (f) Screw

(g) ZigZag (h) Hip (i) Triangle (j) ComeForth (k) Thrust

Figure 4.22: 6D Patterns used for training our HMM-based classifiers.

We then designed two methods to achieve Effort classification: the first one consist in training 8 HMMs, one per Drive and take the maximum of the likelihood returned by the forward algorithm (as explained in figure 4.21) among all HMMs (referred as the Drive HMM classification method). The second method consist in training 6 HMMs, one per Laban Element among the Space, Time and Weight Factors, and classify the input curve in each Factor before combining the three results as a Drive. The later method is referred as the Element HMM classification.

For each method, we performed 3 classification processes using the feature sets described in table 4.3. For the first classification, we used only Euclidean properties as classification features. Per Drive classification performed 25% of successful classification rate against 18% for the Element method. Adding equi-affine properties to our set of feature, we observed significantly improved classification results with 43% success rate for the per Drive method against 25% for the Element one. Finally, best results were achieved when also adding Motion Frame related properties to our feature set, achieving 44% rate for the per Drive method and 25% for the composed method, even if this time the improvement is really subtle. In figure 4.24 we show the confusion matrices of the two methods and figure 4.23 summarizes the different classification results.

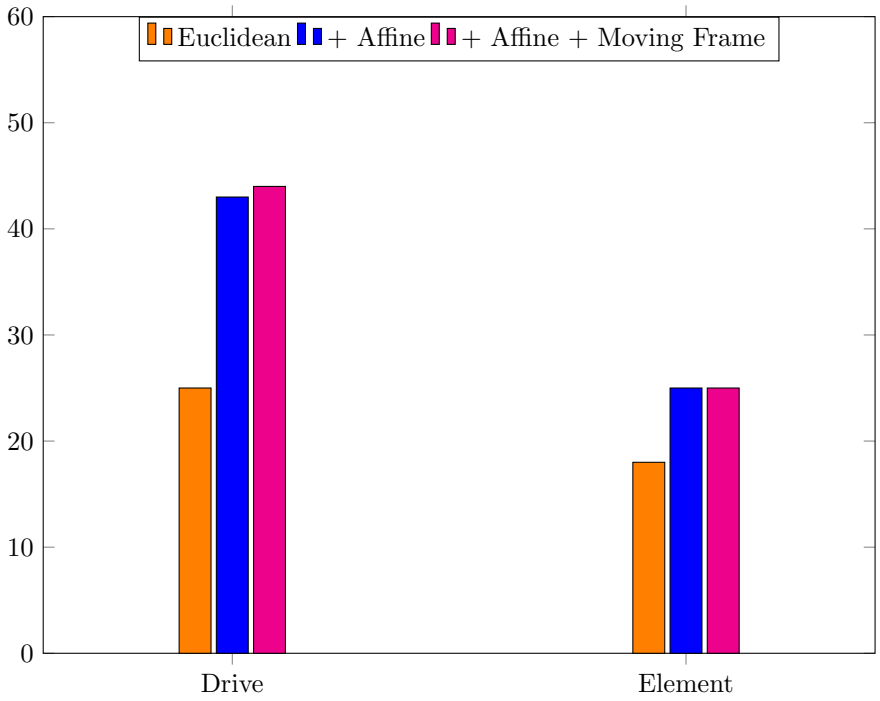
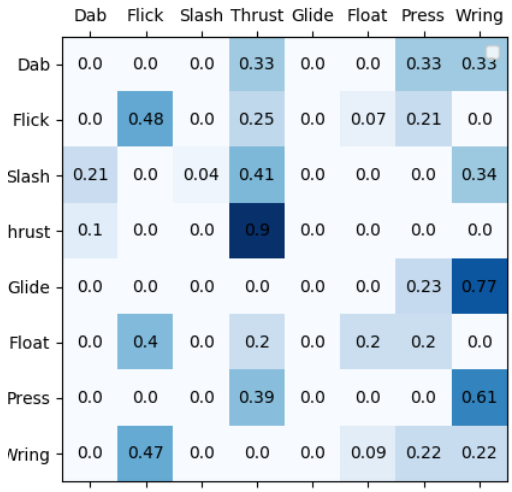
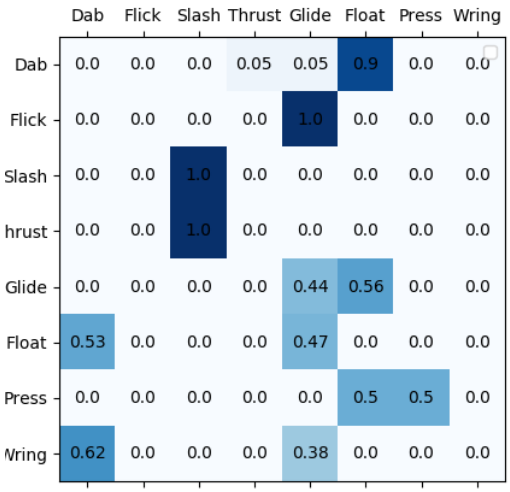


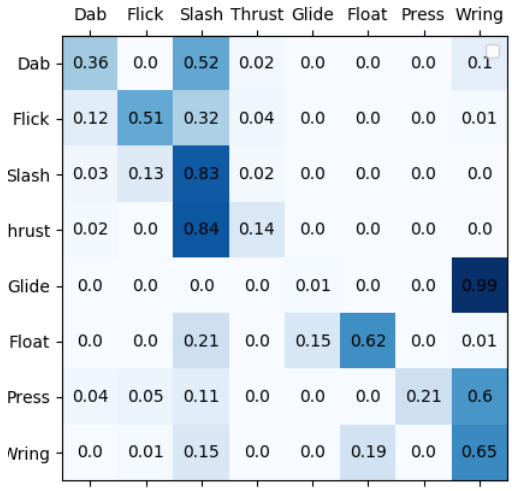
Figure 4.23: Laban Drive classification accuracy in percentage using the Drive HMM classifier and the Element HMM classifier.



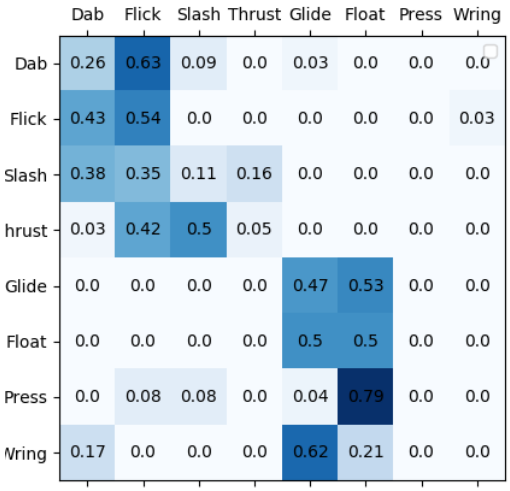
(a) DHMM Euclidean. Accuracy: 0.249



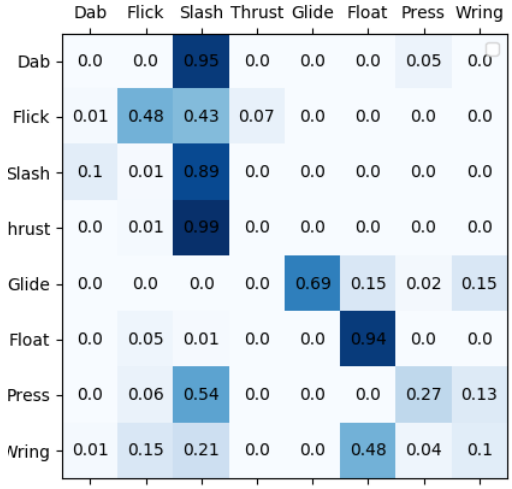
(b) CHMM Euclidean. Accuracy: 0.185



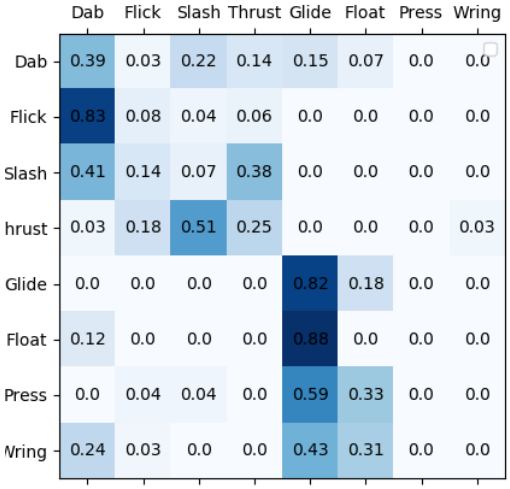
(c) DHMM Euclidean + Affine. Accuracy: 0.426



(d) CHMM Euclidean + Affine. Accuracy: 0.251



(e) DHMM Euclidean + Affine + MF. Accuracy : 0.439



(f) CHMM Euclidean + Affine + MF. Accuracy: 0.25

Figure 4.24: Confusion matrices. Left: Classification confusion using one HMM per Drive. Right: Classification confusion using one HMM per Effort element.

Those first results emphasize that the per Drive classification method outperforms the Element one with classification results which are almost twice better using all features. We also notice that our per Drive method gives better classification results than [Fdili Alaoui et al., 2017]. However, our classification results cannot be directly compared because our testing sets differs. In future experiments, it would be highly relevant to compare classification results using common testing sets.

On the other hand, even if per Drive classification performs better than the Element method, 44% of successful recognition is still quite low, making our classifier missing the correct Effort every other time. To better understand where the error is coming from, we computed the classification rates of the individual Laban Factor (Space, Time and Weight) using the three feature sets of the previous experiment for the two methods. We depict those results in figure 4.25 and 4.26. We first observe that Time is highly recognizable in both methods with classification results above 85% when using all features. On the contrary, Space and Weight Factors are causing Drive recognition rates to drop. In the case of the per Drive methods we can also notice that the Space Factor is the most difficult dimension to classify. As mentioned earlier, the lower classification rates of the Space and Weight Factors might also reflect a lack of additional features which are not extracted from the input trajectories, like the electric activity of the wrist [Fdili Alaoui et al., 2017] used in their experiments.

To better evaluate the impact of the Space Factor in our classification results we removed it from the classification results and computed the State classification rates for the per Drive and per Element methods. These results are very interesting as they highlight that the two methods give almost identical results when considering only the Time and Weight Factors. More precisely, the best results are obtained by combining the Euclidean and Affine feature sets in both cases with 63% of successful recognition for the per Drive method and 62% for the per Element one. We observe a small drop of successful classification rate when considering all feature sets for the two methods (61% for the per Drive method and 58% for the per Element one) indicating that adding the moving frame feature set brings confusion which we can observe for the Time Factor on figure 4.25 and 4.26.

In addition, the similarity of the classification results between the two methods reveals that the per Element method fails to classify curves in terms of Space when it correctly classified these curves in terms of Time and Weight which is less the case for the per Drive method.

As we will justify in section 4.4.3, we will not be able to classify actual SMD with respect to the Space Factor, making these new classification results in terms of Time and Weight Factors more representative of our ability to capture users' hand motion qualities. This restriction works in our favor as we observe an increase of 20% between the Drive and State classification. Still, those results are not completely satisfying as we miss the correct motion quality twice out of five.

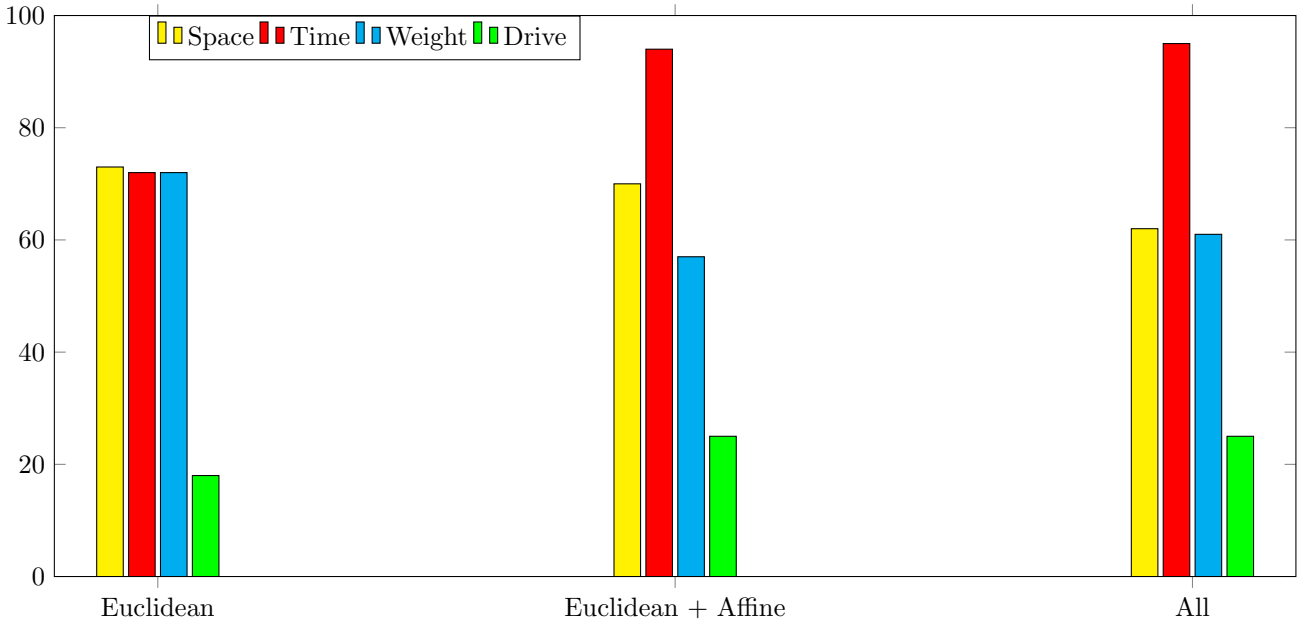


Figure 4.25: Element HMM classification results on individual Efforts and on composed Drives.

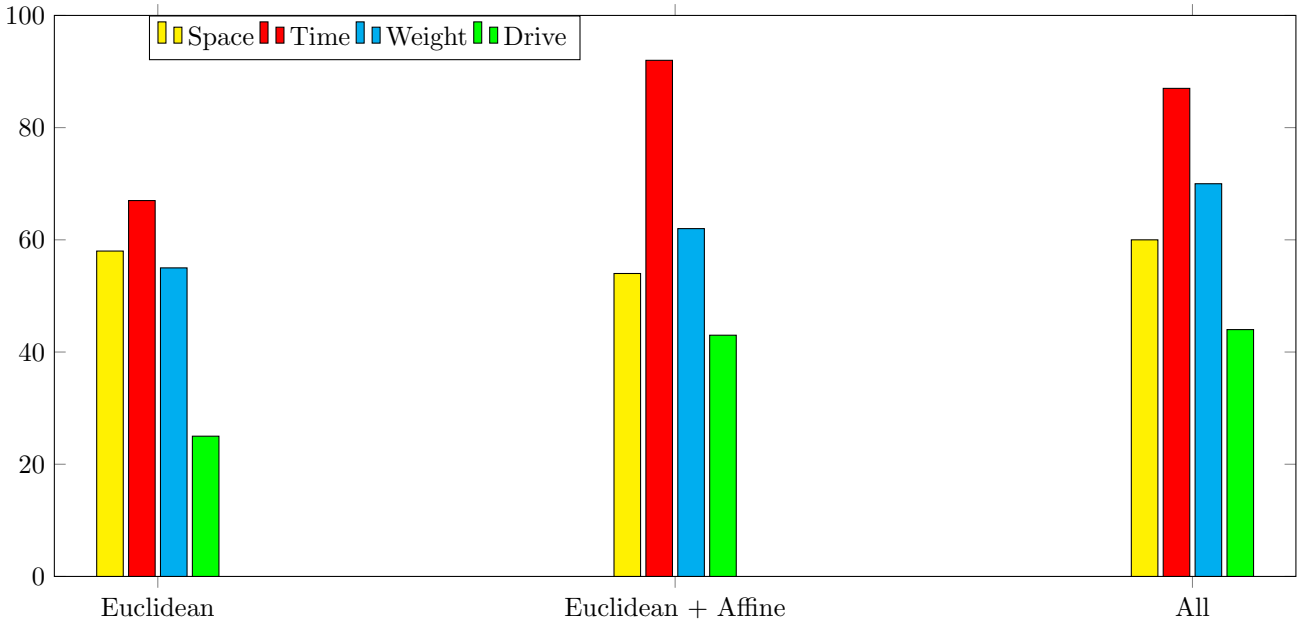


Figure 4.26: Drive HMM classification results on individual Efforts and on Drives.

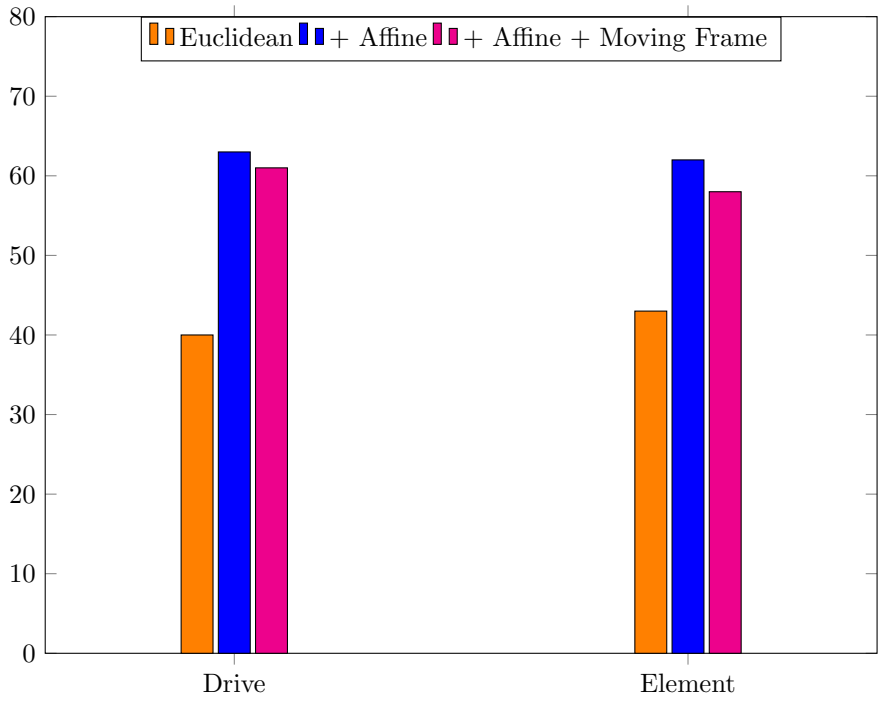


Figure 4.27: Laban State classification accuracy in percentage using the Drive HMM classifier and the Element HMM classifier.

4.4.3 Classifiers Comparison and Discussion

To properly evaluate the relative classification performance between the Naive Bayes and the Drive HMM based method, we conducted another classification study using expressive SMDs as test examples and restraining the recognition space to the Time and Weight Factors. The test set is composed of SMD corresponding to 6 different actions of our library (Jump, Flip ,Punch , Stomp, Crescent Kick, Raise One Arm) which were recorded in 4 different Time and Weight combinations, totaling 24 test examples. In addition, we also retrained our Naive Based classifier using the same training set as the HMM-based classifier allowing us to make proper comparisons between the two methods. Figure 4.28 presents the classification results with and without feature selection on a similar testing set than the one used for testing or HMM-based classifier.

Although we studied the Space Effort in our two classification methods, we cannot study it on actual SMDs as changing action trajectory from Direct to Indirect and vice versa is in conflict with our action recognition algorithm. More precisely, varying an action trajectory in the Space Effort changes the output motion token sequence perturbing the regular expression matching process. Thus, we decided to restrict the classification rate comparison to the Weight and Time Factors and study the States classification rates.

We evaluated the Naive Bayes classifier for which feature selection has been

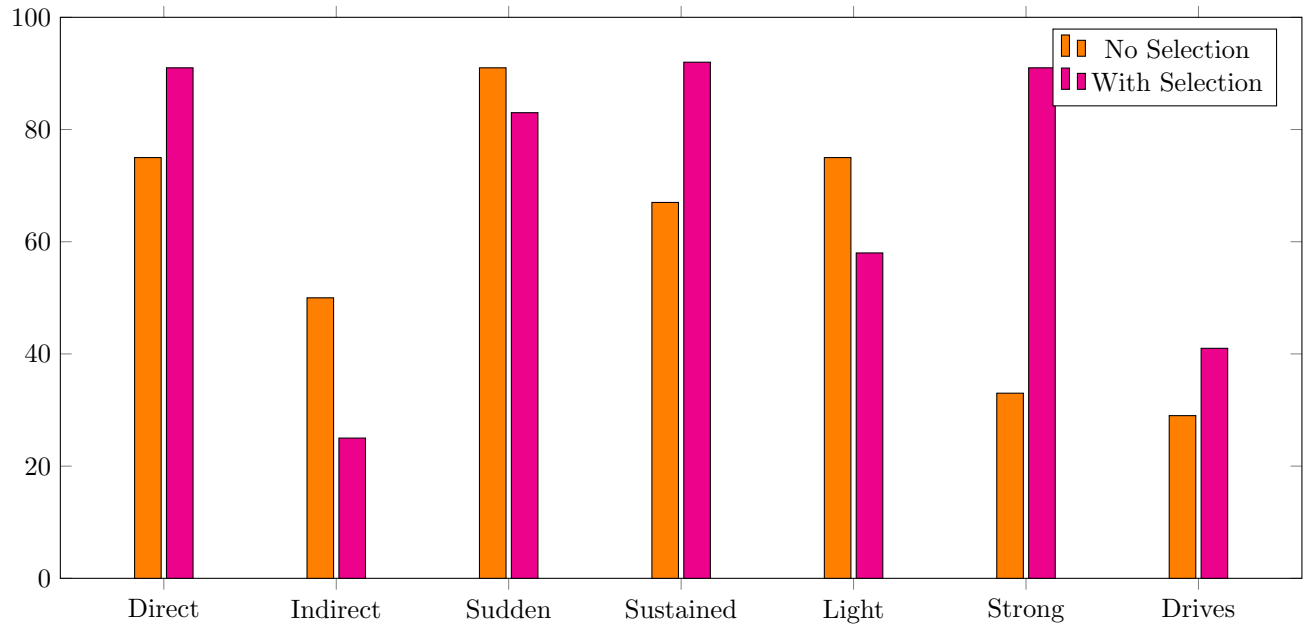


Figure 4.28: Naive Bayes Laban Effort classification accuracy in percentage before and after feature selection, using the same dataset than the HMM classifier.

applied against the Drive HMM classifier trained with all features. We obtained 35% of successful States classification for the Naive Bayes classifier against 43% for the HMM-based classifier. We also depicted the classification rates of the Time and Weight Factors in both methods. Those results show the efficiency of using HMMs against a simple Naive Bayes classifier and especially highlight that the HMM-based classifier performs better at classifying Time.

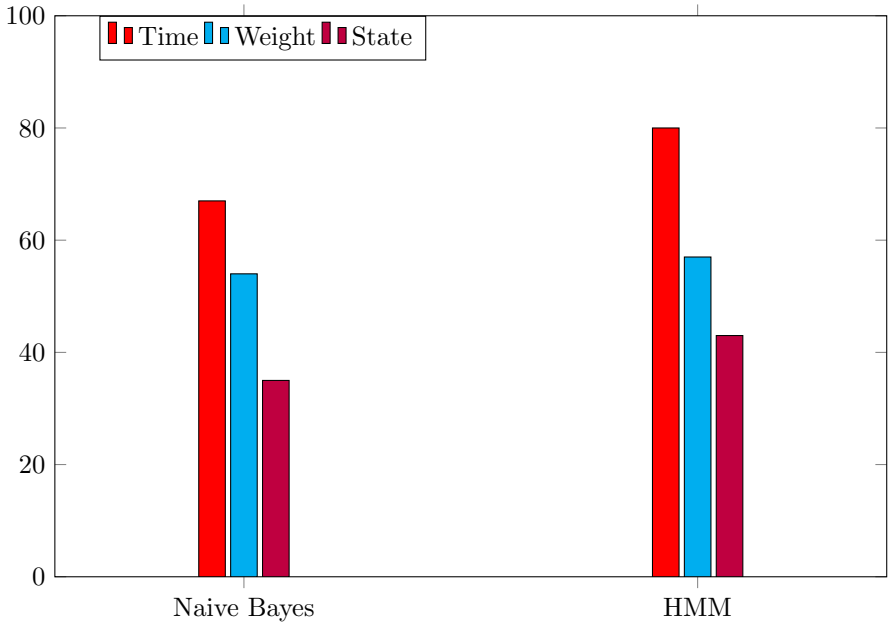


Figure 4.29: Comparison between Naive Bayes and HMM classifiers.

4.4.4 Laban Classification Conclusion and Discussion

We presented two different methods for classifying Laban Effort from hand motion trajectory. As a first experiment we trained a Naive Bayes classifier and further enhanced its classification results through a feature selection process which validated our choice of geometrical feature to analyze. In a second experiment we trained two HMM-based classifiers, one performing independent Laban Element classification and combining the result to classify Drives and one directly classifying Drives. We observed that the per Drive HMM classification method out performs the Element one when classifying Drives. On the contrary, when classifying States (omitting the Space Factor) the two methods present highly similar results. As we will only classify input SMD in terms of Time and Weight Factors, using one method or the other has no effect on how well we retrieve users hand motion qualities. On the other hand, we observed that the per Drive HMM classification method out performs the Naive Bayes method both in the Drive and State cases.

However, we also notice that classifying States and Drives is still difficult with the data we extracted from the input trajectories. This raises the question of the limit we can reach using our feature sets and if additional data is necessary to perform hand motion Laban Effort classification.

4.5 Conclusion

In this chapter, we described how we analyzed user performances, that is to say their hand motions trajectories, to extract either one or two action sequences and Laban Effort qualities on each action curve segment. The methods we introduced were tested through diverse experiments which provided objectives accuracy quantification and showed the efficiency of our recognition algorithm.

During our experiments we observed that the action and motion quality recognition results we presented are still valid when using both hands especially when performing simultaneously the same action with the same quality and when alternating actions and qualities between hands. On the other hand, it is more difficult to perform different action and/or motion qualities at the same time as it necessitate more efforts and skills. In chapter 6, we further detail how we extend our action detection system to recognize character interactions with both hands, adding a contextual analysis to our compiler. We also present a way to remove recognition confusion between highly similar actions.

Those results also highlight that while our action recognition and action

learning algorithms perform well, Laban Effort classification using 6D trajectories as input is still difficult. Moreover, it must be further explored before concluding that geometric features alone cannot be used to correctly identify Laban States and Drives. Furthermore as we cannot classify SMDs in terms of Direct and Indirect qualities using our classification methods, it would be useful to test if this partitioning can be done in practice using two different regular expressions for each action gesture, one Direct and one Indirect version.

In addition, it would be interesting to investigate the use of our action recognition system as an interface tool for other application types using devices like the Slate 2. Moreover, our system could be used to trigger simple edition actions such as redo, undo, open file, save, etc... while manipulating objects or drawing in 3D. In these use cases, we could extend our system such that it can recognize looping patterns starting from any part of the loop; one typical example might be a circle pattern which can be started from the top, the bottom or any parts. Inferring this looping property to our recognition system can be achieved by computing $n - 1$ regular expressions from a reference one, containing n tokens, which are shifted version of this expression and assign the union of these n expressions as the regular expression of the looping action.

In the next chapters, we will detail how we use those extracted information to create expressive animation sequences which follows the user intents.

Chapter 5

Performance Based Animation

In chapter 4, we broke down an input SMD into action segments with motion qualities. This segmentation gives to our system information about which animations to play at a given time intervals and their related motion quality. The next step is to compute a plausible animation sequence from this information. More particularly, three challenges remains to generate an adequate animation sequence: compute the trajectory followed by the character, correctly play and link animations with respect to the transferred trajectory and stylize animations with respect to the detected motion qualities.

We first present the context in which we are able to correctly create animation sequences (section 5.1) before describing how we transfer an input SMD into an adapted trajectory that the character will follow (section 5.2). Then, we detail how we transfer animations from our database to compose an animation sequence with smooth transitions and adapted to the transferred trajectory (section 5.3). Finally, in section 5.4, we describe how we stylize animations with respect to Laban Time and Weight Effort. Animation sequence results are then presented and discussed in section 5.5.

5.1 Animation Context

Our animation transfer algorithm operates under several assumptions which are not constraining. We first suppose that the gravity $\vec{g}$ is vertically oriented and points downward. We also restrain characters to move and jump on solid surface prohibiting underwater actions. Similarly, our system does not handle well actions like skating and sliding, therefore we do not allow such actions to be performed. In this chapter, we describe the performance transfer in the single character animation scenario, the multi-character case is treated in chapter 6.

We built a library of performable actions using the Mixamo motion capture database [1] and assigned a motion token regular expression to each of them

[1] Mixamo database: `https://www.mixamo.com/`

through learning. Currently, we store 28 animations inside our library, taken from the Mixamo database. Adding new actions is an easy task as users only have to download new animations and assign a regular expression to each of them, which can be learned using gesture examples. Note that action regular expression overlapping can occurs while the number of actions grows. As we explained in chapter 4, this is automatically detected by our system which notifies users that they have to choose another pattern for the actions they are adding to the library.

Table 7.1 refers all animations of the library with their corresponding regular expressions. This library contains diverse type of actions through walks, jumps and acrobatics, combat moves and dramaturgic gestures. Actions were progressively added while testing our system in different contexts and most of them were added and trained for recognition by different users.

5.2 Trajectory transfer

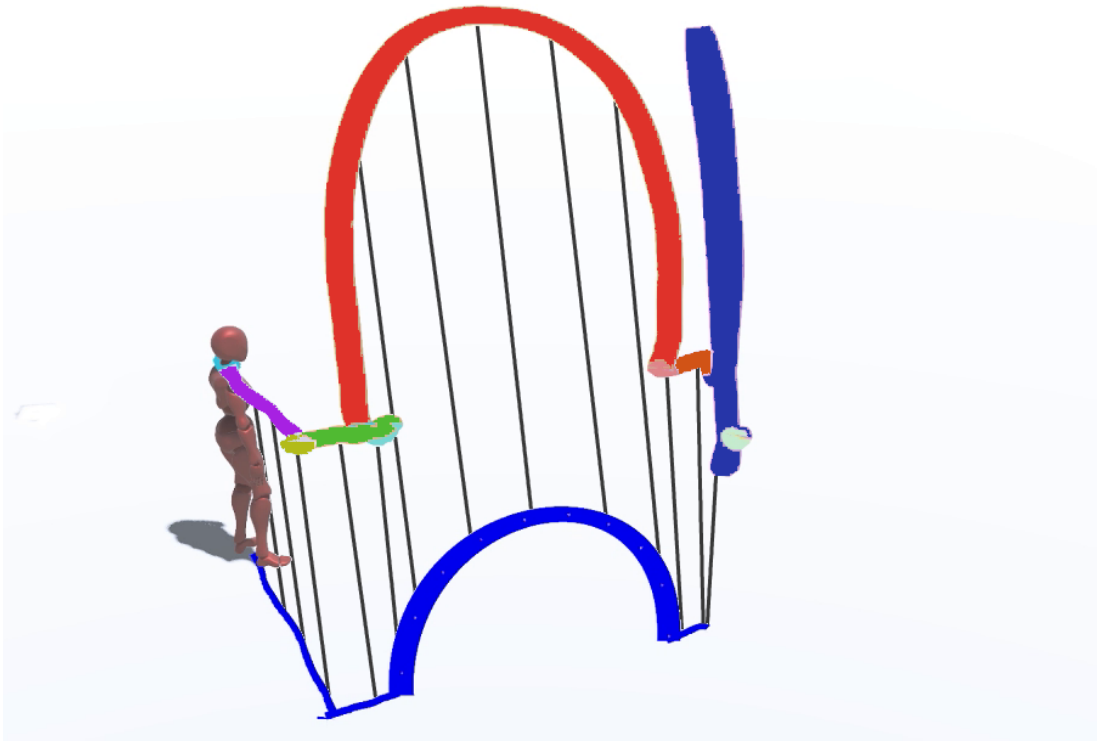


Figure 5.1: Trajectory transfer on our main SMD example. The doodle is first projected into the ground. Then Jumps action trajectories are beautified and in-place action trajectories are merged into their starting point while cyclic and not in-place actions trajectories are left as they are.

Transforming the sketched doodle into a plausible trajectory for the 3D character is not as straightforward as it appears. First, the input doodle is not drawn on the virtual ground the character lies on. Secondly, some parts of the doodle are only used for recognition (in-place actions) and should not be followed by

the character. Consequently, we must process the input doodle so that the character follows a plausible trajectory while keeping the user's intent as much as possible. This process is done sequentially for each action curve segment. Let us describe, step by step, how we transform one action doodle segment into the final trajectory segment the character will follow.

The initial action curve segment is firstly projected into the ground so that the character does not perform the corresponding animation under the terrain. If this action is a jump-like motion we restore the jump trajectory by computing a beautified parabolic trajectory using the non projected curve as reference. In this case the trajectory is also retimed so that the character starts elevating from the ground and falls back at plausible times with respect to the corresponding animation. Furthermore, if the action must be performed in-place, we remove all the corresponding action doodle from the final trajectory so that the character's root stays in place during the corresponding animation. Finally, the transferred curve might be slightly offset as a compensation of the previous step. We now detail each of this processing steps.

Projecting the curve segment. The initial curve segment projection is performed using ray-triangle intersection with the ground mesh: for each point p_i of the action curve trajectory, we cast a ray $r(t) = p_i + t\frac{\vec{g}}{\|g\|}$, $\vec{g}$ being the gravity vector (vertically oriented in our case). We then compute the intersection with all triangles of the ground mesh using Moller algorithm [Möller and Trumbore, 1997] and keep the closest one as the projection point.

Restoring Jump trajectory. After applying the projection, we removed the Jumping trajectory. This is actually intended so that we can compute a beautification of this curve. Indeed while casual Jumps actions may have corresponding gestures (for recognition) that are suitable to be followed by the character, it is probably not the case for actions like Flip or Back Flip (which may contain loops in the gesture). We thus proceed to a beautification process using an Hermite interpolation between the first and last points s_i, e_i of the projected jump trajectory. The choice of the tangent needed for the interpolation can be discussed as it depends on the user intent. In our case, we choose $T_{s_i} = -\alpha \|e_i - s_i\| \frac{\vec{g}}{\|g\|}$ and $T_{e_i} = \alpha \|e_i - s_i\| \frac{\vec{g}}{\|g\|}$ as the start en end tangents (with $\alpha = 2.5$), having opposite directions but are both in the same axis than $\vec{g}$. One may also choose α by computing the difference in height between the starting point and the apex of the original curve (before projection) h_i and impose that $dot(\mathcal{H}(s_i, e_i, T_{s_i}, T_{e_i}, 0.5) - s_i, -\frac{\vec{g}}{\|g\|}) = h_i$, where $\mathcal{H}(a, b, c, d, w)$ is the Hermite interpolation between a and b with c and d as corresponding tangents and computed at weight $w \in [0 : 1]$. With the condition that $T_{e_i} = -T_{s_i}$

this constraint guarantees that the apex of the beautified curves preserves its original height with respect to starting point.

Retiming Jump trajectory. If not further processed, the jump trajectory will induce a non plausible motion during the corresponding animation. Indeed, the jump travel has to be executed during the execution phase of the corresponding animation otherwise the character will prepare for the jump and recover from it above the ground. Thus this trajectory is also re-timed so that the character's root node moves and follows the trajectory during execution only (staying in place during preparation, anticipation, follow-through and transition phases). As an example, for a Jump animation J_i which lasts as much as its corresponding trajectory $\{P_{ij}\}$, ensuring that the character stays in place during the anticipation and follow through phases (ending respectively at t_{a_i} and t_{f_i}), our system applies the following process: $\forall p \in \{P_{ij}\}, t_p \leq t_{a_i} \Rightarrow p = p_{i0}$ and $\forall p \in \{P_{ij}\}, t_p \geq t_{f_i} \Rightarrow p = p_{in}$

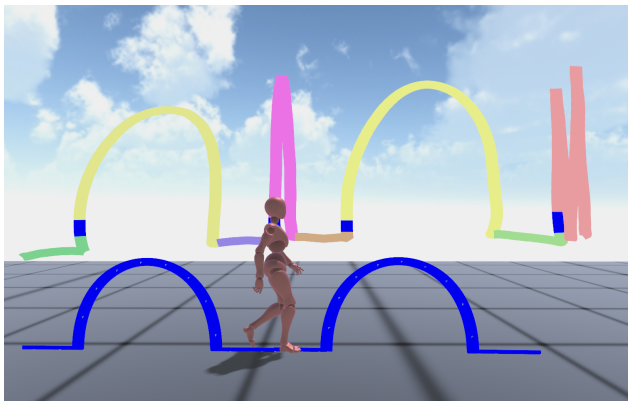
Remove action segments and apply offset. We also need to remove unused parts of the trajectory (such as vertical displacements) for actions that are to be performed in place. To do so, during the whole transfer, our system applies and increments a spatial offset s_{off} which account for the parts of the transferred doodle that are removed for in-place actions. More precisely, for an action A_i and its corresponding trajectory $\{P_{ij}\}$, we apply the spatial offset to each $p \in \{P_{ij}\}$. Then, if the current action is an in-place action, our system increments the spatial offset s_{off} by $\|p_{in} - p_{i0}\|$ and merge all the points of its trajectory into its first point.

The full transferred trajectory results in an animation layer giving the translation, rotation and scale (which can be tweaked by the user depending on the model used) of the character's skeleton's root. In figure 5.2 we show different examples of transferred trajectories on different terrains.

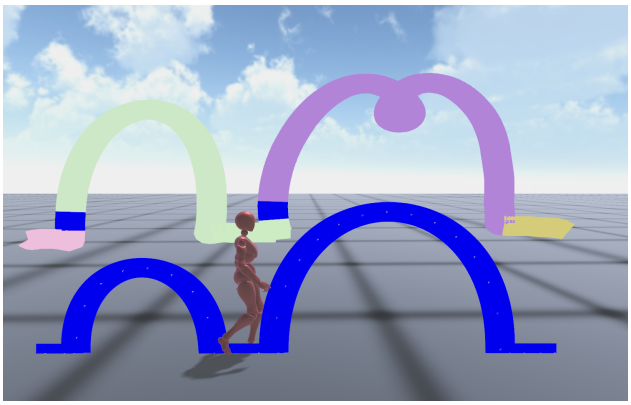
Enforce user doodle positioning. While our current trajectory transfer algorithm applies a spatial offset to account for unused parts of the user initial doodle, which ensures that the first and last points of in-place actions coincide, we can discuss whether this process respects the user's intent or not. Indeed, the transferred character's trajectory is not guaranteed to go through or stop at specific locations the user might want to. Enforcing this constraint must be done by adding additional travel animations, after the corresponding in-place action occurred, to reach the trajectory's end point instead of making it coincide with the first point. To do so, we can use the algorithm described in chapter 6 generating travel trajectories to reach specific spatial goals. However,

adding an new travel animation either shortens the duration of the related in-place action or extends the overall duration of the input doodle segment which might be undesirable in case of synchronized multiple character manipulation (see chapter 6).

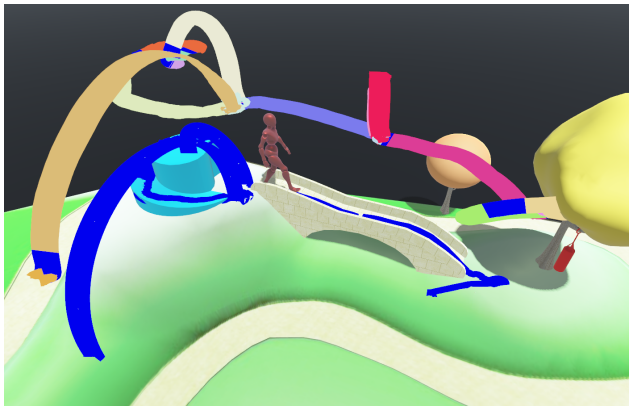
In all cases, we might let the user choose his preferences regarding that aspect of the transfer.



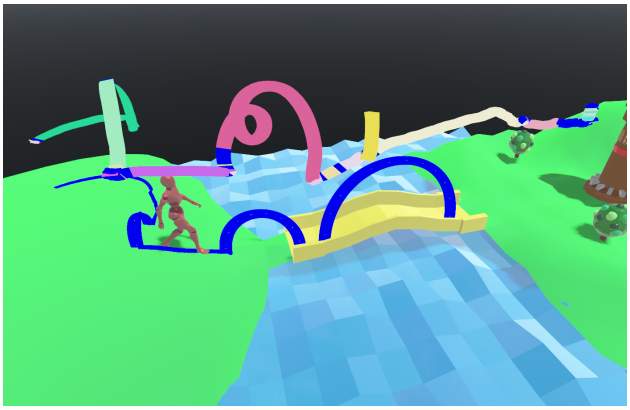
(a) Trajectory transfer on a plane.



(b) Trajectory transfer on a plane.



(c) Trajectory transfer on a relief surface.



(d) Trajectory transfer on a relief surface.

Figure 5.2: Transferred trajectories. The upper doodle represents the user hand motion segmented in color action segments while the bottom curve is the projected and adapted trajectory which is followed by the character.

5.3 Animation sequence transfer

While computing the trajectory the character must follow, our system also builds a sequence of animation clips that will be played during each corresponding action curve segment. Each animation clip corresponding to a detected action is extracted from our library and modified to an expressive animation using the desired Laban qualities (as detailed in Section 5.4). It is then re-timed so that it lasts as long as its corresponding segment of the character's trajectory. Mathematically speaking, animation duration retiming is done the following way: let's consider T_A the duration of an animation clip A and T_P the duration

of the related trajectory segment. For each keyframe Kf_i of A, we compute its new time stamp $t_i = t_0 + \frac{T_P}{T_A}(t_i - t_0)$

Beside re-timing animations, we treat cyclic actions such as Walk as a special case, in order to prevent artifacts such as foot skating: Each cyclic animation trajectory C_i is subdivided into m cycles $\{P_{ijk}\}_{k=1...m}$ where $\|p_{ink} - p_{i0k}\| = C_{length}$, with C_{length}, the traveled distance during one animation cycle. It is worth noticing that the last cycle is most of time not complete (which is only the case when $\|p_{inm} - p_{i01}\| \bmod C_{length} = 0$), thus its corresponding transferred animation must be treated accordingly. Consequently, we proceed to a motion scaling using the operator we introduce in section 5.4.2. As the travel distance of the last cycle equal to αC_{length} with $\alpha = \frac{\|p_{inm} - p_{i0m}\|}{C_{length}}$, we apply a downscale of factor α to the m^{th} transferred animation of the cycle. Note that we also apply this process on cyclic and in-place actions such as Idle, preventing the last animation to feel rushed.

The overall result of these two processes is a sequence $(A_0, ..., A_n)$ of expressive and retimed animations. We then need to compute the different time stamps to make smooth transition between animation A_i and A_{i+1} for all i. This is done using well established motion graph techniques supported by the decomposition of animation into 5 five phases.

.

In section 5.3.1 we describe a method to find transition points between two animation, and in section 5.3.2 we discuss algorithms to automatically segment an animation into its 5 animation stages (see section 5.3.2).

Locomotion Planning and Foot Skate Removal

We presented a method for "correctly" transferring cyclic animations which follow their related trajectory. While this method is generic, in the sense that it could be applied to any cyclic actions like walking or swimming for instance, it does not cope with undesirable artifacts such as foot skating. Indeed while our method works well in sections in which the character moves at uniform speed, sliding artifacts can appear during high speed variations sections.

Several existing methods deal with this issue in different contexts. One notable approach is the work of [Holden et al., 2017] which produces environment adapted locomotion animations such as walking, running or jumping while being responsive to user inputs such as joystick directions. Moreover, their neural network outputs appropriated foot animations with respect to the character current state, the locomotion phase and the user inputs. As we do not have

access to a large database of walking animations, we must rely on other type of locomotion planning method to remove artifacts.

In that aspect [Van De Panne, 1997] proposed a method to compute physically plausible step positions given the character morphology and an input trajectory. [Sreenivasa et al., 2009] proposed an online algorithm computing robot foot steps by controlling its head and checking whenever it is going to loose balance verifying if the head position is inside a control polygon at each frame. While these two methods can be adapted to address our problem, [Agrawal and van de Panne, 2016] introduced a task based locomotion planning algorithm closer to our animation sequencing and is more suitable. Moreover, they infer task adapted motion transitions which greatly enhance the credibility of the output animations, making characters behaving in a more human-like fashion. As our animation database is quite diverse in terms of action types, having adapted transitions and proper locomotion planning between two very different actions is highly desirable, especially as creating these additional steps phases might be a difficult task to achieve using our current system.

To conclude, the method proposed by [Agrawal and van de Panne, 2016] seems to be the best solution to remove adequately all foot skating from the generated animations. Due to lack of time, this solution has not been implemented in our current framework but remains a very useful feature to integrate.

5.3.1 Ensuring Correct Animation Transitions

Computing an automatic smooth transition between two animations has been well studied, especially with the introduction of motion graphs [Kovar et al., 2002] and all techniques extending and improving them [McCann and Pollard, 2007, Heck and Gleicher, 2007, Safonova and Hodgins, 2007, Furukawa et al., 2014]. The main idea behind these techniques is to find transition points between a starting animation S and a target animation T by computing distance between their frames $\{K_{s_i}\}$ and $\{K_{t_i}\}$. Moreover, in our case, $\forall i, j$ we compute $D(K_{s_i}, K_{t_j})$ and store the value in a 2D array. Based on previous work, we can emphasize that the transition points we are looking for are among the local minima of this array. However, taking the global minimum is not bound the be the best option as we might make undesirable transitions such as connecting the beginning of S to the end of T. In our case, we make use of the provided (or computed) animation stages information and choose the local minima at position (i, j) whose corresponding frame times (t_i, t_j) are nearer to some (t_s, t_t) time pair. This pair is computed using one of the following policies, depending on the number of available animation stages:

- if T possesses an preparation stage ending at p_t : $t_t = \frac{p_t}{2}$
- if T possesses only an anticipation stage ending at a_t: $t_t = \frac{a_t}{2}$
- if T does not possess an anticipation stage then $t_t = K_{t_0}.t$, which correspond to the beginning of T.
- if S possesses an transition stage ending at t_s (and thus automatically possesses a follow-through stage ending at f_s): $t_s = \frac{t_s+f_s}{2}$
- if S possesses only a follow-through stage ending at f_s: $t_s = f_s$
- if S does not possess a follow-through stage then $t_s = K_{s_n}.t$, which correspond to the end of S.

This policy set choice ensure that animation transitions do not erase the execution part of animations in the sequence while selecting adequate transition timings.

Following these policies, the only remaining choice lies in the frame distance function. In our case we chose a function similar to the one proposed by [Kovar et al., 2002]. For each frame K_{s_i} (respectively K_{t_j}) and for each skeleton joint j_k we compute a feature vector $v_{s_{ik}} = (p_i^k, \dot{p}_i^k, \ddot{p}_i^k, \dddot{p}_i^k)$ (respectively $v_{t_{jk}}$) containing the position, speed, acceleration and jerk of the joint at frame K_{s_i}. We then build the vector V_{s_i} (respectively V_{t_j}) concatenating all $v_{s_{ik}} \forall k$ and compute the Euclidean distance between V_{s_i} and V_{t_j}.

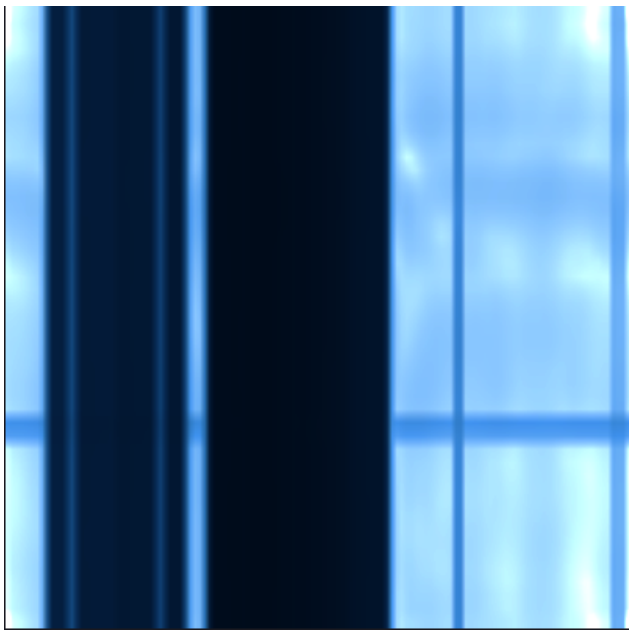
On figure 5.3, we show distance maps between animations using the distance function we described above. We can notice that our distance function is quite appropriated as it clearly highlight animation similarity like Kick and Punch and Jump and Flip whose respective distance maps have large light blue regions which represents small distances between frames.

5.3.2 Extracting Animation Stages

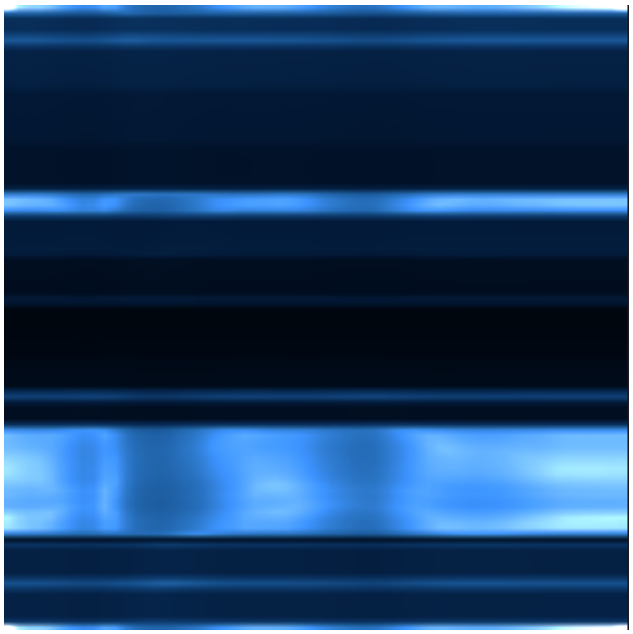
Decomposing an animation into its different stages is a critical step in our process as both transition computation (section 5.3.1) and expressiveness transfer (section 5.4) rely on this segmentation. While this stage information can be provided by an animator, this task can be tedious and is not within the grasp of non experts. In this section, we discuss several methods, including our own, to compute animation stages automatically.

One key for segmenting an animation clip, representing an action, is the fact that the end of each animation stage is highlighted by an extreme pose where speed and acceleration both vanish [Bishko, 2014]. Following this observation with propose an heuristic approach to automatically extract animation stages.

(a) Kick/Stomp animation distance map.



(b) Kick/Punch animation distance map.



(c) Flip/Jump animation distance map.

(d) Jump/Stomp animation distance map.

Figure 5.3: Animations distance maps. Each point (i, j) represents the distance between the i^{th} frame of the source animation and the j^{th} frame of the target animation. The whiter the point is the less is the distance. Transition points are searched among the local minima of the distance map. For (a) and (d) the best transition occurs between the end of the starting animation and the beginning of the target animation. For (b) and (c) best transitions are found before the end of the starting animation and after the beginning of the target animation.

We look for simultaneous speed and acceleration local minima where the speed is below a threshold s as we observed that the motion speed is not necessary null at extreme poses. This threshold s is equal to $(\min speed) * (1.0 + p)$, where p is provided by the user. We empirically found that 0.15 is an appropriate value for our animation set.

This method enables us to segment animations of our database into different stages. Then, depending on the number of segments, we either use manual or

automatic labeling, based on the fact that the five motion stages presented in Figure 2.2 always take place in the same order.

In practice, some animation stages may not be present for some actions: we solve situations where only two extreme poses are detected between the first and last key-frames by selecting either (anticipation, execution) or (execution, follow-through) for these extreme poses, depending on whether the main direction of motion changes or not between the animation segments ending at these extreme poses. Indeed, anticipation is an energy accumulation stage that makes the whole body move in the opposite direction compared to the execution of the action; in contrast, there is no major change of direction between the execution and the follow-through stages.

The method we just presented, while being simple and fast to compute, might be criticized as we do not guarantee to recover all animation stages present in the animation (especially because of the p factor) and might also find more than five stages in cases where it should not.

On the other hand, our segmentation process can also be seen as a keyframe reduction problem for which several solutions have already been proposed. One common method for tackling this issue is the Ramer-Douglas-Peucker algorithm [Ramer, 1972] which was introduced to simplify curves by recursively splitting the input curve into two parts using the farthest point from the line linking the start to the end of the curve. This process stops when an accuracy criterion is reached. Keyframe simplification methods can apply this algorithm to the joint position, rotation and scale curves independently or to compressed keyframe information such as principal components [Miura et al., 2014]. The first has the advantage of guaranteeing a lower trajectory loss but is likely to keep many keyframes as it processes each animation curve independently. On the other hand, the second method preserve much less keyframes at the expense of accuracy. Choosing between those two techniques entirely depends on the needs but the second method better address the problem of keyframe reduction. More recently [Roberts et al., 2019] introduced a dynamic programming based approach which gives an optimal keyframe reduction depending on K, the number of keyframes to keep, and a distance function between two high dimension curves.

Using such as method with K varying between 1 and 5 might be a better solution to our stage segmentation problem. Comparing this solution to our heuristic animation stage decomposition method is left to future work.

5.3.3 Gesture-Based Animation Retiming

As a first approach of motion quality transfer, we studied how animation retiming with respect to user gesture performed in terms of intent transcription. In order to synchronize an animation clip with the user gesture we proceed to an arc length reparametrization of the animation timing. We first assume that the animation clip last as long as the input trajectory with a duration of T, which is the case when creating the animation sequence from an SMD. Then, for an input trajectory $\{P_i = (x_i, y_i, z_i, \theta_i, \phi_i, \gamma_i, t_i)\}$ we compute the related arc length of each point p_i using the following formula:

$$l_i = \sum_{j=0}^{i-1} \|(x_{j+1}, y_{j+1}, z_{j+1}) - (x_j, y_j, z_j)\|$$

and build a mapping function β such that $\beta(t_i) = l_i$. The total arc length of the trajectory L is equal to $\beta(t_n)$. Finally, we define the gesture synchronized retiming function:

$$\Phi(t) = T\frac{\beta(t)}{L}$$

As we only know β for a finite number of points, if $t \in [t_{i-1} : t_i]$ we can compute $\beta(t)$ as a linear interpolation $(1-w)\beta(t_{i-1}) + w\beta(t_i)$ with $w = \frac{t-t_{i_1}}{t_i - t_{i-1}}$. In addition, the same reparametrization can be computed using the rotational information $(\theta_i, \phi_i, \gamma_i)$ which enables users to also synchronize animation clips with rotation only gestures. It is important to note that this reparametrization cannot be correctly applied to gesture combining translations and rotations because we would have to deal with two arc length of different magnitudes.

We compared the results of this simple quality transfer to the neutral and Laban modified animation and found that this approach is suitable for conveying suddenness and sustainability qualities (see section 5.4.3).

5.4 Laban qualities transfer

Like mentioned in chapter 2 animation stylization techniques are divided between procedural and data-driven methods. In this work, we devised a procedural approach for stylizing input animation into Laban Time and Weight Effort variations. This choice was supported by our wish to generalize our stylization to any kind of animation and for various character skeletal structures. In addition, it is also difficult to acquire a sufficiently exhaustive animation database for every Time and Weight variations, especially when LMA expert are more used to perform Laban Effort variations for dance gestures rather than

video game like gestures.

Our Time and Weight animation modifiers rely on three independent animation operators: a Motion Scale operator, a Retiming operator and a Reshaping operator. The Motion scale and Reshaping operators also depend on a character skeleton segmentation we established based on human motion observation. We first introduce our character skeleton partitioning before describing each operator and then detail the Time and Weight modifiers and discuss about their inter-compatibility. Finally, we show results of user studies aiming at evaluating the efficiency of our modifier in terms of quality recognition.

5.4.1 Skeleton Partitioning

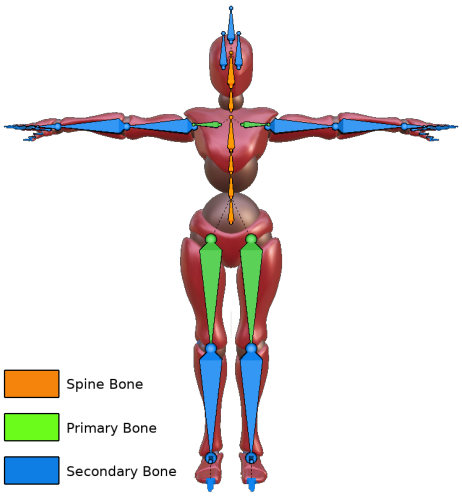


Figure 5.4: Skeleton decomposition into Spine, Primary and Secondary bones.

When human perform actions, we observe that all joints do not have the same impact on the global motions. For instance, when throwing an object the source of the motion comes from one shoulder and propagates through the throwing arm. The same observation can be made for a kick motion where the motion start from the hips and propagates through the legs. We also observe in these same motions that the spine usually twists and bends to accumulate and release energy. These observations suggest that we can classify bones with respect to how they influence motions. This classification process has been studied the context of animation retargeting [Poirier and Paquette, 2009], [Abdul-Massih et al., 2016] for which correspondences between two skeletons are established in order to properly adapt an animation of one character to another one having a different morphology.

In our case, we divide bones between Spine bones, Primary bones and Secondary Bones . Spine bones, as the name suggests, are bones part of the spine while Primary bones are direct children of spine bones, containing shoulders and hips, and Secondary bones are composed of all other unlabeled bones. The spine can be specifically annotated by users but we also provide a semi-automatic method to compute it by only providing the pelvis, chest and head bones supposing that the pelvis is in the parent hierarchy of the two others. More precisely, the spine of the skeleton is composed of the bone chains which link the pelvis to the chest and the pelvis to the head. Figure 5.4 depict our

skeleton representation for the Mixamo skeleton structure we use in most of our animated examples.

In the next sections, we show how we use this partitioning to apply our different animation operators.

5.4.2 Animation Modifier Operators

Motion Scale Operator enables us to re-scale motion in an adapted way for each of the animation stages: parts of motion may be enhanced or made less salient depending on the context the operator is applied. This is used for instance to down scale the anticipation part of Sudden and Sustained actions and enhance Strong motions. More generally, this operator re-scale bones translation and orientation from any keyframe Kf_i to an end keyframe Kf_j ,with $j > i$, by a factor c provided by the user. Applying a spatial scale to a segment of the animation clip is done as follows: given a start and end keyframes Kf_s and Kf_e of an animation stage, we compute for each bone b_i its rotation $q_i(t_s)$ and its position $p_i(t_s)$ at the start of the stage. Then we apply scaling using:

$$\begin{cases} q_i(t) & = scale(q_i(t_s)q_i(t)^{-1}, S_c(b_i))^{-1}q_i(t_s) \\ scale(q_i(t), S_c(b_i)) & = Slerp(q_{unit},\ q_i(t), S_c(b_i)) \\ p_i(t) & = p_i(t_s) + S_c(b_i)(p_i(t) - p_i(t_s)) \end{cases} \tag{5.1}$$

where $S_c(b_i)$ is a correction scale function which depends on the bone b_i and c, and q_{unit} is the unit quaternion whose angle is null. Note that c will also depend on is the Effort values L_t and L_w when using the operator with the Time and Weight Modifier. Let us detail S_c function and justify its use. We distinguish two ways of applying the motion scale operator namely partially versus fully.

When applied partially:

$$S_c(b_i) = \begin{cases} c \text{ if } b_i \text{ Spine or Primary bone} \\ 1 \text{ otherwise} \end{cases}$$

When applied fully:

$$\forall b_i, S_c(b_i) = c$$

We restrict the motion scale to Spine and Primary bones when applying the scaling partially. On the other hand, all bones motion are uniformly scaled when applying the operator fully. We justify this choice because we stated that up scaling all bones rotations often produces non convincing results like shown on figure 5.6. It is especially the case for leaf bones (which do not have any

child) which already have amplified motion from their parent, and the deeper their parent hierarchy is, the more their motion will be scaled. In contrast full scale is most useful for down-scaling animations especially when users want to completely fade out part of an animation. We can also highlight that full scale usage is mandatory for motions where articulation and spine bones are not predominant. Figure 5.6 shows comparisons of partial and full motion scaling mode highlighting "failure" cases for both of them.

At end of the motion scaling process, cohesion with all the neighboring keyframes after e must be maintained and contacts with the ground must also be preserved at the concerned keyframes. To do so, for each keyframe Kf_i with $t_i > t_e$ cohesion is maintained using the following formula for each bone b_j:

$$\begin{cases} q_j(t_i) & = scale(q_j(t_e)_{old}^{-1} q_j(t_e)_{new}, 1 - \frac{t_i - t_e}{t_n - t_e}) q_j(t_i) \\ p_j(t_i) & = t_j(t_i) + (p_j(t_e)_{new} - p_j(t_e)_{new})(1 - \frac{t_i - t_e}{t_n - t_e}) \end{cases} \tag{5.2}$$

Finally we perform the contact preserving process using IK on keyframes where at least one foot was in contact with the ground before scaling. More precisely, for each contact frame Kf_i^c and for each bone b_i which was in contact during this keyframe, we create an IK chain from this bone place an IK target at $p = (p_{i\,new}^x, p_{i\,old}^y, p_{i\,new}^z)$ where p_i is the global position of the bone. We then use the FABRIK IK solver [Aristidou and Lasenby, 2011] to restore the bone height back such that it stays in contact with the ground.

Retiming Operator consists in reparametrizing and modifying an animation overall timing $\Phi(t)$ which takes as input the time $t \in [start : end]$ and returns the time $t_{reparam}$ at which the animation will be computed. Moreover, this operator can extend or shorten animation phases using linear time scale. While any retiming scheme can be used for Φ it is important to choose schemes which can induce ease-in (acceleration) and ease-out (deceleration) in a coherent way between animation stages (figure 2.2).

To do so, we propose a family of recursive functions to define a new time parameterization which conserves consistency between stages and defines a wide variety of retiming schemes:

$$f_i(t) = f_{i-1}(end_{i-1}) + a_i.(t - start_i)^{b_i} \tag{5.3}$$

where i (from 1 to 5) relates to the animation phase, f_0 is the constant function returning time t_0 when the action starts, and a_i, b_i are parameters computed from the effort values to be transferred. This scheme allows users to linearly

Punch Partial up-scale.

Punch Full up-scale. Self intersections and unusual articulation breaking of the arms are predominant in this mode.

Punch Partial down-scale.

Punch Full down-scale.

Stomp Partial up-scale. One of the few case where Full scale is more appropriated than partial scale.

Stomp Full up-scale. This mode is more adapted because it amplifies the knee joint rotation when hitting the ground.

Stomp Partial down-scale.

Stomp Full down-scale.

Kick Partial up-scale.

Kick Full up-scale. Elbow articulations tend to be reversed in this mode and the leg self intersects the character.

Kick Partial down-scale.

Kick Full down-scale.

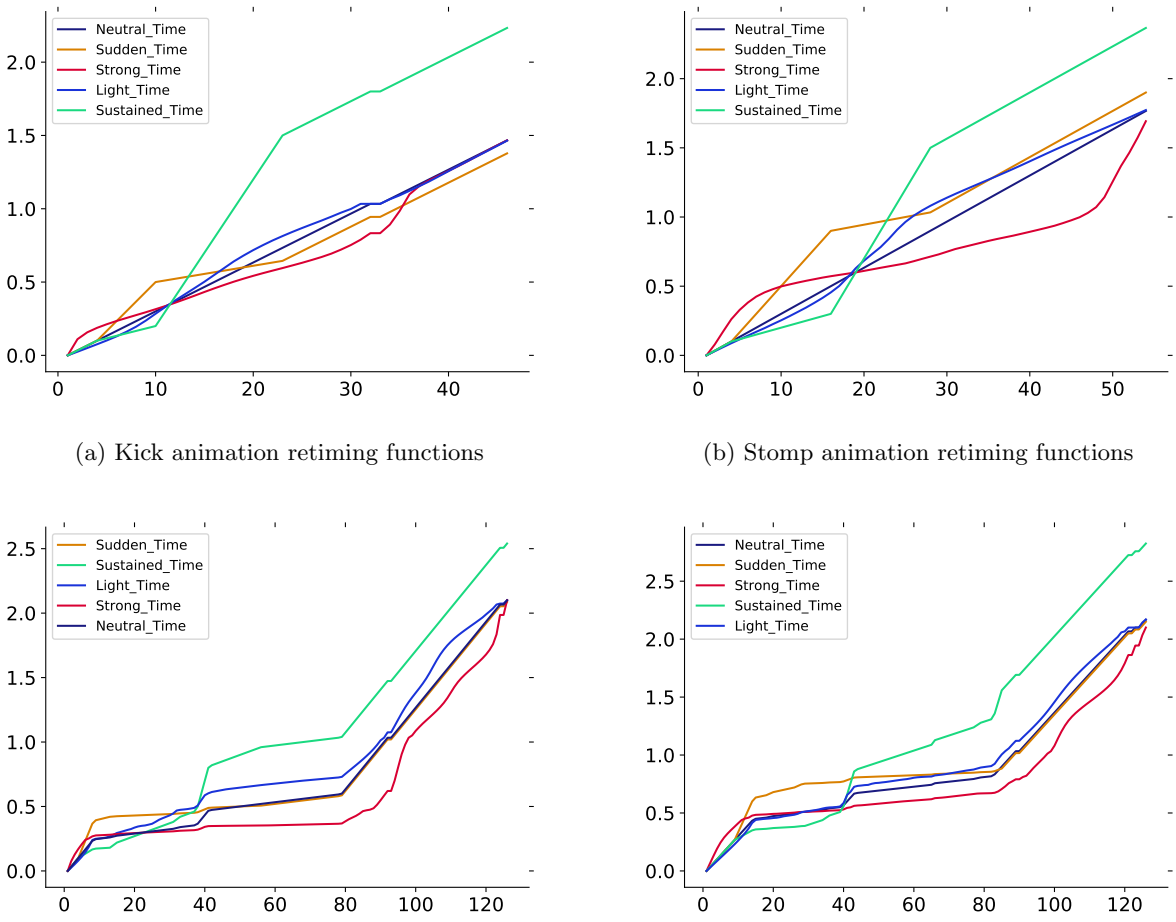
Figure 5.6: Motion-scale operator examples. Left column: Partial scale. Right column: Full scale.

shorten or extend animation phases by changing the value of a_i while controlling the time variation type by tweaking the power value b_i.

This retiming scheme is especially useful for inducing ease-in and ease-out ($b > 1$ and $b < 1$) as well for masking animation stages by taking a value near 0 for a which coincides with the way the Time modifier operates (see section 5.4.3).

In Figure 5.7 we show different retiming scheme computed for 4 animations in modified 4 different Laban qualities, using the retiming operator.

Note that the gesture-based retiming introduced in section 5.3.3 is one particular application of the retiming operator.



(a) Kick animation retiming functions

(b) Stomp animation retiming functions

(c) Punch animation retiming functions

(d) Throw animation retiming functions

Figure 5.7: Animation Retiming functions for the four different Time and Weight qualities computed using our Retiming Operator. The x-axis shows the animation frame number while the y-axis represents the animation time.

Reshaping Operator. This operator changes the overall posture of a char-

acter during the execution phase of an animation. Using the previously introduced skeleton segmentation (see figure 5.4), the reshaping operator applies a rotation shift δ_θ on spine and articulation bones. Horizontal shape feature is simulated by applying the rotation shift to spine bones relatively to their local x axis such that for a bone b_i, $q_i = q_i(\delta_\theta, 0, 0, \delta_\theta)$. In a similar manner, Vertical shape feature is simulated y applying the rotation shift to articulation bones relatively to their local z axis such that $q_i = q_i(0, 0, \delta_\theta, \delta_\theta)$. Figure 5.8 shows how our operator applies Horizontal and Vertical Shape features.

In addition, the reshaping operator enables us to visually enhance the Weight Effort by applying an adequate tuning of the body posture during motion. We do this based on the relation between Laban *Effort* and *Shape* categories, which states that a Light motion is better expressed with a Rising body posture, and a Strong motion with a Sinking posture.

(a) Sinking shape (b) Rising shape (c) Spreading shape (d) Enclosing shape

Figure 5.8: Reshaping operator. This operator applies Laban Shape features to the character changing its global posture during the execution phase with respect to the Horizontal, Vertical and Sagittal plane.

5.4.3 Laban Modifiers

In this section, we finally present our Laban animation modifiers for the Time and Weight Efforts. Those modifiers combines the animation operators we presented in section 5.4.2 to stylize neutral animations into their related Light, Strong, Sudden or Sustained version. We then discuss about Laban modifiers combination to produce Laban States animation versions and present an evaluation of our stylization process.

Time Modifier

The Time modifier, parametrized by $L_t \in [-1 : 1]$, aims to give a feeling of impulse for Sudden motions ($L_t > 0$), respectively a feeling of pose and control for sustained ones ($L_t < 0$). In both cases, this implies changing the relative duration of some of the action stages as well as their space amplitude. Moreover, preparation and anticipation phases are down-scaled, in a full scale

fashion, by a factor $c = \frac{1}{3|L_t|+1.0}$ using Equation (5.1) for both Sustained and Sudden motions. Indeed, Sudden motions feel unprecedented and should not display the intention of preparing an action while Sustained motions do not need anticipation as their execution needs less effort. Notice that when $L_t = 0$, $c = 1$, thus the animation remains unchanged.

With respect to the feature selection results we obtained in chapter 4 and following the principles of animation introduced by [Thomas and Johnston, 1981], we apply our retiming operator on both Sustained and Sudden motion making Sustained motions execution longer and anticipation shorter while sudden motions have shorter execution phases and longer anticipations. Our retiming operator also induces an important acceleration phase at the beginning of Sudden motions execution phase while inducing a deceleration for Sustained motion executions.

Therefore, we use the following implementation of Equation (5.4.2):

$$\begin{cases} f_{ant}(t + end_{prep}) = \begin{cases} f_{prep}(t_{prep}) + \frac{1}{-0.5L_t+1}t \text{ if } L_t \geq 0 \\ f_{prep}(t_{prep}) + (0.5L_t + 1)t \text{ otherwise} \end{cases} \\ f_{exec}(t + end_{ant}) = \begin{cases} f_{ant}(t_{ant}) + \frac{1}{2L_t+1}t^{-0.4L_t+1} \text{ if } L_t \geq 0 \\ f_{ant}(t_{ant}) + (-2L_t + 1)t^{-0.4L_t+1} \\ \text{otherwise} \end{cases} \end{cases}$$

where t_{prep}, t_{ant}, and t_{exec} are respectively the end times of the preparation, anticipation, execution, and follow-up stage. Note that for each stage reparametrization $t \in [0 : end_i]$.

In early experimentation we also applied a global time scaling post-process aiming at conserving the initial animation duration. However we rejected this additional step as we observed that it greatly deteriorated the expressiveness of the output animation to the point that users were not able to distinguish Suddenness from Sustainability in several animations. This was to be expected as one major factor of the Time feature is the duration of the execution phase.

Weight Modifier

When applying the Weight modifier, parametrized by $L_w \in [-1 : 1]$, we would like to make Light motions ($L_w \leq 0$) feel more uniform and more subtle against more powerful ($L_w \geq 0$) for Strong motions.

To infer those traits into a given animation, we first apply the motion scaling operator to the whole animation, enhancing Strong motions and restraining Light motions. To do so we choose ($c = \frac{1}{-L_w+1}$) ≤ 1 for light motions and ($c = L_w + 1$) ≥ 1 for Strong motion in Equation 5.1. This operator is applied

partially except for animations where neither spine and articulations bones motions are predominant and for few special cases. Moreover, this is the case of the Stomp animation as well as the Raise One Arm, Point, Dismiss and other more subtle animations.

We then re-time neutral animations in order to amplify (for Strong motions) or flatten (for Light motions) speed variations. To achieve such a retiming, we do not use the function family we introduced in section 5.4.2 as we want to infer a global timing behavior without taking into account the stage subdivision. Thus, in a very similar manner than in section 5.3.3, we proceed to an arc length reparametrization $\gamma(t)$ of the input animation while modifying its speeds.

Moreover, at each frame i we compute the arc length $\gamma(t_i) = L^i = \overline{V_e^i}^{\alpha L_w}(t_i - t_{i-1}) + L^{i-1}$ ($\alpha = 1.5$ in our case) where $\overline{V_e^i}$ is the average joint linear velocity at frame i:

$$\overline{V_e^i} = \frac{\sum_{j \in bones} \left\| (x_{i+1}^j, y_{i+1}^j, z_{i+1}^j) - (x_i^j, y_i^j, z_i^j) \right\|}{\#bones} \tag{5.4}$$

Note that for Light motions, as L_w is negative we clamp values between 0 and 1 to avoid divergence of the speed when it comes near 0. Finally, we define the Weight reparametrization function in the time domain (for $t \in [t_{i-1} : t_i]$):

$$\Phi(t) = ((1-w)\gamma(t_{i-1}) + w\gamma(t_i))\frac{T}{L} \tag{5.5}$$

with $w = \frac{t - t_{i_1}}{t_i - t_{i-1}}$ and where L is the total arc length and T the total duration of the animation. The resulting effect amplifies speed variations for strong motions and flatten them between 0 and 1 for light motions, making the computed speed more uniform. For neutral animation, $L_w = 0$, $\overline{V_e^i} = 1$ and the initial timing is maintained. We can notice that the initial duration of the animation is preserved independently of the L_w value.

Lastly, we improve Weight transfer by making use of the reshaping operator. Moreover, we apply the Horizontal feature with a factor $\delta_\theta = 0.1L_w$, depending of the *Weight* value so that Light motions use an uprising body posture while Strong motions use down body-postures.

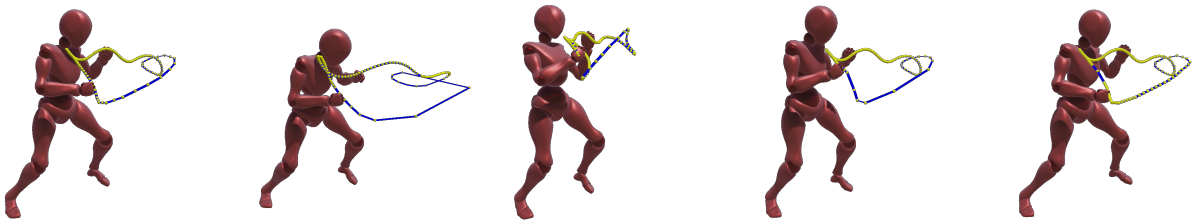
Combining modifiers

While we perform Laban Effort recognition to detect Laban States we would like to be able to faithfully transfer then using our Time and Weight modifiers. However, while our operators are invertible and commutative, it is not the case of our modifiers which can be combined but are neither commutative or invertible. This can be simply observed in the case of the Time modifier which ap-

plies downscaling during the anticipation phase both in Sudden and Sustained motions. Consequently, the neutral anticipation cannot be reconstructed by applying successively a Sudden and Sustain modifier and vice versa. In addition Time and Weight retiming scheme are not commutative because of the arc length reparametrization inferred in by the Weight operator.

Despite those restrictions, we still are able to combine Time and Weight operators by choosing in which order we apply them. After experimentation with all the animation of our database we found that applying the Time modifier before produce more plausible results especially in terms of timing. We also observed that combining Strong and Sudden as well as Light and Sustained motion often produced over exaggerated results because of the similitude of those features. In section 5.5 we show results of combining Time and Weight modifiers on SMDs animation sequences and figure 5.13 shows independent modifiers results on a subset of animations from our database with corresponding geometrical properties in figure 5.16.

Stomp Neutral Stomp Strong Stomp Light Stomp Sudden Stomp Sustained



Punch Neutral Punch Strong Punch Light Punch Sudden Punch Sustained

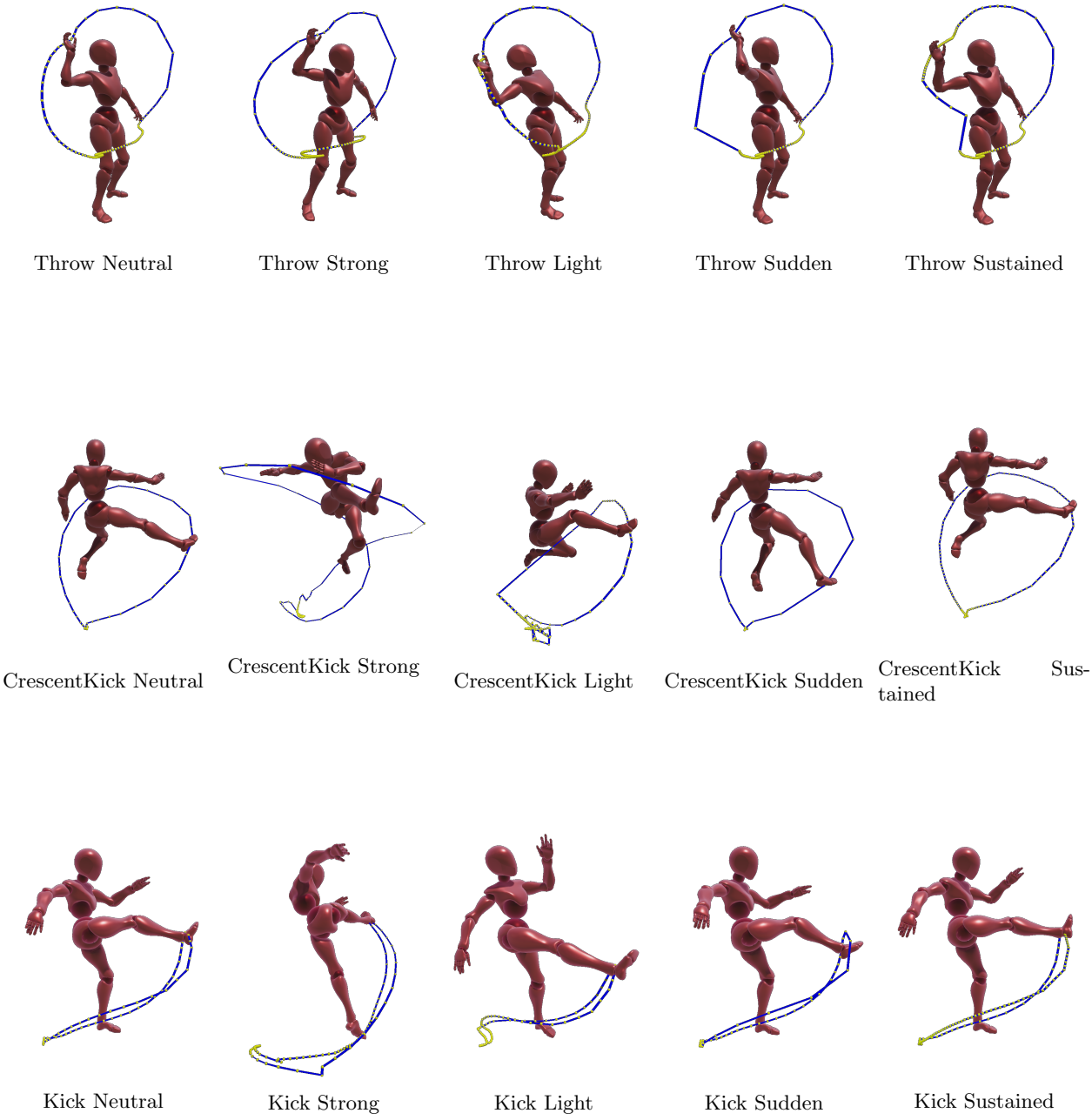


Figure 5.13: Animation Time and Weight stylization with our Laban modifiers. For each animation we show the trajectory of the most relevant end effector with yellow spheres corresponding to the bone positions at the different keyframes.

Modifiers Evaluation

We evaluated the efficiency of our Laban stylization in two different subjective experiments aiming at rating the expressivness of our modifiers and at measuring its gain with respect to a simple gesture-based retiming.

In the first experiment, we asked 8 non expert-users to label 32 animations in one of the four Laban Factor (Light, Strong, Sudden, Sustained). Those animations were generated from 8 animations from our database by applying the

Laban modifier we introduced in section 5.4.3. Results in terms of classification accuracy (percentage of correct recognition) are depicted in Figure 5.14. Those experiments demonstrate recognition rates well above chance level in all cases($\geq$ 80%). We observed that for some cases users confused strong animations as sudden and light animations as sustained which is also likely to be cause of Drives classification confusion in our recognition rates.

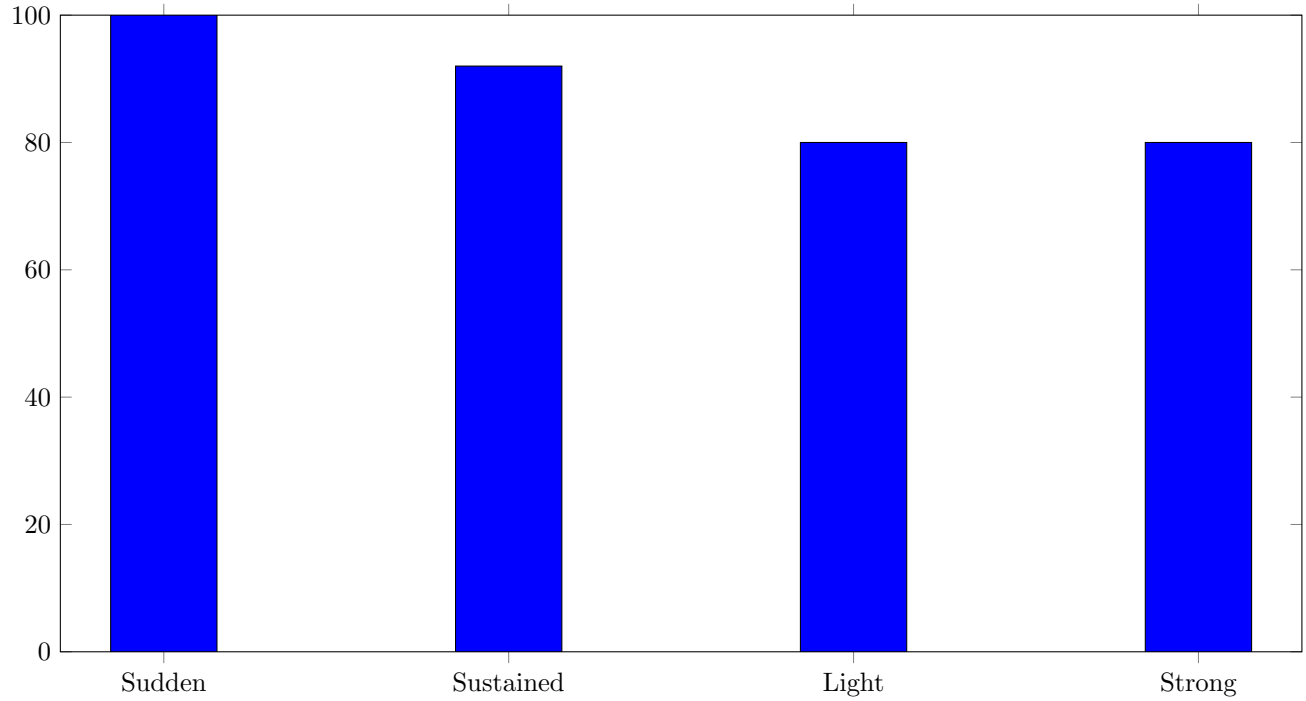


Figure 5.14: Accuracy of subjective recognition of transferred Laban effort qualities. Users were asked to recognize Laban effort qualities after viewing the transferred animations. Results are reported in percentage.

In the second experiment, we evaluated the efficiency of our Laban modifiers with respect to the simple retiming stylization we introduced in section 5.3.3. We presented twenty pairs of animations to five users in which one animation was naively stylized and the other was modified using our Laban stylization. Those animations were derived from 5 neutral animations stylized in the four different Laban Factors. It is important to note that each naively stylized animation, which are retimed with respect to a user gesture, result from expressiveness adapted SMD. Moreover, Light animations were produced from Light SMDs.

For each animation pair, users were then asked to choose the animation that best matched the intended motion quality, either choosing the naive method or our stylization method (even though they were not aware of the method used for modifying animations). Classification results are depicted in Figure 5.15. Those results highlight two interesting facts: firstly, our Time modifiers do not seem to bring more expressiveness that a user gesture retiming as classification results are similar (with a little tendency towards the gesture retiming method);

in some few case users even found that Time modified animations were too exaggerated and less plausible than the simply retimed animation. However, we can also interpret this balanced tendency as a proof that, to some extend, we successfully reproduced procedurally Sudden and Sustained timing but without gain with respect to the simple retiming method. This implies that the motion down-scaling masking the anticipation phase did not bring any additional quality in this case.

Secondly, our Weight modifier out performed the simple retiming method especially for strong motions. Users found that the additional motion scaling and reshaping operations greatly enhance the Weight quality of the animations. For more subtle animations like the Raise One Arm of our library, our Light modifier applied slight changes compared to the naively stylized animation.

To conclude, this experiment suggests that the best way of transferring Laban Effort qualities from an input action segment is to retime the output animation using gesture-based retiming and apply our reshaping and motion scaling operator with respect to the corresponding recognized Laban Time and Weight qualities.

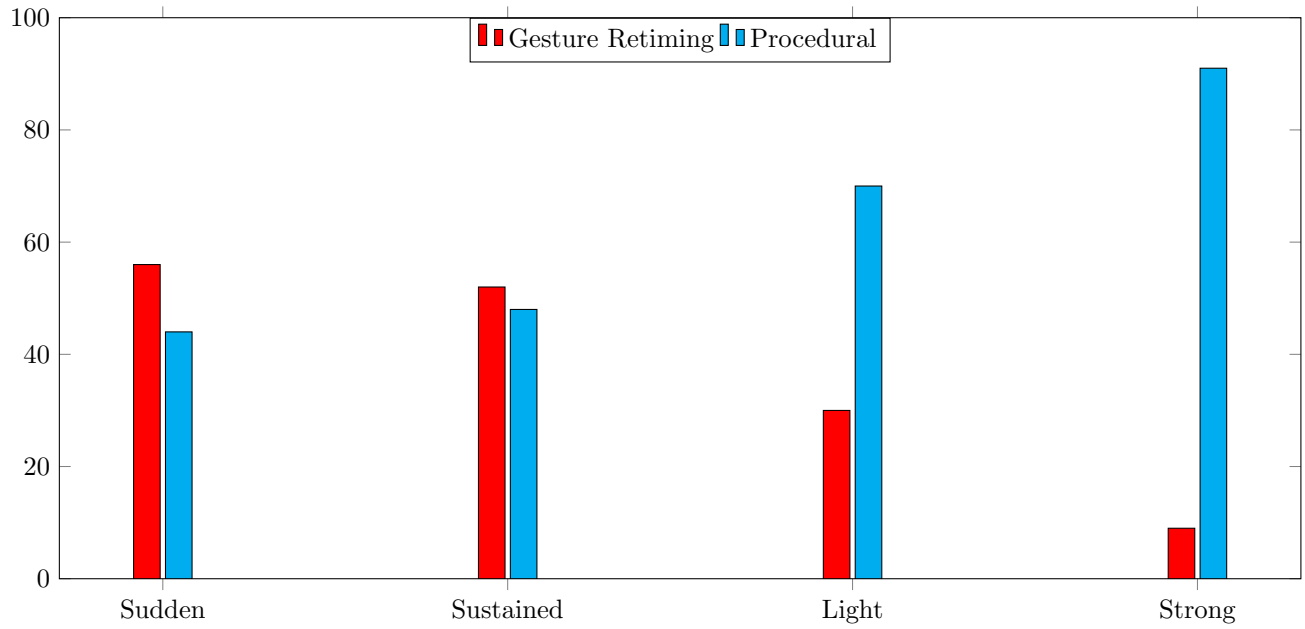


Figure 5.15: Comparison between our modifiers and the gesture-based retiming method.

5.5 SMD Expressive Transfer Results

We finally tested the ease of use and expressiveness of our animation sequence authoring method through several experiments. In a first experiment we asked 5 users to freely create 3 complex animations and asked them to subjectively evaluate the resulting animations. All users first trained our system to use their own gestures during their recordings. The training sessions lasted five minutes on average. Users then created three free-style animations, for a total running time of three minutes each on average. The animations were then presented to them in a subjective view in the HTC Vive headset. Users recognized their intended actions in most of the cases and were generally satisfied with the transferred motion styles. They found the system easy to use. In some cases, false detections occurred while users were pausing or gesturing non intentionally. Transferred animation results created by these users are shown the Figures 5.17 ,5.18, 5.19,5.20,5.21.

We also present animation results in which a user created expressive sequences combining two Laban Effort Time and Weight qualities. Moreover Figure 5.22 shows an animation example combining Sudden and Light qualities and Figure 5.23 shows a Sustained and Strong animation sequence. Figure 5.24 shows the stylized version of our main SMD example. Note that the majority of action segments' qualities were correctly recognized by our Laban classifiers, while the misses were forcefully assigned to the correct qualities.

Overall, users were satisfied with the produced results of our system both in terms of gesture recognition and animation sequencing. During this thesis we observed that allowing users to use their own gestures really made our system usable and let them fully express their creativity. However, more work is needed to correctly recognize Laban Effort as we mentioned in chapter 4. Furthermore, users also complained that the HTC Vive controllers limited their manipulation in the case of turning motions. Finally users also requested the possibility to edit their animation sequence.

5.6 Conclusion

In this chapter we presented how we create an expressive animation sequence from users' hand motion trajectories and the action and motion quality sequences extracted from the gesture analysis stage. More precisely, in the context of single character performance transfer, we showed how we transform the input curve into the actual trajectory the character will follow. We also presented how we sequence animations and ensure smooth transitions while

limiting animation artifacts such as foot skating and ground penetration. In addition, we detailed how we procedurally infer Laban Effort qualities into input animations combining three low level operators. Finally, we evaluated the usefulness of our system through different user studies and showed resulting expressive animation sequences executed by several users.

The tools and methods we presented in this chapter open up several research perspectives. First, proposing a way to infer the Space Factor qualities into input animations should be investigated. As we observed that Curvature and Torsion variations were features dissociating Direct from Indirect motions, one key of performing this stylization on full body animation could be to enforce these behaviors to the end effectors trajectories. Secondly, Even if we did not focused on recognizing the Flow Factor in this work it could be very interesting to propose a way to infer it into input animations.

More generally, having a ground truth to compare our stylized animations with would be really useful and might give more information about how to proceed when stylizing full body animation with respect to the different Laban Efforts. Acquiring such a database might require some tedious work as we would like to study Factor, States and Drives which makes 26 variations for a single animation. Indeed having all the variations will gives both information on individual Efforts but also on the consequences of combining them. As we discussed in section 5.4.3 combining Factors gives different results depending on the order we apply them, which might be undesirable. In the future, it could be very interesting to study if there is one way to guarantee the commutativity and the reversibility of the Laban modifiers.

Finally, through our experiments we observed that using Laban Effort theory as a motion quality vocabulary was not well adapted, as most users are not familiar with this formulation. Therefore, similarly to [Aristidou et al., 2017], it would be interesting to use a higher level representation such as emotions and compute an additional mapping with Laban Efforts.

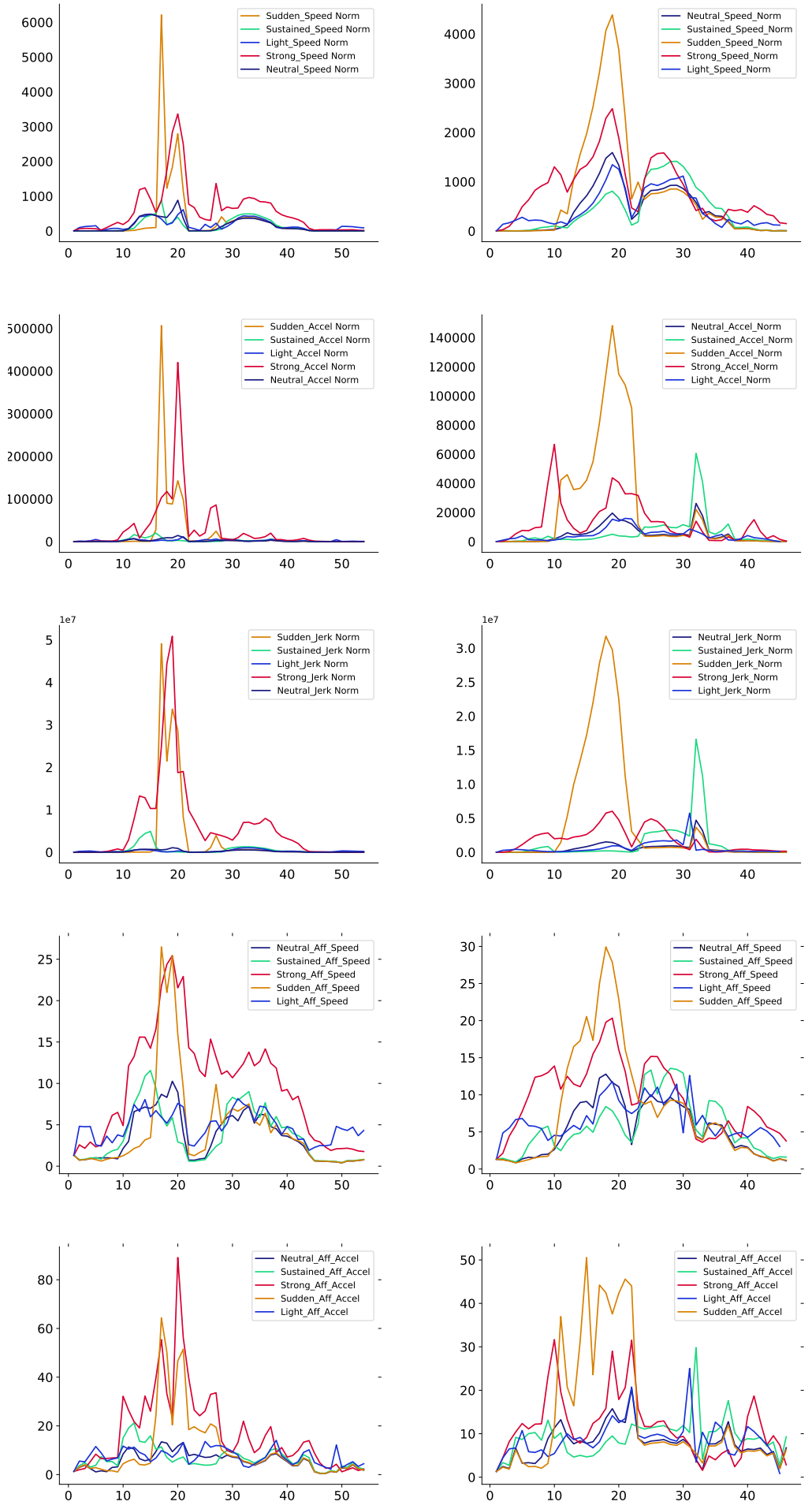


Figure 5.16: Stomp and Kick animations geometric properties for neutral and stylized version. Speed norm, Acceleration norm, Jerk norm, Egui-Affine Speed and Equi-affine Acceleration are represented for the different animation styles and for one end-effector. Left: Stomp right foot properties. Right: Kick right foot properties.

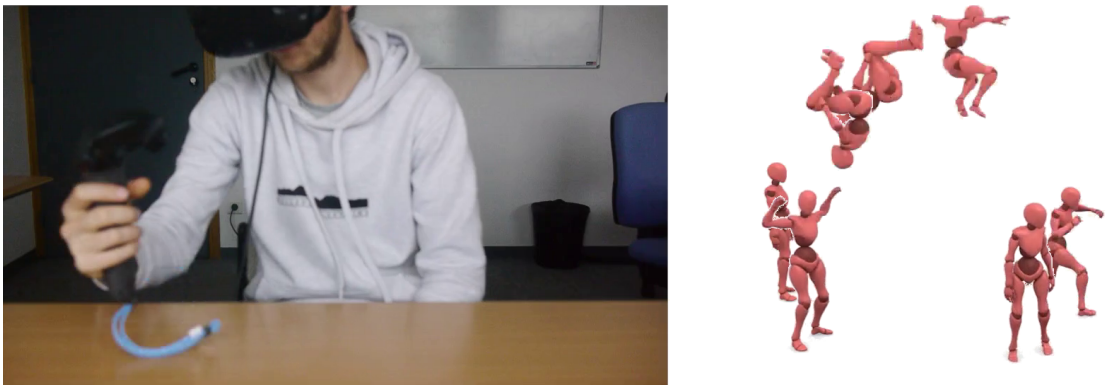


Figure 5.17: SMD example. Left: User performing Jump, Punch, Back Flip and Raise Arms actions. Right: Corresponding generated animation sequence.

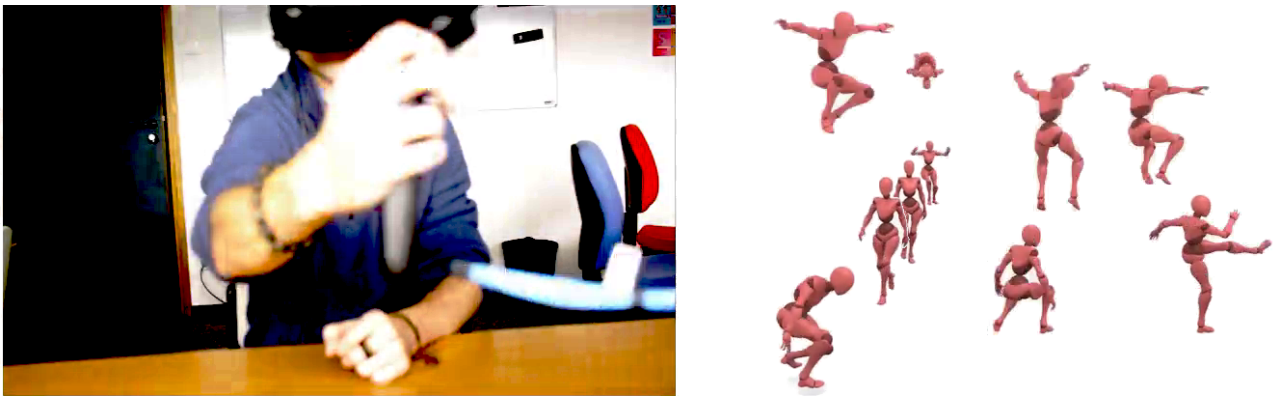


Figure 5.18: SMD example. Left: User performing Flip, Jump, Kick and Stomp actions. Right: Corresponding generated animation sequence.

Figure 5.19: SMD example. Left: User performing ‚Jump, Pick Up, Throw, Kick, Stomp, Punch and Raise Arms actions. Right: Corresponding generated animation sequence.

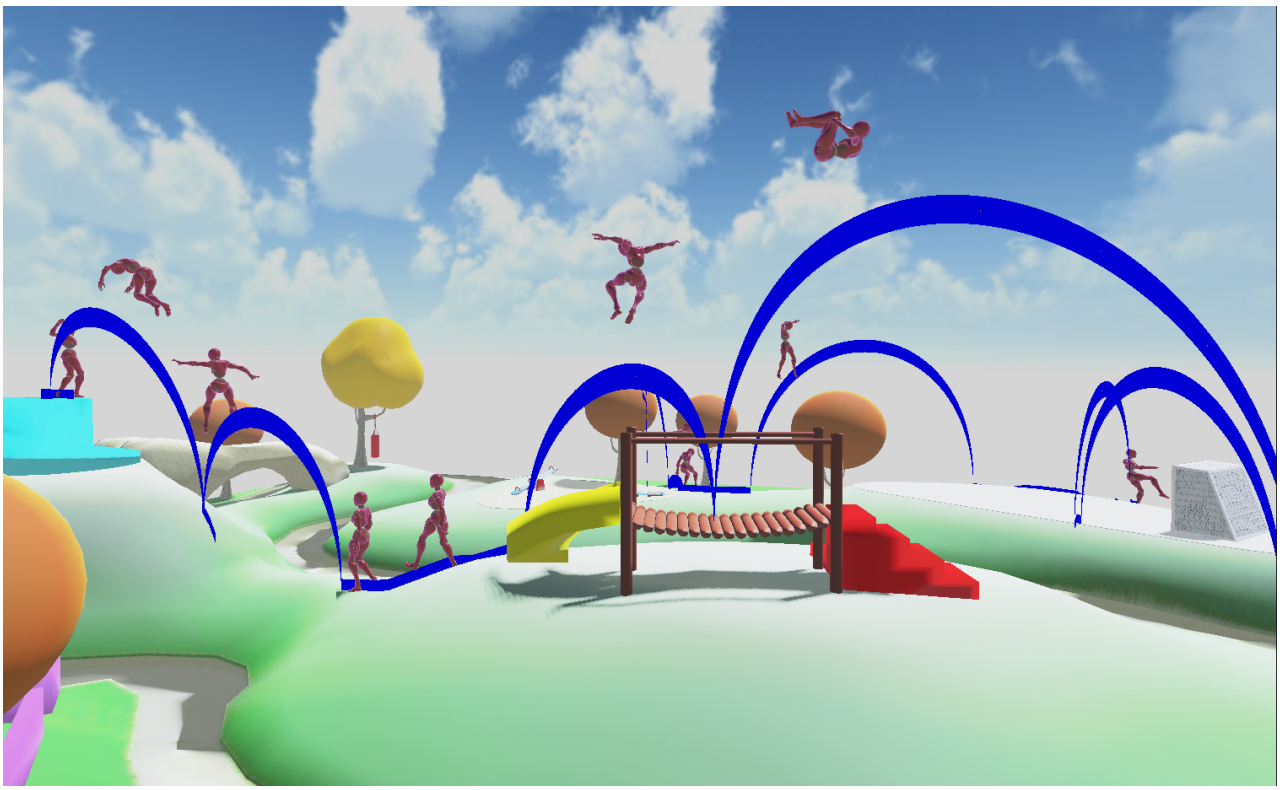


Figure 5.20: SMD example on a kids garden environment combining multiple jumps and intermediate actions.

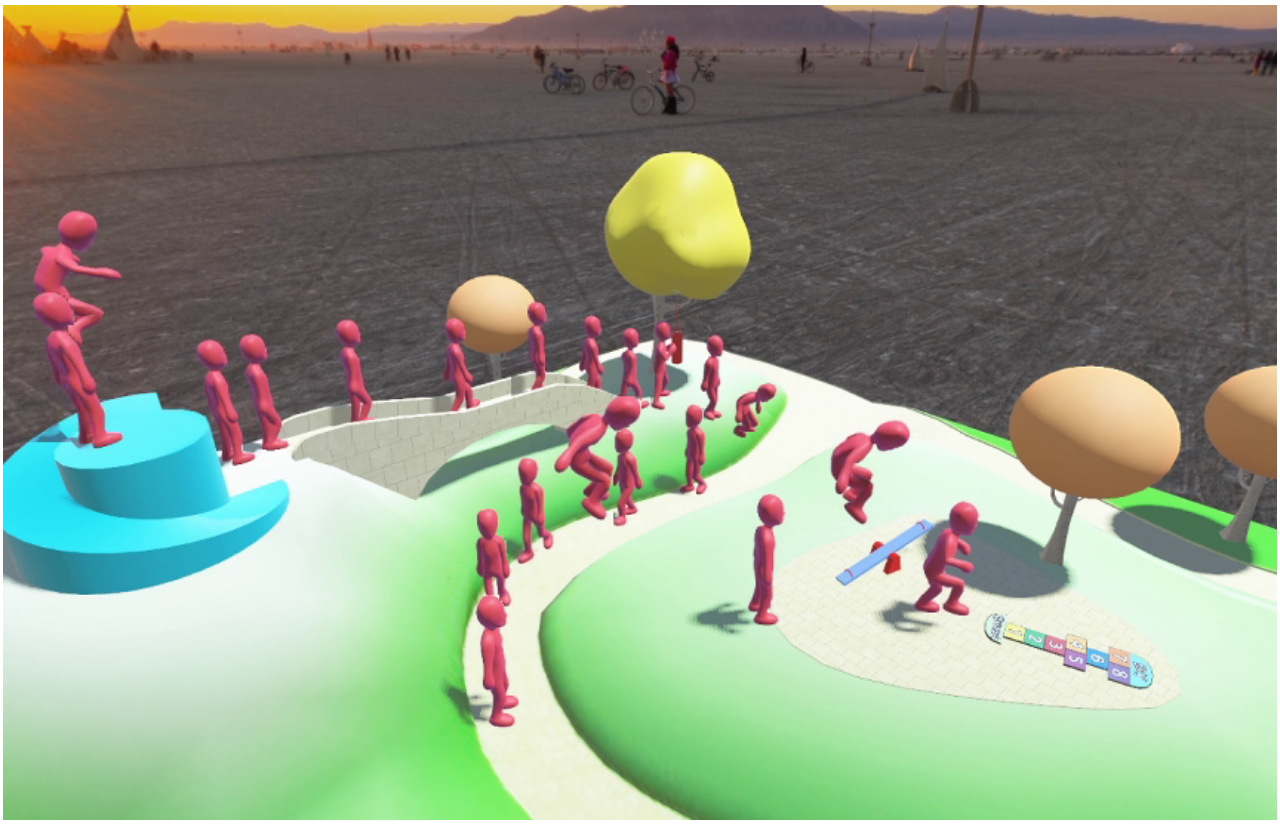


Figure 5.21: SMD example on a kids garden environment combining multiple jumps and intermediate actions. Note that we used a different model and skeleton structure for this example.

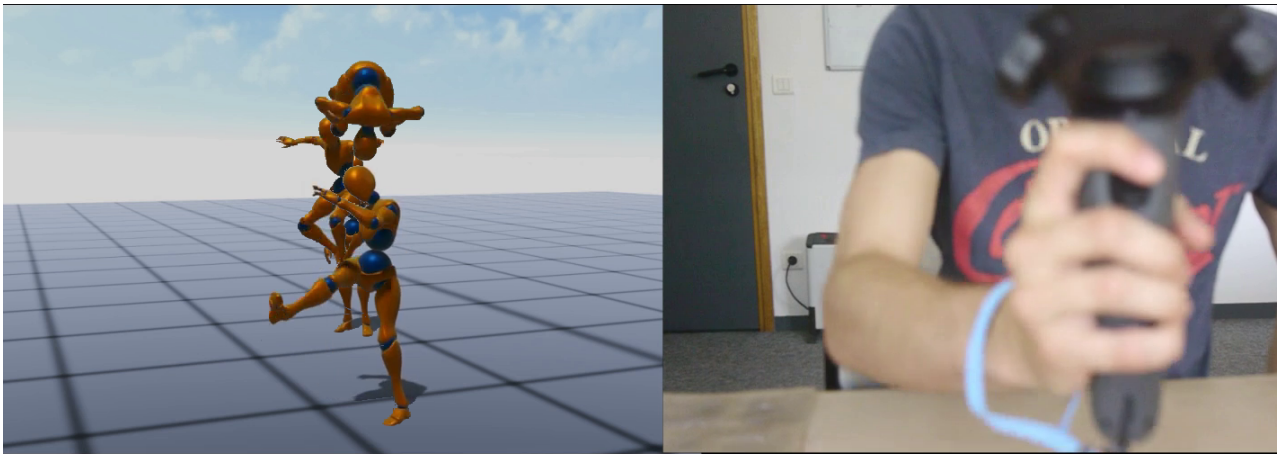


Figure 5.22: Stylized SMD example. The user performed Sudden and Light hand gestures which were transferred into a corresponding stylized animation sequence.

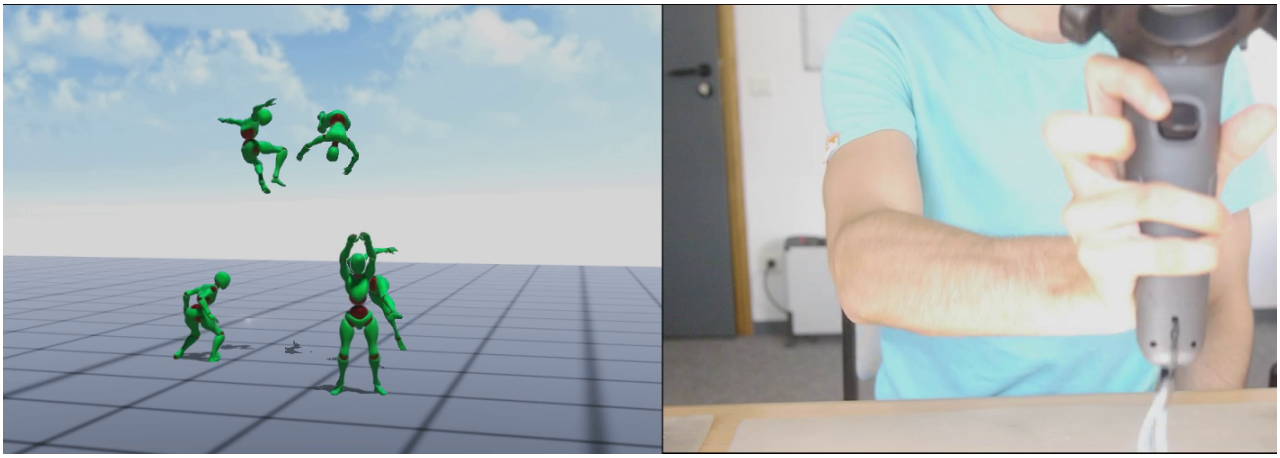


Figure 5.23: Stylized SMD example. The user performed Sustained and Strong hand gestures which were transferred into a corresponding stylized animation sequence.

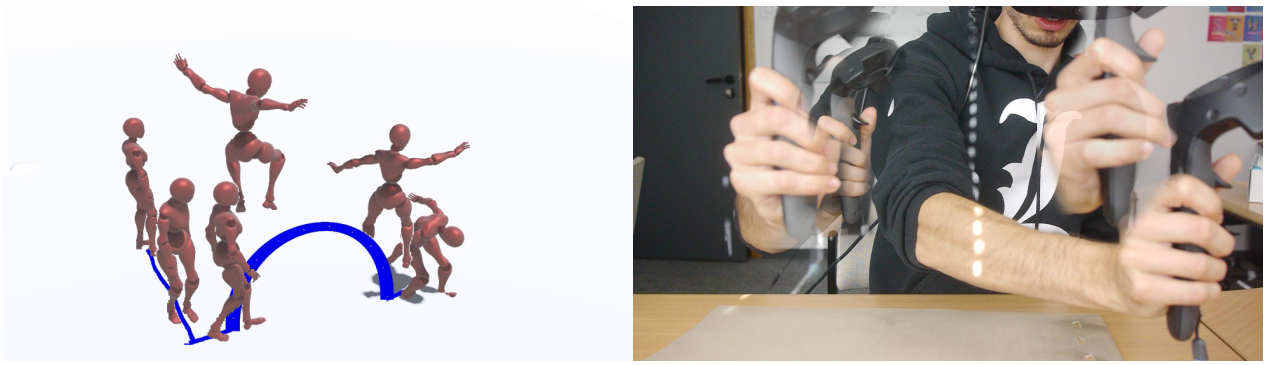


Figure 5.24: Stylized version of our main single character performance transfer example: Our system successively transferred a Sudden/Light Walk, a Sustained/Light Walk, a Light Jump and a Strong Stomp animations.

Chapter 6

Performance Based Storytelling

In previous chapters, we introduced the concept of Spatial Motion Doodles and how we process them to generate animation sequences in the context of single character animation. The main goal of our system is to allow users to control several characters at time, make them interact or perform synchronized animations, letting users express their narrative inspiration and create animated stories.

In this chapter we further describe how we process multiple SMDs at a time and introduce new tools to let users create multi-character sequences. Moreover, we show how our system can be used to create animated stories with multiple characters using two handed interactions. It is also important to note that we do not study and apply motion qualities recognition and transfer in this chapter.

We first show how to transfer multiple SMDs in the case where characters performs independent actions and show transfer results in section 6.1. We then detail how we model, recognize and transfer interactions in section 6.2, and introduce interaction constraints allowing users to create more coherent stories in section 6.3. In section 6.4, we describe an algorithm adding implicit actions which are necessary for transferring interactions. Finally, we show and discuss interaction transfer results in section 6.5.

6.1 Independent Characters

Using the HTC Vive controllers as acquisition devices enables us to record two trajectories at the same time. The two SMDs we extract from these trajectories are synchronized and represent characters' motion evolving in the same 3D space. These properties are actually quite powerful as they allow users to perform synchronized actions and action/reaction gestures while directing the path each character will travel. This opens up many possibilities for creating diverse scenarios.

From those two synchronized trajectories, we can transfer two characters plays by parsing their related SMDs in parallel and apply the same recognition and transfer algorithms as in the one character case to both of them.

Still, we needed to confirm that users were actually able to use our system while manipulating at least two props at the same time. More precisely, we had to assert that they were able to perform synchronized actions, alternating role actions and two different actions at the same time. Thus, we ran several experiments in which users played with two characters at the same time in different contexts. No interactions between characters were handled in those experiments but users could "fake" them by executing synchronized or successive gestures simulating character reactions to each other.

In a first experiment, we asked a user to improvise a fight scene between two characters in a video game style (figure 6.1). The user first selected fighting animations from our database and trained our system to recognize his gestures for each animation.

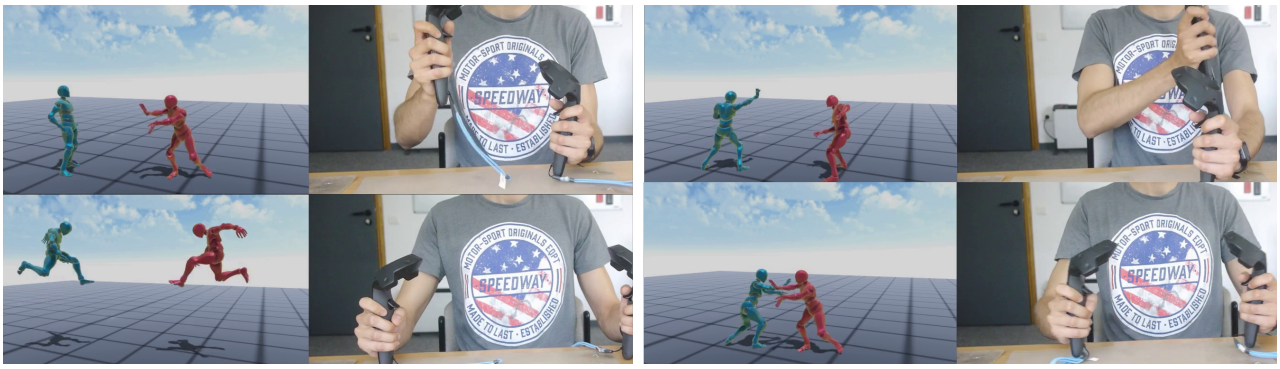


Figure 6.1: Fighting scene between two characters created by a single user simultaneously using both HTC Vive controllers

While demonstrating that the user was able to author two synchronized animation sequences at a time, this particular example shows the interest of our system for real-time applications like video games, although our system operates as an offline process.

A second experiment was done in collaboration with a semi-professional theater director who reproduced a scene from Don Juan by Moliere in four different variations. In this scene extracted from Scene V Act V, Don Juan and his valet Sgnanarelle are arguing for whether Don Juan should repent for his sins or not. In this four variations, the balance of power differs, alternating from making Sganarelle submit to Don Juan and taking the opposite behavior. In figures 6.2,6.3,6.4 and 6.5 we show the four variations sequences with their

related scripts. Note that in this experiment, we only reproduced the end of this scene (last two paragraphs) as we wanted to focus on manipulating only two characters at a time.

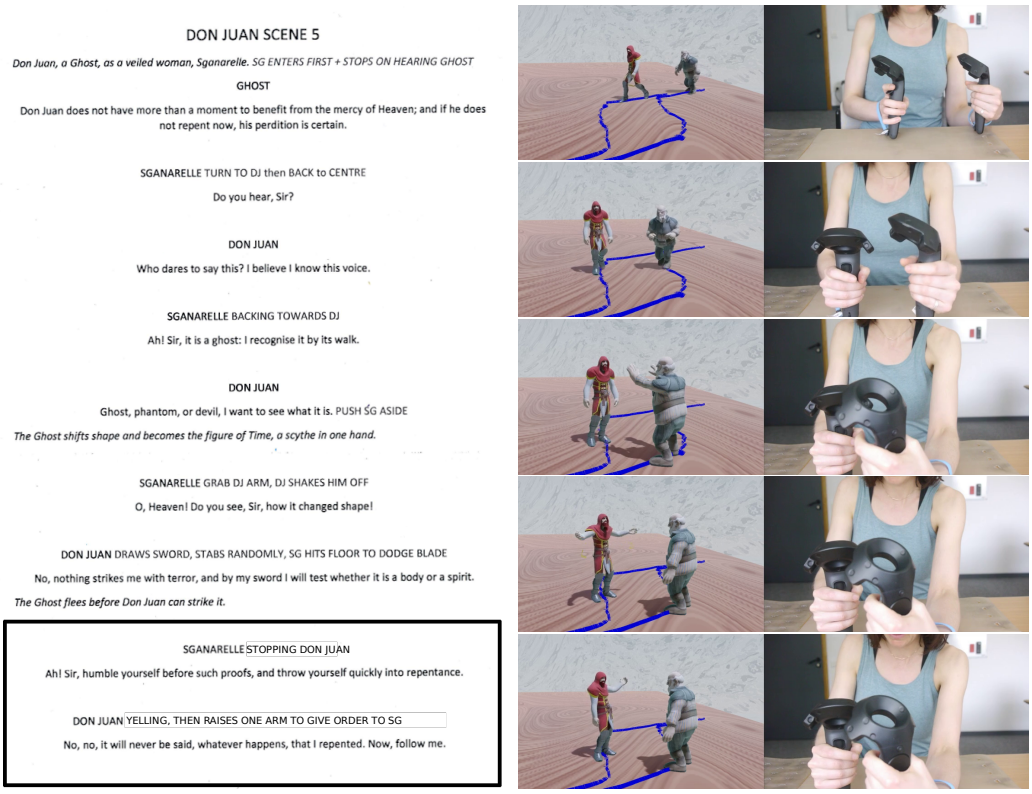


Figure 6.2: Don Juan Scene first variation. Sganarelle first Stops Don Juan asking him to repent for his sins. Don Juan answers that he does not plan to repent while Yelling and orders (Raises one arm) Sganarelle to follow him. In this variation, Sganarelle follows him.

This experiment showed the interest and usability of our system for quickly drafting different stagings of a given scene. It also demonstrates the narrative freedom we gives to users in scenarios where characters react to each other. In a self-assessment of the experiment, the user found that the animations generated in the four variations correctly reproduced her dramatic intentions. She also complained about the use of the HTC Vive controllers which was sometime uncomfortable especially for making full turns.

6.2 Interacting characters

In previous examples, we showed how we can process multiple independent character SMDs to create animated sequences. Although, users could "fake"

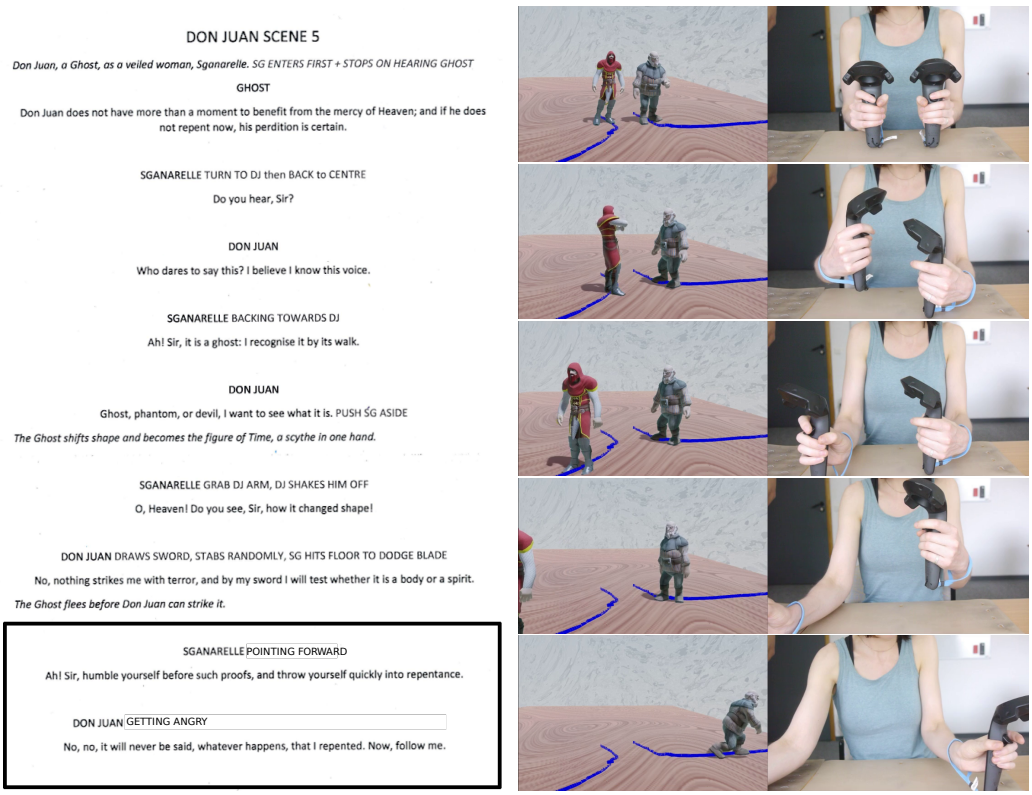


Figure 6.3: Don Juan Scene second variation. Sganarelle Points forward while asking Don Juan to repent for his sins. Don Juan Gets Angry and answers that he does not plan to repent and orders (Raises one arm) Sganarelle to follow him. In this variation, Sganarelle executes a Dismissing gesture and follows his own path.

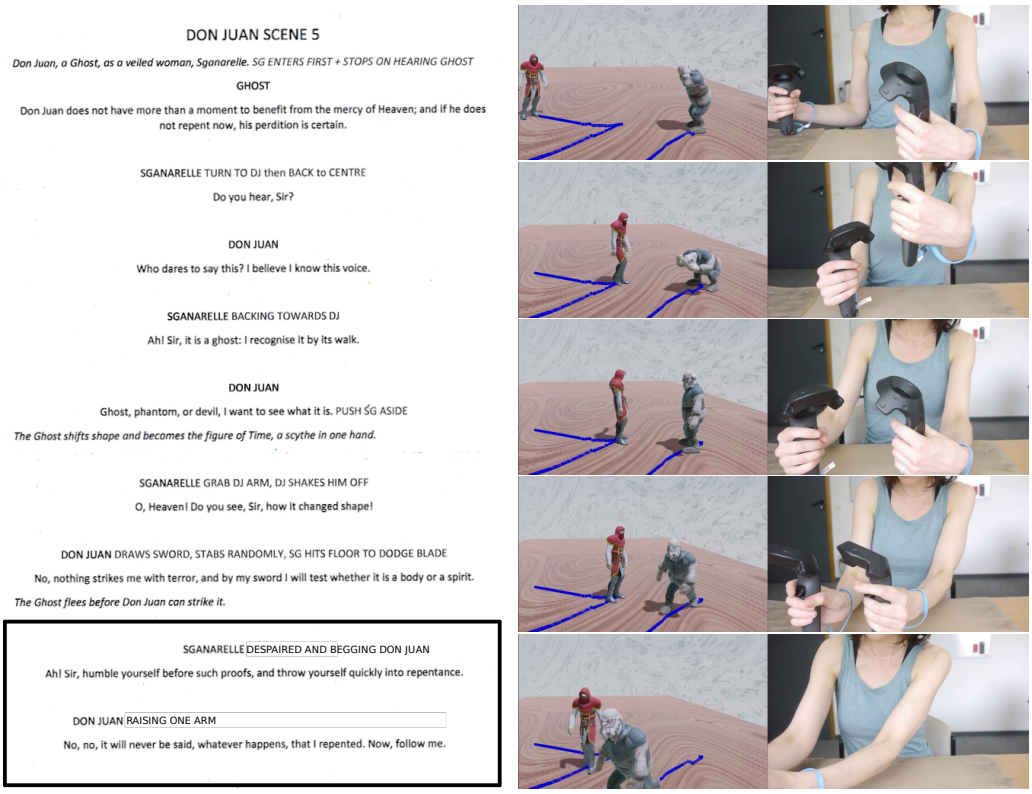


Figure 6.4: Don Juan Scene third variation. Sganarelle is Despaired and begs Don Juan to repent for his sins. Don Juan answers that he does not plan to repent while ordering (Raises one arm) Sganarelle to follow him. In this variation, Sganarelle follows him.

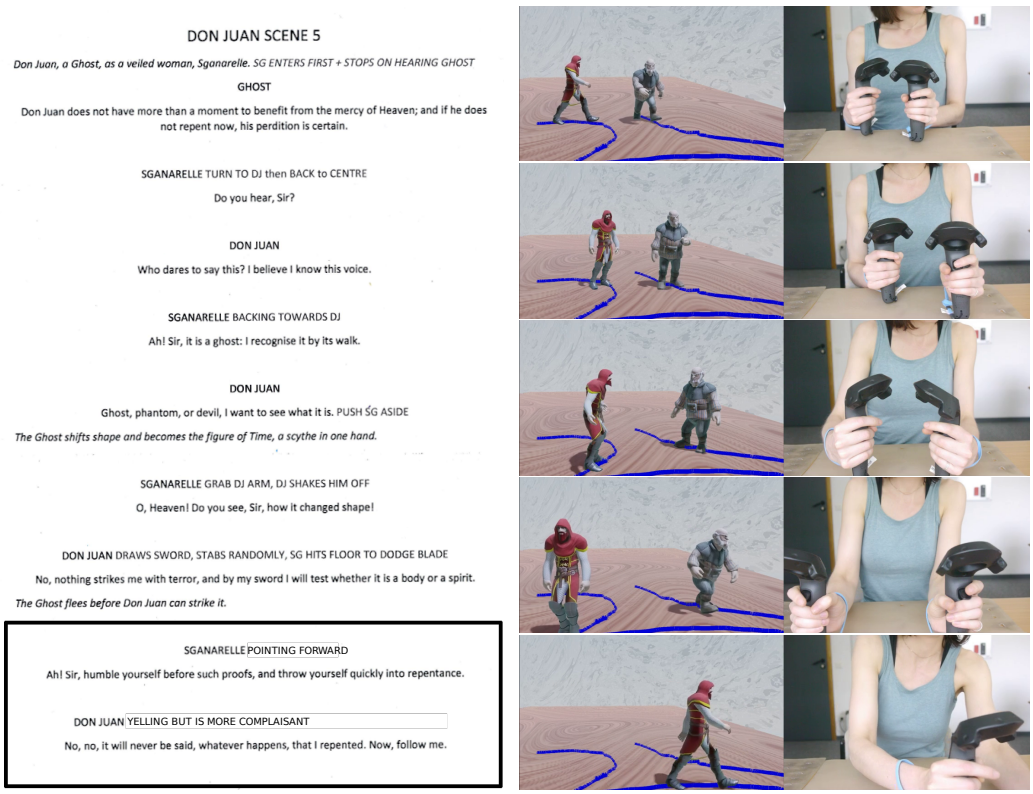


Figure 6.5: Don Juan Scene fourth variation. Sganarelle Points forward while asking Don Juan to repent for his sins. Don Juan answers that he does not plan to repent while Yelling and orders (Raises one arm) him to follow him. In this variation, Sganarelle takes his own path and is followed by Don Juan.

character interactions in the sense that they could act some actions/reactions or synchronized motions, our system was not designed to handle character interactions and treat them in an appropriate way.

In this section we show how we extend our action recognition algorithm presented in chapter 4 to recognize interactions. We then detail how we transfer them into the generated animation sequences.

6.2.1 Interaction Recognition

As described in chapter 4, our action recognition algorithm behaves like a compiler, performing lexical and syntax analysis with respect to motion token regular expressions. To achieve our narrative goals, we need to extend the recognition system of chapter 4 so it can handle interactions. Indeed, while our system can recognize independent action sequences, it does not account for token sequence influence over other sequences meaning that it lacks contextual analysis, which is also part of a compiler. This issue is addressed by the Contexter which reads characters trajectories in parallel and identify interactions between them. We will now introduce the formal definition of interactions.

Interaction representation. An interaction between n characters is de-

fined by by A_n, a set of n actions corresponding to the regular expressions each character should execute to trigger the interaction, and C_m, a set of m additional constraints the characters should respect. Our system can handle different constraints types which are detailed in section 6.3. They are then combined either through union or intersection. We define an interaction I as the pair $\{A_n, C_m\}$. As an example, in section 6.5 a user made two character Kiss which was modeled as an interaction whose action set A_2 is $((T\_N\_Z)+(T\_Z)+(T\_N\_Z)+(T\_Z)+, (T\_N\_Z)+(T\_Z)+(T\_N\_Z)+(T\_Z)+)$ and constraint set $C_3 = (D(0, 1, Intimate), V(0, 1, 0.5), V(0, 1, 0.5))$, where the $Intimate$ is a distance computed using the characters' position and bounding boxes (see section 6.3.1).

The Contexter then performs interaction recognition as follows: while reading each character doodle in parallel, meaning that each character Segmenter processes its input trajectory at the same time, the Contexter store their token outputs and looks for token sequences matching interactions regular expressions. When Segmenters output new tokens to the Contexter, the later automatically transmits them to the corresponding Matcher which proceeds to independent action recognition. Thus for each character, in addition to their token sequence stored in their respective Matcher, we store an additional interaction token sequence for each character which are updated at each new output token tuple.

In addition, when a new output token tuple is read, the Contexter receives each Matcher regular expression match results and either adds the matched actions to the corresponding character action sequence or store them in a stack if an interaction is currently partially matched. Later on, if the interaction is fully matched in terms of regular expressions and constraints it is them added to the characters' action sequences instead of the stored action stack (which is then emptied). On the contrary, when the interaction is considered as not matched (either in terms of constraint or regular expression) the stored action stack is added to the corresponding character's action sequence. In both cases the interaction token sequences are reset when adding new actions to the characters' action sequences. We can highlight that this process prioritizes interactions over independent actions which is coherent with the fact that they are more difficult to trigger than individual actions. They also bring more narrative depth to the performed stories.

Let us now detail how interaction matching is performed given n interaction sequences. For each interaction I between k characters, we search for all k-sequences satisfying the interaction constraints and regular expressions. In total, we have to test $k!\binom{n}{k}$ configurations. In practice $n, k \leq 4$ which always

necessitate a reasonable amount of configurations to test. In our examples, we only recorded two characters play and therefore $n = k = 2$ in our test cases.

When checking if a combination of k token sequences S_k matches an interaction regular expression set, several orderings (configurations) might satisfy them. In that case, for each regular expression R_i of I we keep the longest sequence S_j matching it. When all sequences as well as all constraints are satisfied, each character will execute the action of attached to the regular expression matching its token sequence.

It is important to mention that, as we introduced the concept of constraints for interactions, they can also be used for single character actions (see section 6.3). Moreover, motion quality constraints could be used to differentiate actions with the same regular expressions (like Walk and Run).

As we added the Contexter as the final component of our action and interaction recognition algorithm, we summarized our whole recognition system in Figure 6.6.

6.2.2 Interaction Transfer

Detecting and generating adequate 3D character interactions is an hard research topic that has been studied in different contexts (see chapter 2).

Moreover, [Kim et al., 2012] proposed an interaction detection and generation algorithm based on contact, proximity and synchronization of characters triggering adapted animations depending on their relative position orientation and velocities. In a similar fashion, our algorithm directly embed a synchronicity constraint as interaction regular expressions must be executed nearly at the same time because SMD are read in parallel. More precisely, for an interaction I between n characters and S_n its corresponding regular expressions set, we note $[t^i_{start} : t^i_{end}]$ the time interval during which the i^{th} regular expression is performed. By construction our recognition algorithm can recognize an interaction only if $\bigcap_{i \in [1:n]} [t^i_{start} : t^i_{end}] \neq \emptyset$. This native synchronicity constraint also imposes that other additional constraints are respected during this intersection time interval. In section 6.3, we introduce and discuss the different types of constraints our system can manage.

Knowing the time intervals during which interactions must be executed as well as the characters' position and orientation in these intervals, we then need a way to correctly transpose them into a multi-character animation. Contrarily

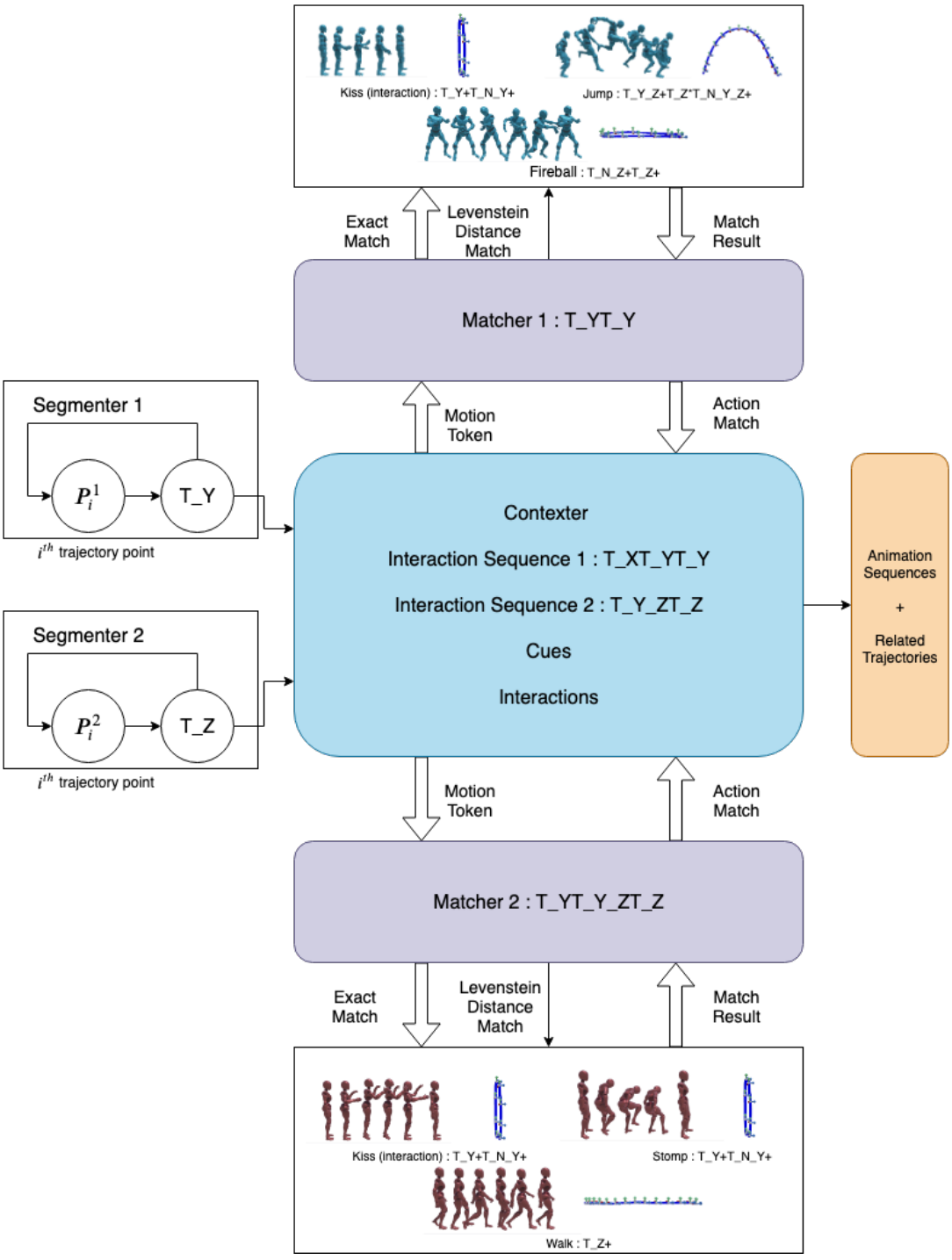


Figure 6.6: Action and Interaction recognition system. Each character's trajectory is processed by a Segmenter which successively extract motion tokens from input 6D points which corresponds to linear or angular velocity direction changes. When a motion token is extracted it is directly transmitted to the Contexter and to the trajectories' Matcher. While each Matcher looks for single character action matching (exact and partial) the Contexter detects interactions (which have priority over actions). The final output is a sequence of actions and interactions for each character with related correspondences in their respective trajectories.

to single character animation transfer, interactions can impose challenging constraints especially in terms of contact preservation between characters. Indeed, in our case where character space-time trajectories are fully constrained by their corresponding SMD executing interaction can be done in different ways. In this section, we present two different interaction transfer methods, the first one correctly placing characters before playing interaction animations and the second one, more flexible, adapting independent single character animations to build interactions.

Interaction animations and automatic placement

The first method consists in storing each character interaction inside one single animation which contains joint and root motion information for each character. Having this configuration we can easily compute the relative distance of each character at the beginning of the interaction as well as their initial orientation. Thus before executing an interaction at given point of the characters trajectories we need to move each character such that they fit their initial relative position and orientation at the first frame of the interaction.

To do so, we choose one character as being a reference for the others, meaning that we consider that it is correctly placed. We then can generate a path for the remaining characters which places them correctly with respect to their initial position and orientation at the first frame. While these characters move to their goal destination the reference character waits in an Idle state. Then, once both characters are correctly placed we can execute the interaction. Finally, we need to move the characters back to their previous positions before interacting such that they can continue to follow the original trajectory drawn by the user. However, using this method we need to account for the time each additional Walk animation takes in order the preserve future synchronicity between characters. This can be done in two ways. Either we reduce the duration of the interaction by the duration of the Walk round-trip or we make characters which are not part of the interaction wait while interacting characters Walk.

Let us mathematically formalize how we perform trajectory transfer and animation sequencing using this first method. Keeping the notation of chapter 5, each character K_k possesses a transferred trajectory P_i^k it will follow during the animation sequence. In addition, to the spatial offset s_{off} we increment to cope with in-place actions trajectory shifts, we introduce a temporal offset t_{off} which copes with the duration of the additional Walk animations we introduce to correctly place the interacting characters and restore their original position

after the interaction. For each interaction between n characters for which the related interaction curve segment occurs between $P^k_{i_k start}$ and $P^k_{i_k end}$ (with $k \in [1:n]$), we first determine the distance the $n-1$ characters must travel to reach the correct interaction configuration, excluding the first one which is used as a reference. To do so, for $k \in [2:n]$ we compute $\overrightarrow{I^1_0 I^k_0}$, which is the vector between the first and k^{th} character center of gravity at the first frame of the interaction animation. We also compute their relative orientations $(q^1_0)^{-1} q^k_0$ (expressed with quaternions) at the first frame of the interaction. The travel distance for each k^{th} character is then equal to:

$$D_k = \|\vec{v_k}\| \tag{6.1}$$

$$\text{with } \vec{v_k} = d_k - P^k_{i_k start} \tag{6.2}$$

$$\text{and } \vec{d_k} = P^1_{i_1 start} + \overrightarrow{I^1_0 I^k_0} \tag{6.3}$$

Using the algorithm we describe in section 6.4 we can generate a Walking trajectory for those characters that reach the correct position $\vec{d_k}$ and orientation $q^1_{i_1 start}(q^1_0)^{-1} q^k_0$. This algorithm computes $n-1$ space-time trajectories $\{T^k_i\}_{[1:m_k]}$ which may have different point number and durations. Considering the $j^{th} \in [2:n]$ generated trajectory to have the longest duration we complete all other trajectories $\{T^k_i\}_{[1:m_k]}$ with $k \in [1:n] \setminus j$ (note that we include the first character here) with l_k points equal to $T^k_{m_k}$, making all characters wait for the j^{th} to end its Walk animation. Finally, l_k is computed as the product: to $(t(T^j_{m_j}) - t(T^k_{m_k}))F_d$, F_d being the recording device's frequency.

At this point, we can make two choice, reducing interaction time by $\delta_t = (T^j_{m_j}) - t(T^j_0)$ or increment t_{offset} by $2\delta_t$, also accounting for the time the characters will take to reach their original position. Additionally, the later solution also requires that we make non interacting characters waits in an Idle state for 2δ to keep future synchronicity. This new action is added as soon as these characters end the current animation they were executing when the interaction started. Choosing to not reduce the interaction duration seems to be a more reasonable choice as in some case, the additional Walk animations might last longer than the interaction it self completely masking it and in all other cases it the interaction may be played really fast decreasing its credibility. Finally, at the end of the interaction each character get back to its original position by following $\{T^k_i\}_{[1:m_k]}$ in the reverse order and in which we applied a π-rotation with respect to the up axis at each point, making the front vector facing in the opposite direction.

While this technique allows us to transfer interactions in accordance with

the character configuration present in the animation data, acquiring this data is actually tedious. For example, in section 6.5, we want 4 characters to perform a Slap/Slapped interaction. Acquiring all character interaction configurations actually requires $2!\binom{4}{2} = 16$ animations. While using motion capture and animation retargetting methods could ease the database building process we did not use this method for transferring interactions.

Adapting independent animations

We propose a second interaction transfer method which necessitates less animation data and which is in the line of several research works in multi-character animations. Moreover, these works use several single character animation data to animate interactions and constraint each characters' independent animation with space-time constraints, resulting in plausible and adaptable interactions. In our case, we want to adapt character animations such that contacts between interacting characters are maintained independently of their positions at the beginning of the interaction and of their proportions.

Constraining multi-characters positions and contacts at given frames has been explored by several works such as [Liu et al., 2006] or [Kim et al., 2009]. Moreover [Kim et al., 2009], propose a method for editing multi-character animations while preserving contact constraints in an as-rigid-as-possible manner, meaning that all characters' joints orientations at frame i around the frame of the contact constraint will be modified in a way that preserves the initial differences with the neighboring orientation of frame $i-1$ and $i+1$.

In our system, we use the same kind of constraints to adapt interactions. Similarly to [Liu et al., 2006], each interaction can impose space-time constraints $C_i(p,t) = p_i(t) - p$ to a character i^{th} bone's position at time t. This constraint is then solved using IK on the targetted bone in an as-rigid-as possible manner. For a given bone we compute its trajectory $\{p_i\}$ during the whole animation and notice that:

$$\forall i, p_i = p_{i-1} + \alpha_i \vec{A_i} + \beta_i \vec{B_i} \tag{6.4}$$

$$\text{with } \vec{A_i} = \frac{p_{i+1} - p_{i-1}}{\|p_{i+1} - p_{i-1}\|} \tag{6.5}$$

This formulation translates the fact that p_i can be expressed in a frame in which the frontal vector $\vec{A_i}$ and left vector $\vec{B_i}$ are contained in the $p_{i-1}\hat{p_i}p_{i+1}$ plane, with p_i being expressed with respect to its neighbors. Using simple trigonometry (see figure 6.7) it is quite simple to compute α_i and β_i:

$$\alpha_i = dot(p_i - p_{i-1}\vec{A}_i)$$
$$\beta_i = \left\| cross(p_i - p_{i-1}\vec{A}_i) \right\|$$

Finally, we can deduce:

$$\vec{B}_i = \frac{p_i - p_{i-1} - \alpha_i \vec{A}_i}{\beta_i} \tag{6.6}$$

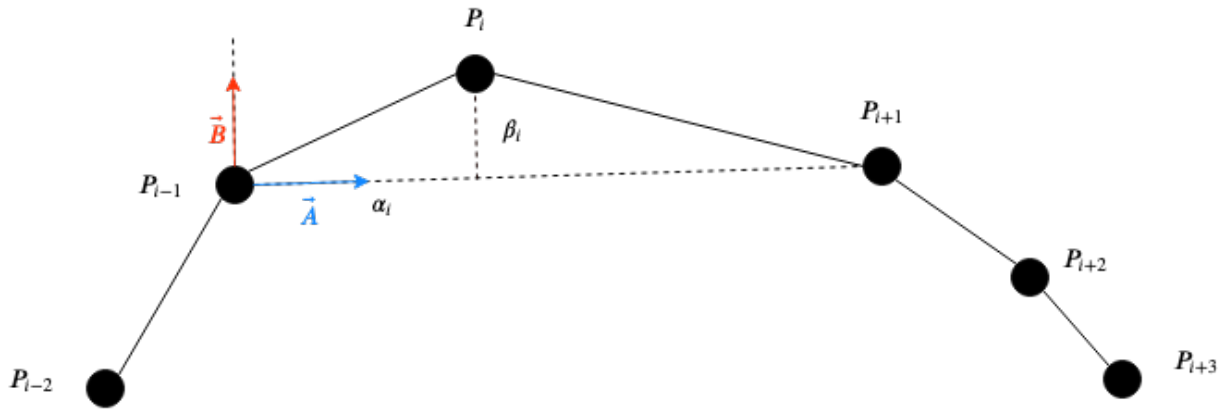


Figure 6.7: Expressing a point curve P_i with respect to its two neighbors. This point is computed in its previous neighbor local frame directed by the $P_{i-1}P_{i+1}$ vector.

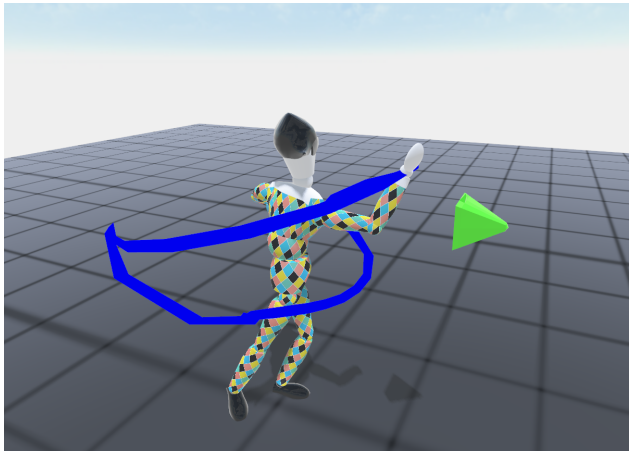
Using this characterization of each point of the bone's trajectory with respect to its neighbors, we seek to conserve each original α_i and β_i when deforming the trajectory to address the $C_i(p,t)$ constraint. We then compute the new bone trajectory as an optimization process minimizing the cost function L_i:

$$L_i(p_0, ..., p_n) = \sum_j \left\| (\alpha_j^{init}, \beta_j^{init}) - (\alpha_j, \beta_j) \right\|^2 + \gamma \| C_i(p,t) \|^2 \tag{6.7}$$

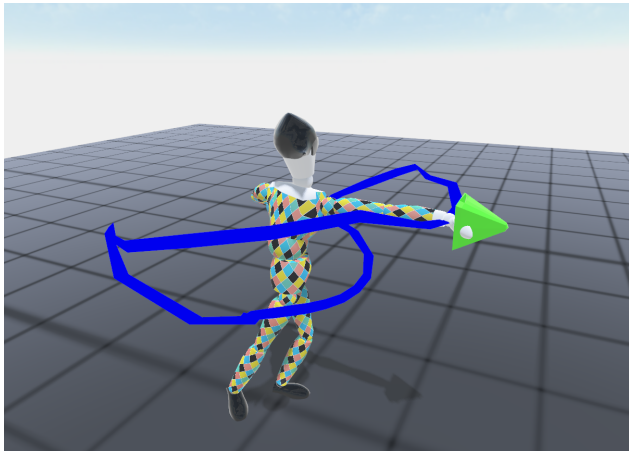
Using a gradient descent method we can update the p_i trajectory at each step of the optimization process in which: $\forall j, p_j = p_j - \epsilon \nabla L_i(p_0, ..., p_n)_j$. Furthermore, we propose to users to link n space-time constraints $C_i(p_j, t_j)$ in which p_j is a common value between all the constraints. Moreover, we propose two different policies for computing p_j:

- $p_j = \frac{\sum_j p_j}{n}$, assigning p_j to the mean of the target constraint
- $p_j = p_i$ with $i \in [1:n]$ assigning p_j to one of the target constraint

Note that this method preserves the interaction duration executed by the user and do not necessitate any Walk and Idle animations addition. We demonstrate the effect of the contact constraint for a Slap and Kick action as shown on figure 6.8 and 6.9 both constrained to reach the green target.

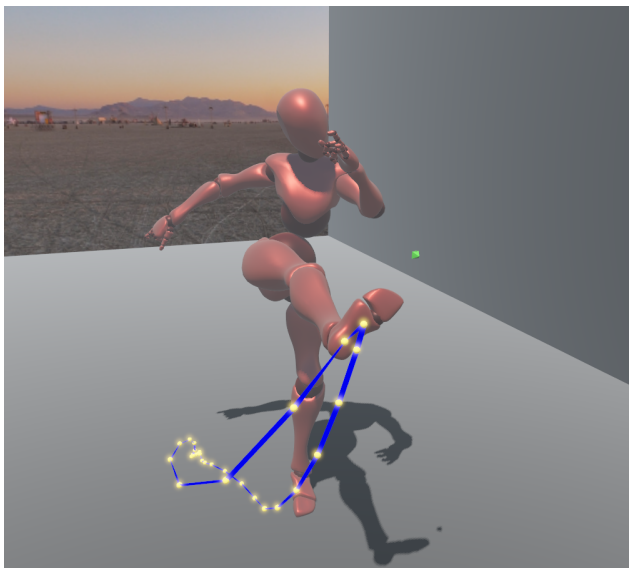


(a) Default Slap animation which do not reach the green target.

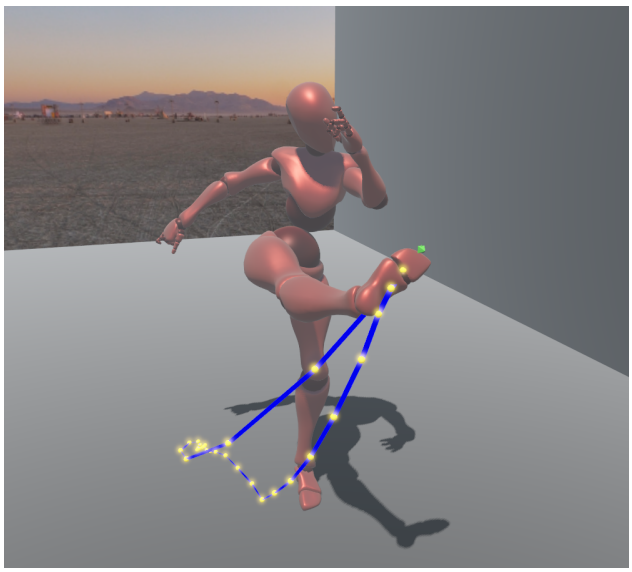


(b) Deformed Slap animation using our optimization algorithm making the character reach the green target.

Figure 6.8: Slap animation position constrained example.



(a) Default Kick animation which do not reach the green target.



(b) Deformed Kick animation using our optimization algorithm making the character reach the green target.

Figure 6.9: Kick animation position constrained example.

The second method we presented is our preferred choice for transferring interactions as it is more flexible and requires less animation data that takes time to acquire (but is less faithful to the original animations). In practice, the two methods could be used but entirely depends on the data available and on users preferences regarding the quality of the transferred animations.

6.3 Interaction Constraints

In previous sections, we described how we transfer interaction animations after they have been recognized by our system. Moreover, we mentioned that additional constraints have to be respected in order to trigger interactions. In this section we detail the different kind of constraints our current system can handle. More particularly, we introduce distance constraints between characters, field of view constraints and behavior-based constraints.

6.3.1 Distance Constraints

Distance to an environment object or to other characters is a commonly used constraint to trigger events in video games and other interactive applications. It is rather trivial that interactions in which characters need to make contact or to communicate necessitate that they are close enough to execute it. Likewise one character cannot open a door if it is not close enough. For two characters i, j whose position are P_i and P_j, we define the distance constraint $D(i, j, \alpha) = \|P_i - P_j\| \leq \alpha$. In practice imposing a global value for α might necessitate some tweaking depending on the characters proportions. Instead we propose a to define distance threshold values based on sociology factors as introduced by [Hall, 1968]. This work introduces four distances thresholds, Intimate, Personal, Social, Public in increasing order, whose borders symbolize censorial and behavioral changes for human beings. This theory has been a computer graphics research topic aiming at finding ways to adapt virtual agents and environment to users [Kastanis and Slater, 2012], [Sanz et al., 2015], [Llobera et al., 2010]. We also propose to use this terminology and propose a more generalized (and rougher) way to compute those thresholds using character and objects' spherical bounding box. More precisely, for two characters i, j whose Spherical Bounding Boxes radius are r_i and r_j and positions P_i and P_j we defined the four distance threshold as:

- Intimate: $\|P_i - P_j\| \leq r_i + r_j$
- Personal: $\|P_i - P_j\| \leq 2(r_i + r_j)$
- Social: $\|P_i - P_j\| \leq 4(r_i + r_j)$
- Public: $\|P_i - P_j\| \leq 8(r_i + r_j)$

Therefore, constraining two characters distance such that their Personal area intersects for a Slap/Slapped interaction (as a example) can be written $D(i, j, Personal)$.

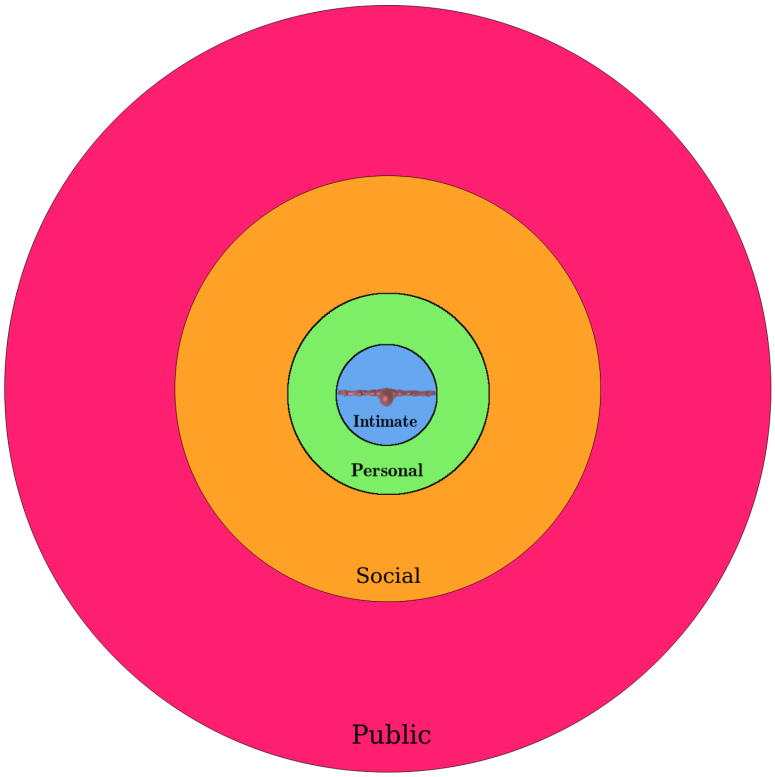


Figure 6.10: Distance constraints thresholds computed from the radius of the character spherical bounding box.

6.3.2 Field of View Constraints

Another typical constraint used for triggering interactions is to determine whether two characters can actually see each other. We propose two different formulations to simulate this constraint:

- $dot(P_j - P_i, \vec{F_i}) \geq \gamma$, written as $\gamma \geq 0$ or $V(i, j, \gamma)$
- $dot(\vec{F_i}, \vec{F_j}) \leq \beta$, written as $\beta \neq 0$ or $MV(i, j, \beta)$

P_i and P_j being the position of the two characters and F_i, F_j their respective front vector. The first constraint actually forces one character to see the other while the second one forces the two characters to see each other at the same time. Choosing one formulation or the other fully depends on the context and the aim of interactions. For instance a Push/Pushed interaction might not necessitate that the pushed character sees the pushing one while a shake hand interaction obviously necessitates that the two characters see each other.

6.3.3 Behavior Constraints

For narrative exploration, we would like our system to include behavior constraints forcing characters to be in a certain state to execute actions or interaction and which usually change their own state or the state of other interacting

characters as a consequence. This kind of constraints are also commonly used for authoring event centered animations and are at the heart of works like [Kapadia et al., 2011] and [Shoulson et al., 2013].

More generally, we define a character's state as a set of boolean variables $S_i(s_1, ..., s_n)$ which are used as pre-conditions (constraints) of actions and interactions. In addition, these state variables can be updated when an action or interaction ends, defining post-conditions. As a example, Fall on the ground is an action requiring the falling character to be in a Stand Up state and changes this state to Fallen at the end of the corresponding animation. At this point, it is important to impose that each action or interaction changing one state variable must be invertible in order to maintain a large set of possible actions at all points during the story.

6.4 Implicit Actions

In previous sections and chapters, we mentioned that, at some point, we need to add extra Idle and Walk animations to correctly transfer action and interactions. In this section we describe a method to procedurally add those animations in the form of a SMD segment that will be correctly recognized as the implicit action we want to add and following space-time constraints.

We propose a method for generating a single character SMD from actions regular expression and additional constraints. This method is decomposed into three steps: first we generate a minimal SMD from one of the Levenshtein expression of the action, guaranteeing that the generated trajectory will be recognized as intended. Then for actions inducing displacements like Jump or Walk, an end position and orientation constraint might be added. We thus, rotate and scale the minimal SMD with respect to the trajectory starting point such that the end point reach the desired constraint. Finally, we proceed to a transition smoothing process especially for walking animations (in place actions are not concerned by this additional step) which greatly reduces sudden turns using an optimization process.

Minimal SMD Generation. For a given action defined by its regular expression and a given its duration t_{tot} we propose an algorithm to generate an equivalent SMD which can be processed by our recognition system later on and recognized as intended. Moreover, as described in chapter 4, we can extract one or several Levenshtein expressions from the action's regular expression. We then select one of them and start generating a minimal SMD from an input starting point $p_0 = (x_0, y_0, z_0, \theta_0, \phi_0, \gamma_0) = (\vec{c_0}, \vec{q_0})$ by converting the successive Levenshtein expression tokens $L = T_1...T_n$ into curve segments. The main idea

is to generate $m = \frac{t_{tot*F_d}}{n*\alpha}$ curve segments per token from the local frame of the last point inserted in the curve (α is the Segmenter sensitivity and F_d the recording device frequency). Thus, to generate the i^{th} point p_i of the minimal SMD we compute the local direction $\vec{D} = (\vec{v}, \vec{\omega})$ of the current read token and assign $p_i = p_{i-1} + (R_z(\gamma_{i-1})R_y(\phi_{i-1})R_x(\theta_{i-1})\vec{v}, \vec{\omega})$. Computing $\vec{D}$ is done by ensuring that the generated curve segment will be recognized as the currently processed token. The following list shows several examples of tokens and their corresponding directions:

- $T\_X \rightarrow \vec{D} = 2\epsilon(1, 0, 0, 0, 0, 0)$
- $R\_N\_X \rightarrow \vec{D} = 2\epsilon(0, 0, 0, -1, 0, 0)$
- $T\_N\_X\_Y\_Z \rightarrow \vec{D} = 2\epsilon(-1, 1, 1, 0, 0, 0)$
- $R\_X\_N\_Y\_Z \rightarrow \vec{D} = 2\epsilon(0, 0, 0, 1, -1, 1)$

where ϵ is the minimum displacement our system can recognize. At the end of the process, we generated mn curve segments which can be recognized as the L token sequence by the Segmenter.

Reaching a specific target. At the end of the minimal SMD generation process, users may add a target reaching constraint such that actions like Jump or Walk end at a specific location. To comply with such a constraint we first align the normalized $\overrightarrow{c_1c_n}$ vector with the normalized $\overrightarrow{c_1c_{goal}}$ vector by rotating all p_i with $i \in [2 : n]$ by $dot(\overrightarrow{c_1c_n}, \overrightarrow{c_1c_{goal}})$ with respect to the axis $(c_1, cross(\overrightarrow{c_1c_n}, \overrightarrow{c_1c_{goal}})$. Then, in order force $p_n = p_{goal}$ in a coherent manner we scale every point with respect to p_1 by $a = \frac{\|c_{goal} - c_1\|}{\|c_n - c_1\|}$ meaning that:

$$\forall i \in [2 : n], c_i = c_1 + a(c_i - c_1) \tag{6.8}$$

Finally we assign q_n to q_{goal}. It is important to note that as our recognition system is rotation and scale invariant, this process guarantees that the SMD is still recognized as intended.

Optimization Smoothing. Finally, we propose an additional smoothing step for actions which follow the SMD transferred trajectory, preventing sudden character turns. Indeed, when rotating the whole curve at the previous stage we actually introduce an orientation discontinuity between the first and second points of the output SMD curve. In addition, the same kind of discontinuity can occur between the last and second-to-last point of the curve. To cope with this issue we propose to smooth the orientations in a as-rigid-as-possible manner while removing orientation discontinuities at the beginning and end of the curve. We address this issue through an optimization process very similar

to the one described in section 6.2, with two additional discontinuity removal terms, minimizing:

$$L_i(q_1, ..., q_n) = \sum_j \left\| (\alpha_j^{init}, \beta_j^{init}) - (\alpha_j, \beta_j) \right\|^2 + \gamma_1 \|q_1 - q_2\|^2 + \gamma_2 \|q_n - q_{n-1}\|^2 \tag{6.9}$$

with all $(\alpha_j^{init}, \beta_j^{init})$ computed from the initial $(q_1, ..., q_n)$ orientation curve and $\gamma_1 = \gamma_2 = \frac{n}{2}$ in our case.

Figure 6.11 shows an example of SMD generated from a sequence of action regular expressions.

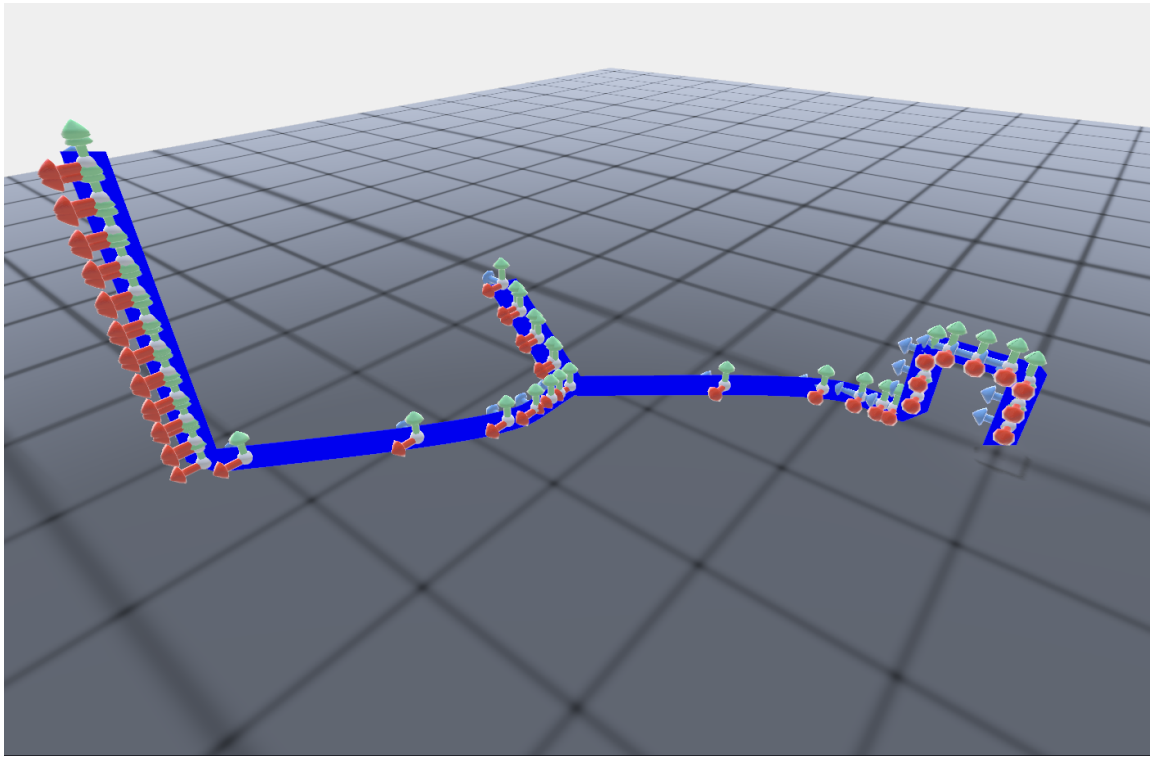


Figure 6.11: Procedurally generated SMD of a Jump, Walk, Kick, Side Step, Raise Arms sequence.

6.5 Interaction experimental results

We tested our multi-character animated stories authoring tool in several simple examples where two characters interact. For each example, the user first trained the system to use his own gestures for action and interaction recognition. Due to lack of time we could not integrate the contact preserving constraint into our performance transfer pipeline although it was tested as a side module. Therefore all transferred interactions come from well adapted animations which are assumed to be synchronized.

We tested our system in a small scenario where a Witch deceives a Knight, claiming to be a princess that will be transformed back if the Knight kisses her. The Knight, acting like a fairy tail hero, kisses her but feel disgusted as the Witch is not transforming back into a princess. She finally laugh at him for being deceived. This scene is actually part of the scenario we devised in [Barbulescu et al., 2017] as use case of the FIGURINES system. The scene we reproduced is depicted in figure 6.12 and shows the detection and transfer of the Kiss interaction modeled as: $(Kiss : (T\_N\_Z) + (T\_Z) + (T\_N\_Z) + (T\_Z)+, Kiss : (T\_N\_Z) + (T\_Z) + (T\_N\_Z) + (T\_Z)+)$, with $C = (D(0, 1, Intimate), V(0, 1, 0.5), V(0, 1, 0.5))$ as constraint set.

This first example shows that our system can successfully recognize an interaction and correctly transfer it under the assumption that the underlying animations are well calibrated.

To further test our interaction recognition and transfer system, we started to devise multiple scenarios using Commedia Dell'Arte characters. With their unique personality, morphology and way of executing actions, we started to build tools to create complex and diverse scenarios. The characters, animations as well as the corresponding 3D printed figurines used for these examples were done by a professional artist specifically for our needs. While they are currently only able to Walk, Jump, Slap and be Slapped we tested our system on simple interaction cases, demonstrating promising results. Note that, due to lack of time, we could not use the 3D printed figurines instrumented with the HTC Vive trackers.

Figure 6.13 shows Colombine walking towards Pierrot, then Slapping him (who react with a Slapped animation) before going away from him. Figure 6.14 shows Arlequin walking and jumping towards Pantalone, then they exchange slaps before Pantalone get frustrated and leaves the stage. In those two examples the Slap/Slapped interaction is modeled as: $(Slap : (T\_N\_Z) + (T\_Z), Slapped : T\_STILL+)$, with $C = (D(0, 1, Personal), V(0, 1, 0.5), V(1, 0, 0.5))$ as constraint set.

With only few actions and one interaction at our disposal, those Commedia Dell'Arte performance transfer results are encouraging and suggest that developing the action and interaction vocabularies for those characters is a good direction to follow.

Overall, these experiments show that our system can be used at a greater level of storytelling by incorporating true interactions between two characters.

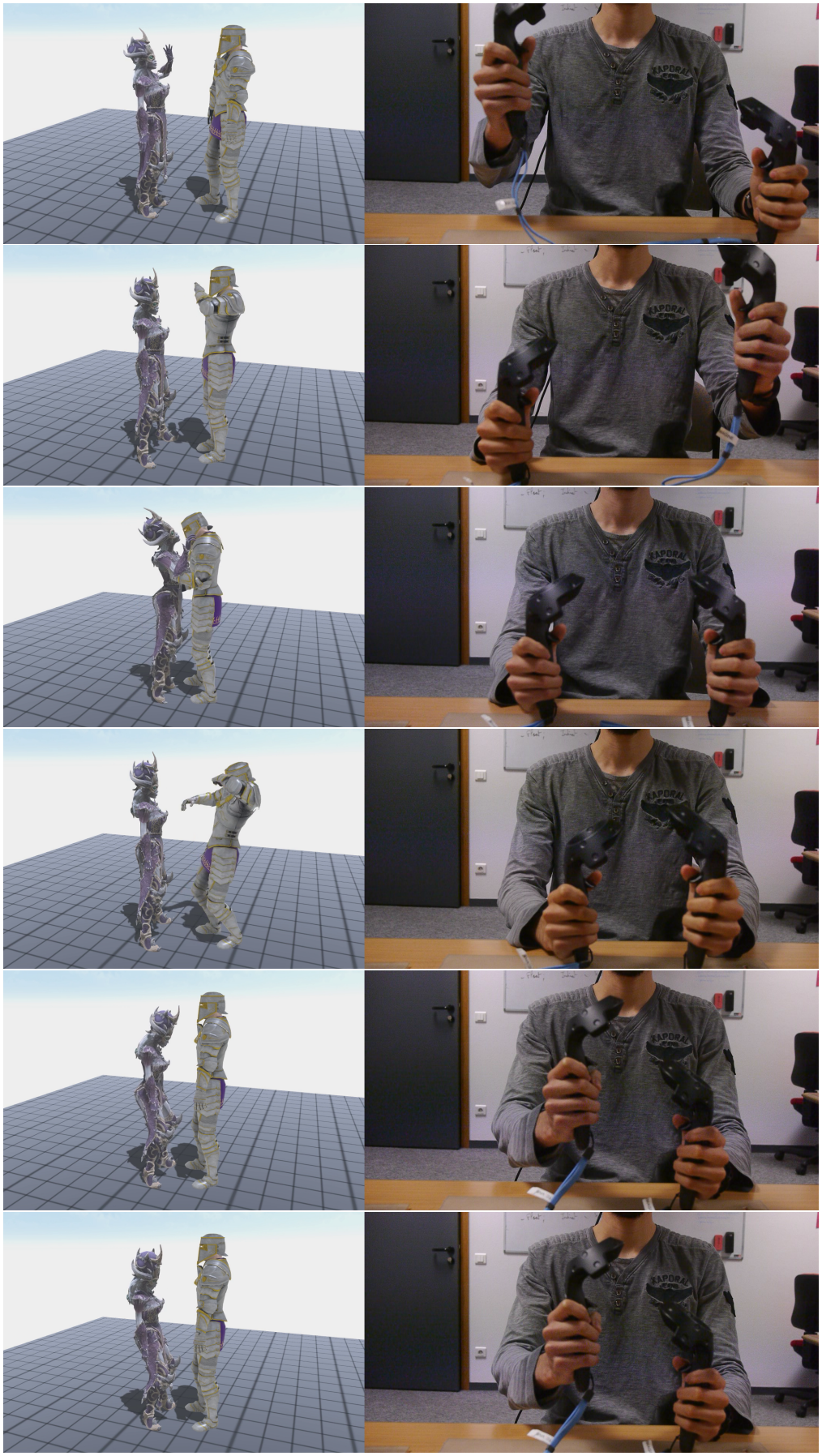


Figure 6.12: Knight and Witch scene: in this scenario the user imagined that the knight has been deceived by the witch who told him that she is actually a princess and that he has to kiss her in order the transform her back. The action sequence is: Greet (Witch), Greet (Knight), Kiss (interaction), Disgust (Knight), Laugh (Witch)

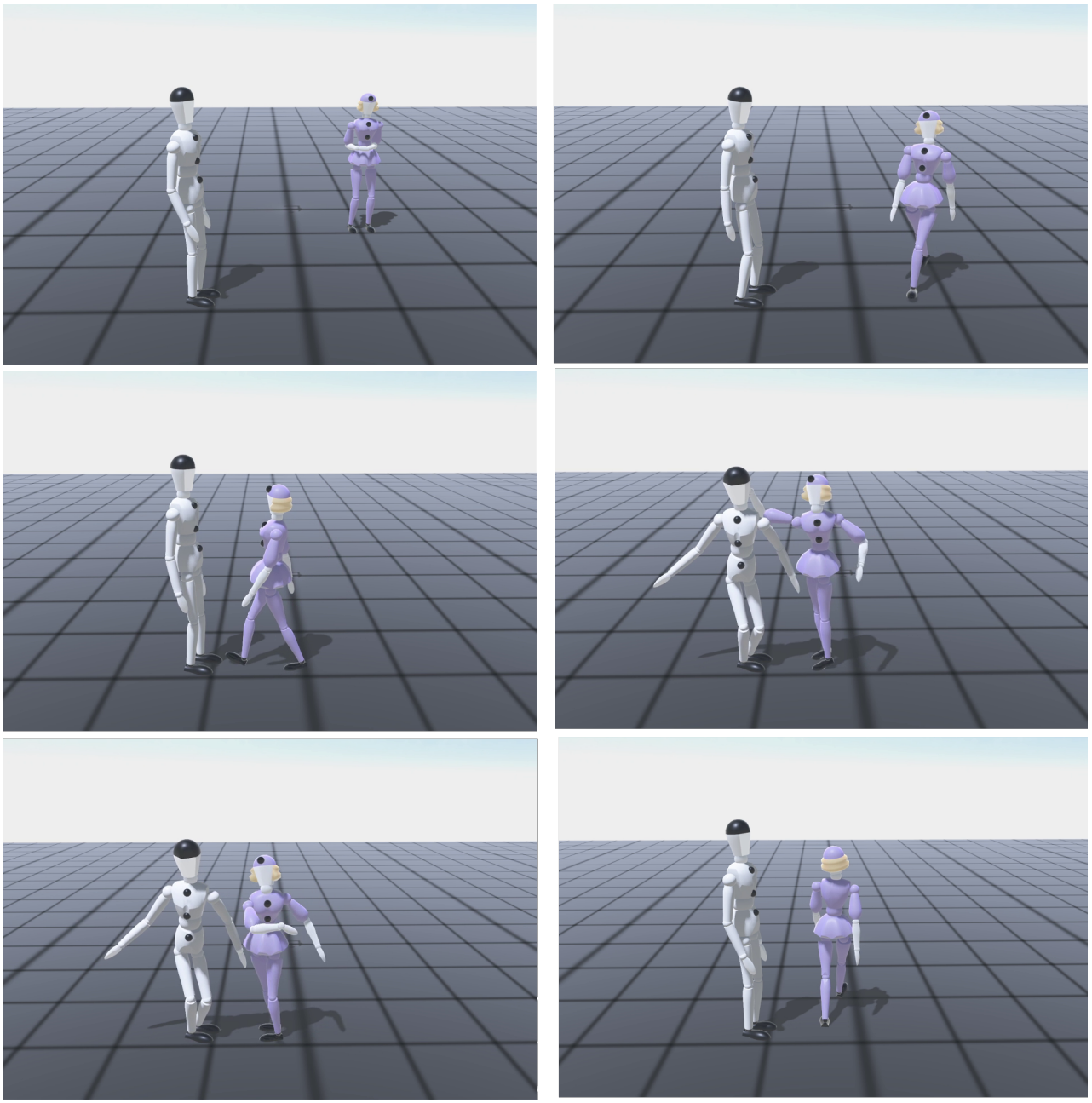


Figure 6.13: Pierrot and Colombine Slap scene. Colombine walks toward Pierrot and Slap him before going away. This example shows interaction detection and transfer for the Slap/Slapped actions.

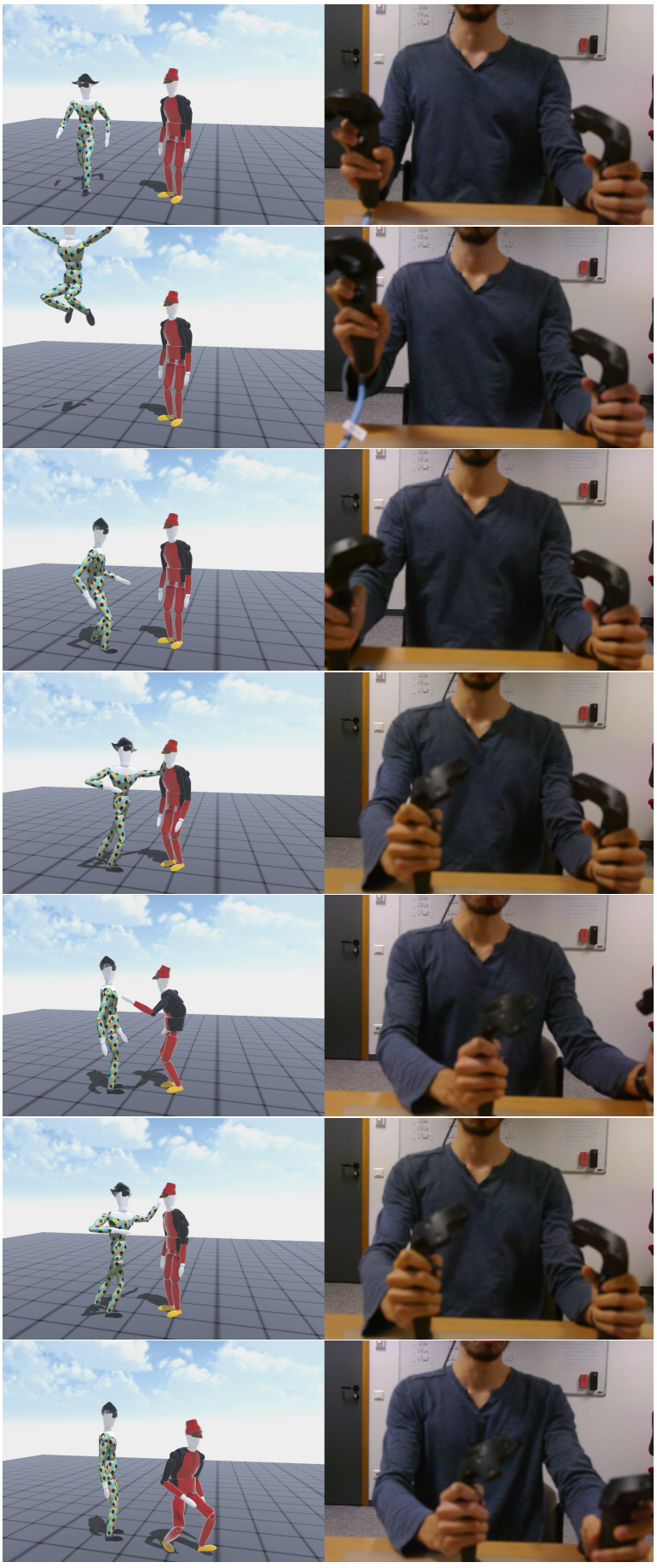


Figure 6.14: Pantalone and Arlequin slape scene. In this scene Arlequin Walks and Jumps towards Pantalone before slapping each other. Eventually, Pantalone, frustrated, leaves the stage.

While having been tested on quite short examples, we emphasize that this feature greatly enhance the narrative aspect of the animation users can produce with our system.

6.6 Conclusion

In this chapter, we detailed how we extended the performance transfer algorithm presented in chapter 5 to let users create stories with two interacting characters. We introduced a new model for representing interactions and additional recognition constraints such as character distances and behavior-based constraints. We also described two methods to transfer detected interactions while preserving important contacts in the animations. The first method adds implicit actions to correctly place characters before they perform an interaction stored inside a single animation. The second method automatically modifies independent character animations, adapting them to different character morphologies and placement.

The experiments we performed using two characters at the same time also highlighted the importance of the physical to virtual world mapping. When playing without any virtual display that can be provided by the HTC Vive head mounted display, it is very important to preserve a 1:1 geometry mapping, meaning that the corresponding 3D character and environment models should have the same size as the controllers and the physical playground. Indeed, this mapping guarantees that interactions and character placements with respect to the physical playground will be correctly transferred as users intend. More precisely, if character sizes are not preserved, distances between virtual characters are transformed and it can compromise the interaction recognition process as distance constraints might not be respected when they should. This mapping can be changed when using the virtual display offered by the HTC Vive which is a useful feature letting users play in differently scaled environments. However it might prevent them to make characters touch each other if the characters are smaller in the virtual display.

While we tested the newly introduced features on simple examples, we augmented our initial system to propose a more complete authoring tool that can be used for creating diverse animated stories using at least two characters.

Due to lack of time, we did not properly evaluate the interaction recognition rate and transfer. We also did not properly study the motion quality recognition and transfer in the context of two hands performances. These tasks are left to future work and should be performed in diverse scenarios in order to

show the narrative extent of our system. It would also be useful to include children in the users studies as they might react differently to our system and have a better affinity to create stories. In addition, in the same manner as we presented contact preserving constraint it would be very useful to integrate animation time constraints like proposed by [Kim et al., 2009] to also synchronize character animations during interactions.

Chapter 7

Conclusion and Perspectives

Creating 3D character animations is a difficult process which requires a lot of iterations and a good understanding of living beings motions. That is why this task is usually done by expert animators who have spent years of studying and training. Research work aiming at easing this creation process has been done through different approaches either using sketches, physical props or its own body to author animations. While exploring various directions to accomplish their goals, relatively few works tackle the challenge of proposing systems and methods to intuitively create full-body character animation sequences. Additionally, the current proposed tools do not let users easily and quickly create complex sequences, especially in the case of multi-character scenarios.

Hence, this thesis proposes a way to allow non-experts to create animated stories while playing with tangible objects such as figurines, letting them produce character animation sequences of their narration. Moreover, our system is able to recognize character actions and interactions as well as motion qualities from user hand motions and transform these trajectories into a sequence of stylized animations. Adding context and behavior-based constraints, this thesis provides new tools for drafting complex and expressive animated stories. The efficiency and usefulness of our system has been evaluated through different users studies and has shown its potential in the field of scene staging, video games and animation layout drafting.

In this work, we introduced a new hand motions action and interaction recognition algorithm which is translation, rotation and scale invariant. In particular, these properties allow users to execute action gestures from any starting orientation and position at their preferred amplitude. We also devised and evaluated an action learning algorithm based on user provided training examples, letting them use their own gesture when recording.

We studied recognition of hand motion qualities as Laban Efforts and pro-

posed a new method to infer them in "neutral" animations based on animation decomposition into five stages. Laban Effort classifiers were scrupulously evaluated through different experiments. Laban animation modifiers were evaluated through subjective studies and compared to a baseline stylization method, showing the efficiency of our method.

Finally, we designed a new performance transfer algorithm sequencing recognized actions and interactions into multi-character animations, in which their root motion is automatically extracted from the user hand motion trajectories and in which smooth animation transitions are automatically computed. Furthermore, we also showed how to combine this process with additional contact preserving and behavior-based constraints enhancing the plausibility of the output animations.

While answering several of the challenges of our performance transfer goal, many tasks remain to recreate the introduction scene of Toy's Story 3 from figurines play. First, the output animations we produce with our system can be improved and rendered even more expressive. Moreover, it would be interesting to merge the output facial animations of the FIGURINES system with the output full-body animations of our performance transfer algorithm. We also did not take into account the interaction with the environment which is a research topic by itself; exploring the formalism described in narrative system like [Marti et al., 2018] (smart objects) could be one way to tackle this problem. Our system also does not compute automatically narrative camera placements (which has already been investigated by works like [Galvane et al., 2014]) nor let users author them; the same could be said regarding lighting. Eventually, our current system does not contain any post edition process which can be highly desirable.

This work opens up several direction of research and perspectives:

Real-Time Application. Modifying the current algorithm such that the performance transfer is done in real-time (with a small delay due to the current performed action) is straight forward as computation costs afford more than 60 fps on laptop devices. Furthermore, it would be interesting to adapt our method for real-time application such as video games or virtual reality painting systems which introduces new challenges especially in terms of trajectory transfer.

Level of Control. During this thesis we focused on mapping the hands motion trajectory to the characters root motion while the characters perform recognized action. In future studies, it would be interesting to also map the

user hands and head motion to one of the character's head and hands joint motion while allowing users to switch between controlling full body motions with SMD or acting one of the character animations. Likewise, this process could also be applied to the face motion like we did in our FIGURINE framework. More generally, it would be interesting to allow users to animate all the elements of the scene including the camera. In addition, we would also like to investigate the use of voice to trigger actions instead of performing recognizable action patterns, easing the creation process but requiring that we can precisely determine the beginning and end of each action.

Children user studies. We could not make user studies with children during this thesis, but this is clearly a direction to follow as we want to build a animation authoring system which is within their reach and which recreates the story they imagine with figurines. In their case, using voice to recognize actions instead of making them perform patterns might be more appropriated as it requires focus and attention children might not have depending on their age.

Music and Animation. While investigating animation stylization and gesture transfer, we noticed that some research work [Shiratori et al., 2006], [Kuen-Meau et al., 2008] studied the similarity and more precisely the duality between Laban Effort and music in the context of character animation. In our case, we especially tried to relate Legato and Staccato to Sustained and Sudden motions and Piano and Forte to Light and Strong motions. We did short preliminary studies trying to figure out if playing appropriated background music with stylized animation amplifies the feeling of Effort variations. Preliminary results clearly showed a potential to pursue such studies. Going further, it could be very interesting to author animations and stylization using music as input.

Animation from script. In this thesis, we showed how actions could be represented as regular expressions of motion tokens and translated into adapted animations. We extended this concept to character interactions and introduced contextual constraints necessary to trigger both actions and interactions so that users can create consistent stories. The whole action recognition and transfer system behaves like a compiler building a low level language into animation sequences. Like we hinted in chapter 6, our system could be used as a base to build a tool transforming a higher level representation of the user's story like clauses and sentences into SMDs before being translated to animation sequences. Some works like [Won et al., 2014] or [Hyun et al., 2016] introduced methods to transform scripts expressed in a high level grammars to multi-character animations. However, these works are applied in specific contexts and do not take the envi-

ronment into account. In the future, it would be very interesting to tackle the challenging task of generalizing these application fields using our action representation as an intermediate language and propose a system that interprets an entire storytelling grammar into animation sequences.

Bibliography

[Abdul-Massih et al., 2016] Abdul-Massih, M., Yoo, I., and Benes, B. (2016). Motion style retargeting to characters with different morphologies. *Computer Graphics Forum*.

[Agrawal and van de Panne, 2016] Agrawal, S. and van de Panne, M. (2016). Task-based locomotion. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*.

[Amaya et al., 1996] Amaya, K., Bruderlin, A., and Calvert, T. (1996). Emotion from motion. In *Proceedings of the Conference on Graphics Interface '96*, GI '96, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.

[Aristidou and Lasenby, 2011] Aristidou, A. and Lasenby, J. (2011). Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*.

[Aristidou et al., 2017] Aristidou, A., Zeng, Q., Stavrakis, E., Yin, K., Cohen-Or, D., Chrysanthou, Y., and Chen, B. (2017). Emotion control of unstructured dance movements. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17.

[Barbulescu et al., 2017] Barbulescu, A., Begault, A., Boissieux, L., Cani, M.-P., Garcia, M., Portaz, M., Viand, A., Heinish, P., Dulery, R., Ronfard, R., and Vaufreydaz, D. (2017). Making Movies from Make-Believe Games. In *6th Workshop on Intelligent Cinematography and Editing (WICED 2017)*, Lyon, France. The Eurographics Association.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*.

[Bessmeltsev et al., 2016] Bessmeltsev, M., Vining, N., and Sheffer, A. (2016). Gesture3d: Posing 3d characters via gesture drawings. *ACM Trans. Graph.*

[Bishko, 2014] Bishko, L. (2014). Animation principles and laban movement analysis: Movement frameworks for creating empathic character performances. In Tanenbaum, J., El-Nasr, M. S., and Nixon, M., editors, *Nonverbal Communication in Virtual Worlds*, Pittsburgh, PA, USA. ETC Press.

[Bouchard and Badler, 2007] Bouchard, D. and Badler, N. (2007). Semantic segmentation of motion capture using laban movement analysis. In *Proceedings of the 7th International Conference on Intelligent Virtual Agents*.

[Bribiesca, 2007] Bribiesca, E. (2007). Classification and generation of 3d discrete curves.

[Camurri et al., 2003] Camurri, A., Mazzarino, B., Ricchetti, M., Timmers, R., and Volpe, G. (2003). Multimodal analysis of expressive gesture in music and dance performances. In *Gesture-Based Communication in Human-Computer Interaction. Lecture Notes in Computer Science, 2915*.

[Cerezo et al., 2015] Cerezo, E., Marco, J., and Baldassarri, S. (2015). Hybrid games: Designing tangible interfaces for very young children and children with special needs. In *Gaming Media and Social Effects*.

[Chetverikov, 2003] Chetverikov, D. (2003). A simple and efficient algorithm for detection of high curvature points in planar curves. *Lecture Notes in Computer Science*.

[Chi et al., 2000] Chi, D., Costa, M., Zhao, L., and Badler, N. (2000). The EMOTE Model for Effort and Shape. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co.

[Choi et al., 2016] Choi, B., i Ribera, R. B., Lewis, J. P., Seol, Y., Hong, S., Eom, H., Jung, S., and Noh, J. (2016). SketchiMo: Sketch-based Motion Editing for Articulated Characters. *ACM Trans. Graph.*

[Choi et al., 2012] Choi, M. G., Yang, K., Igarashi, T., Mitani, J., and Lee, J. (2012). Retrieval and Visualization of Human Motion Data via Stick Figures. *Comput. Graph. Forum.*

[Ciccone et al., 2017] Ciccone, L., Guay, M., Nitti, M., and Sumner, R. W. (2017). Authoring Motion Cycles. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17.

[Ciccone et al., 2019] Ciccone, L., Öztireli, C., and Sumner, R. W. (2019). Tangent-space optimization for interactive animation control. *ACM Trans. Graph.*

[Desai et al., 2019] Desai, R., Anderson, F., Matejka, J., Coros, S., McCann, J., Fitzmaurice, G. W., and Grossman, T. (2019). Geppetto: Enabling semantic design of expressive robot behaviors. In *CHI'19*.

[Dontcheva et al., 2003] Dontcheva, M., Yngve, G., and Popović, Z. (2003). Layered acting for character animation. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, New York, NY, USA. ACM.

[Durupinar et al., 2016] Durupinar, F., Kapadia, M., Deutsch, S., Neff, M., and Badler, N. I. (2016). PERFORM: Perceptual Approach for Adding OCEAN Personality to Human Motion Using Laban Movement Analysis. *ACM Trans. Graph.*

[Ekman, 2003] Ekman, P. (2003). Emotions revealed: Recognizing faces and feelings to improve communication and emotional life.

[Fdili Alaoui et al., 2017] Fdili Alaoui, S., Françoise, J., Schiphorst, T., Studd, K., and Bevilacqua, F. (2017). Seeing, sensing and recognizing laban movement qualities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17.

[Furukawa et al., 2014] Furukawa, M., Akagi, Y., Kawai, Y., and Kawasaki, H. (2014). Interactive 3d animation creation and viewing system based on motion graph and pose estimation method. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14.

[Galvane et al., 2014] Galvane, Q., Ronfard, R., Christie, M., and Szilas, N. (2014). Narrative-Driven Camera Control for Cinematic Replay of Computer Games. In *MIG'14 - 7th International Conference on Motion in Games* , Los Angeles, United States. ACM.

[Guay et al., 2013] Guay, M., Cani, M.-P., and Ronfard, R. (2013). The Line of Action: An Intuitive Interface for Expressive Character Posing. *ACM Trans. Graph.*

[Guay et al., 2015] Guay, M., Ronfard, R., Gleicher, M., and Cani, M.-P. (2015). Space-time Sketching of Character Animation. *ACM Trans. Graph.*

[Gupta et al., 2014a] Gupta, A., Agrawala, M., Curless, B., and Cohen, M. (2014a). Motionmontage: A system to annotate and combine motion takes for 3d animations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, New York, NY, USA. ACM.

[Gupta et al., 2014b] Gupta, S., Jang, S., and Ramani, K. (2014b). Puppetx: A framework for gestural interactions with user constructed playthings. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, AVI '14.

[Hall, 1968] Hall, E. T. (1968). Proxemics. *Current Anthropology.*

[Hayes-Roth and van Gent, 1997] Hayes-Roth, B. and van Gent, R. (1997). Story-Making with Improvisational Puppets. In *Proceedings of the first international conference on Autonomous agents (AGENTS '97).*

[Heck and Gleicher, 2007] Heck, R. and Gleicher, M. (2007). Parametric motion graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07.

[Held et al., 2012] Held, R., Gupta, A., Curless, B., and Agrawala, M. (2012). 3d Puppetry: A Kinect-based Interface for 3d Animation. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12. ACM.

[Holden et al., 2017] Holden, D., Komura, T., and Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Transactions on Graphics*.

[Hyun et al., 2016] Hyun, K., Lee, K., and Lee, J. (2016). Motion grammars for character animation. *Comput. Graph. Forum*.

[Ishii and Ullmer, 1998] Ishii, H. and Ullmer, B. (1998). Tangible bits: Towards seamless interfaces between people, bits and atoms. *Conference on Human Factors in Computing Systems - Proceedings*.

[Jacobson et al., 2014] Jacobson, A., Panozzo, D., Glauser, O., Pradalier, C., Hilliges, O., and Sorkine-Hornung, O. (2014). Tangible and modular input device for character articulation. *ACM Trans. Graph.*

[Kapadia et al., 2016] Kapadia, M., Frey, S., Shoulson, A., Sumner, R. W., and Gross, M. (2016). CANVAS: Computer-Assisted Narrative Animation Synthesis. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16. Eurographics.

[Kapadia et al., 2011] Kapadia, M., Singh, S., Reinman, G., and Faloutsos, P. (2011). A behavior authoring framework for multi-actor simulations. In *IEEE Computer Graphics and Applications*.

[Kastanis and Slater, 2012] Kastanis, I. and Slater, M. (2012). Reinforcement learning utilizes proxemics: An avatar learns to manipulate the position of people in immersive virtual reality. *ACM Trans. Appl. Percept.*

[Kim et al., 2012] Kim, M., Hwang, Y., Hyun, K., and Lee, J. (2012). Tiling motion patches. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, Goslar Germany, Germany. Eurographics Association.

[Kim et al., 2009] Kim, M., Hyun, K., Kim, J., and Lee, J. (2009). Synchronized multi-character motion editing. *ACM Trans. Graph.*

[Kovar et al., 2002] Kovar, L., Gleicher, M., and Pighin, F. (2002). Motion Graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02. ACM.

[Kuen-Meau et al., 2008] Kuen-Meau, C., Siu-Tsen, S., and Stephen D., P. (2008). Using music and motion analysis to construct 3d animations and visualisations. *Digital Creativity*.

[Laban, 1950] Laban, R. (1950). *The mastery of movement*. Macdonald & Evans, London.

[Lamberti et al., 2019] Lamberti, F., Gatteschi, V., Sanna, A., and Cannavò, A. (2019). A multimodal interface for virtual character animation based on live performance and natural language processing. *International Journal of Human–Computer Interaction*.

[LaViola and Keefe, 2011] LaViola, J. J. and Keefe, D. F. (2011). 3d spatial interaction: Applications for art, design, and science. In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11.

[Leite and Orvalho, 2012] Leite, L. and Orvalho, V. (2012). Shape your body: control a virtual silhouette using body motion. In *CHI Extended Abstracts*.

[Liu et al., 2006] Liu, C. K., Hertzmann, A., and Popović, Z. (2006). Composition of complex optimal multi-character motions. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Llobera et al., 2010] Llobera, J., Spanlang, B., Ruffini, G., and Slater, M. (2010). Proxemics with multiple dynamic characters in an immersive virtual environment. *ACM Trans. Appl. Percept.*

[Lu et al., 2011] Lu, F., Tian, F., Jiang, Y., Cao, X., Luo, W., Li, G., Zhang, X., Dai, G., and Wang, H. (2011). Shadowstory: creative and collaborative digital storytelling inspired by cultural heritage. In *CHI*.

[Marti et al., 2018] Marti, M., Vieli, J., Witon, W., Sanghrajka, R., Wotruba, D., Simo, I., Schriber, S., Kapadia, M., and Gross, M. (2018). Cardinal: Computer assisted authoring of movie scripts. In *23rd International Conference on Intelligent User Interfaces*.

[Masuda et al., 2010] Masuda, M., Kato, S., and Itoh, H. (2010). A laban-based approach to emotional motion rendering for human-robot interaction.

[Mayora et al., 2009] Mayora, O., Costa, C., and Papliatseyeu, A. (2009). itheater puppets: Tangible interactions for storytelling. In *International Conference on Intelligent Technologies for Interactive Entertainment*. Springer.

[McCann and Pollard, 2007] McCann, J. and Pollard, N. (2007). Responsive characters from motion fragments. *ACM Trans. Graph.*

[Mentis and Johansson, 2013] Mentis, H. and Johansson, C. (2013). Seeing movement qualities. In *Conference on Human Factors in Computing Systems. Proceedings.*

[Miura et al., 2014] Miura, T., Kaiga, T., Shibata, T., Katsura, H., Tajima, K., and Tamamoto, H. (2014). A hybrid approach to keyframe extraction from motion capture data using curve simplification and principal component analysis. *IEEE Transactions on Electrical and Electronic Engineering.*

[Mori et al., 2005] Mori, G., Belongie, S., and Malik, J. (2005). Efficient shape matching using shape contexts. *IEEE transactions on pattern analysis and machine intelligence.*

[Möller and Trumbore, 1997] Möller, T. and Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1).

[Neff and Fiume, 2005] Neff, M. and Fiume, E. (2005). Aer: aesthetic exploration and refinement for expressive character animation. In *Symposium on Computer Animation.*

[Nitsche and McBride, 2018] Nitsche, M. and McBride, P. (2018). A character in your hand. puppetry to inform game controls. In *DiGRA'18 - Proceedings of the 2018 DiGRA International Conference: The Game is the Message.*

[Novakovic, 2010] Novakovic, J. (2010). The impact of feature selection on the accuracy of bayes classifier. In *18th Telecommunications forum TELFOR.*

[Oore et al., 2002] Oore, S., Terzopoulos, D., and Hinton, G. (2002). A desktop input device and interface for interactive 3d character animation. In *Graphics Interface.*

[Perlin and Goldberg, 1996] Perlin, K. and Goldberg, A. (1996). Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques.*

[Pink and Ramanathan, 2000] Pink, T. H. and Ramanathan, U. (2000). Intelligent selection of useful features for optimal feature-based classification. In *IGARSS 2000. IEEE 2000 International Geoscience and Remote Sensing Symposium.*

[Poirier and Paquette, 2009] Poirier, M. and Paquette, E. (2009). Rig retargeting for 3d animation. In *Proceedings of Graphics Interface 2009*, GI '09.

[Pollick et al., 2009] Pollick, F. E., Maoz, U., Handzel, A. A., Giblin, P. J., Sapiro, G., and Flash, T. (2009). Three-dimensional arm movements at constant equi-affine speed. *Cortex.* Special Issue on Cognitive Neuroscience of Drawing.

[Pollick and Sapiro, 1997] Pollick, F. E. and Sapiro, G. (1997). Constant affine velocity predicts the 1/3 power law of planar motion perception and generation. *Vision Research*.

[Portaz et al., 2017] Portaz, M., Garcia, M., Barbulescu, A., Begault, A., Boissieux, L., Cani, M.-P., Ronfard, R., and Vaufreydaz, D. (2017). Figurines, a multimodal framework for tangible storytelling. In *WOCCI 2017 - 6th Workshop on Child Computer Interaction at ICMI 2017 - 19th ACM International Conference on Multi-modal Interaction*, Glasgow, United Kingdom. Author version.

[Ramer, 1972] Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*.

[Roberts et al., 2019] Roberts, R., Lewis, J. P., Anjyo, K., Seo, J., and Seol, Y. (2019). Optimal and interactive keyframe selection for motion capture. *Computational Visual Media*.

[Rose et al., 1998] Rose, C., Cohen, M. F., and Bodenheimer, B. (1998). Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18.

[Russell, 1980] Russell, J. A. (1980). A circumplex model of affect. *Journal of Personality and Social Psychology*.

[Safonova and Hodgins, 2007] Safonova, A. and Hodgins, J. K. (2007). Construction and optimal search of interpolated motion graphs. *ACM Trans. Graphics, Proceedings of Siggraph*.

[Sanz et al., 2015] Sanz, F. A., Olivier, A., Bruder, G., Pettré, J., and Lécuyer, A. (2015). Virtual proxemics: Locomotion in the presence of obstacles in large immersive projection environments. In *2015 IEEE Virtual Reality (VR)*.

[Savitzky and Golay, 1964] Savitzky, A. and Golay, M. J. E. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. In *Analytical Chemistry*.

[Schulz and Mihov, 2002] Schulz, K. U. and Mihov, S. (2002). Fast string correction with levenshtein automata. *International Journal of Document Analysis and Recognition*.

[Senecal et al., 2016] Senecal, S., Cuel, L., Aristidou, A., and Magnenat-Thalmann, N. (2016). Continuous body emotion recognition system during theater performances. *Comput. Animat. Virtual Worlds*.

[Shiratori et al., 2006] Shiratori, T., Nakazawa, A., and Ikeuchi, K. (2006). Dancing-to-music character animation. *Computer Graphics Forum*.

[Shoulson et al., 2013] Shoulson, A., L. Gilbert, M., Kapadia, M., and Badler, N. (2013). An event-centric planning approach for dynamic real-time narrative. In *Proceedings - Motion in Games 2013, MIG 2013*.

[Shoulson et al., 2014] Shoulson, A., Marshak, N., Kapadia, M., and Badler, N. (2014). Adapt: The agent development and prototyping testbed. In *IEEE Transactions on Visualization and Computer Graphics*, volume 99.

[Slyper et al., 2015] Slyper, R., Hoffman, G., and Shamir, A. (2015). Mirror puppeteering: Animating toy robots in front of a webcam. In *Tangible and Embedded Interaction*.

[Smith et al., 2019] Smith, H. J., Cao, C., Neff, M., and Wang, Y. (2019). Efficient neural networks for real-time motion style transfer. *Proc. ACM Comput. Graph. Interact. Tech.*

[Sreenivasa et al., 2009] Sreenivasa, M. N., Soueres, P., Laumond, J., and Berthoz, A. (2009). Steering a humanoid robot by its head. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Su et al., 2007] Su, W.-P., Pham, B., and Wardhani, A. (2007). Personality and emotion-based high-level control of affective story characters. *IEEE transactions on visualization and computer graphics*.

[Terra and Metoyer, 2004] Terra, S. C. L. and Metoyer, R. A. (2004). Performance timing for keyframe animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04.

[Thomas and Johnston, 1981] Thomas, F. and Johnston, O. (1981). The illusion of life: Disney animation. Disney Editions.

[Thompson, 1968] Thompson, K. (1968). Programming techniques: Regular expression search algorithm. *Commun. ACM*.

[Thorne et al., 2004] Thorne, M., Burke, D., and van de Panne, M. (2004). Motion Doodles: An Interface for Sketching Character Motion. SIGGRAPH '04. ACM.

[Van De Panne, 1997] Van De Panne, M. (1997). From Footprints to Animation. *Computer Graphics Forum*.

[Van Zaanen, 2000] Van Zaanen, M. (2000). ABL: Alignment-based Learning. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 2*, COLING '00, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Won et al., 2014] Won, J., Lee, K., O'Sullivan, C., Hodgins, J. K., and Lee, J. (2014). Generating and ranking diverse multi-character interactions. *ACM Trans. Graph.*

[Xia et al., 2015] Xia, S., Wang, C., Chai, J., and Hodgins, J. (2015). Realtime Style Transfer for Unlabeled Heterogeneous Human Motion. *ACM Trans. Graph.*

[Yumer and Mitra, 2016] Yumer, M. E. and Mitra, N. J. (2016). Spectral style transfer for human motion between independent actions. *ACM Trans. Graph.*

[Ziola et al., 2011] Ziola, R., Grampurohit, S., Landes, N., Fogarty, J., and Harrison, B. (2011). Examining interaction with general-purpose object recognition in lego oasis. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).*

Appendix

Appendix 1: Implementation

This work has been implemented in two different open source frameworks available at the following link `https://team.inria.fr/imagine/maxime-garcia/`. Moreover, every animation has been generated inside the SkyEngine, a game engine I developed during this thesis with colleagues and former co-workers.

This game engine is based on a Entity Component System (ECS), which is a common software architecture among modern game engines like Unity 3D or Unreal Engine 4, and makes use of modern C++ 17 features, allowing users to easily create their own plug-ins. While this project is still under development, it already features a Physically Based Rendering engine with a modulable shading pipeline, a complete Animation system which can be used both for real-time applications or for creating scripted animated scenes, a Physic module based on the Bullet physic engine, a Mesh and Trajectory manipulation module and a Scene manipulation module. This engine also aims at implementing recent research works which ease the 3D content creation process, being for modeling or animating. Most of the renderings of this thesis were directly done inside the SkyEngine real-time renderer.

Working with this environment allowed me to modify and sequence animations on the fly which was not possible to do with Unity 3D at the time I started my thesis (which is not the case anymore). In addition, we also implemented several research algorithm such as the FABRIK IK solver [Aristidou and Lasenby, 2011] with additional pole vector constraints which was used for several processes of this work. The Math module of this engine also contains a simple optimization problem solver which was used for minimizing cost functions. Finally, Python binding are half way integrated but I could already use this feature for classifying action curve segments into the Laban Effort qualities. I personally learned a lot in graphics programming while developing this software and plan to continue extending and maintaining it for the next years to come.

The code used for action/interaction recognition and Laban Effort Classification is contained in the second framework and is split between a C++ library for action recognition and Naive Bayes classification and Python scripts for HMM-based classification using the *hmm-learn* library.

The whole performance transfer software/hardware pipeline is described in figure 7.1.

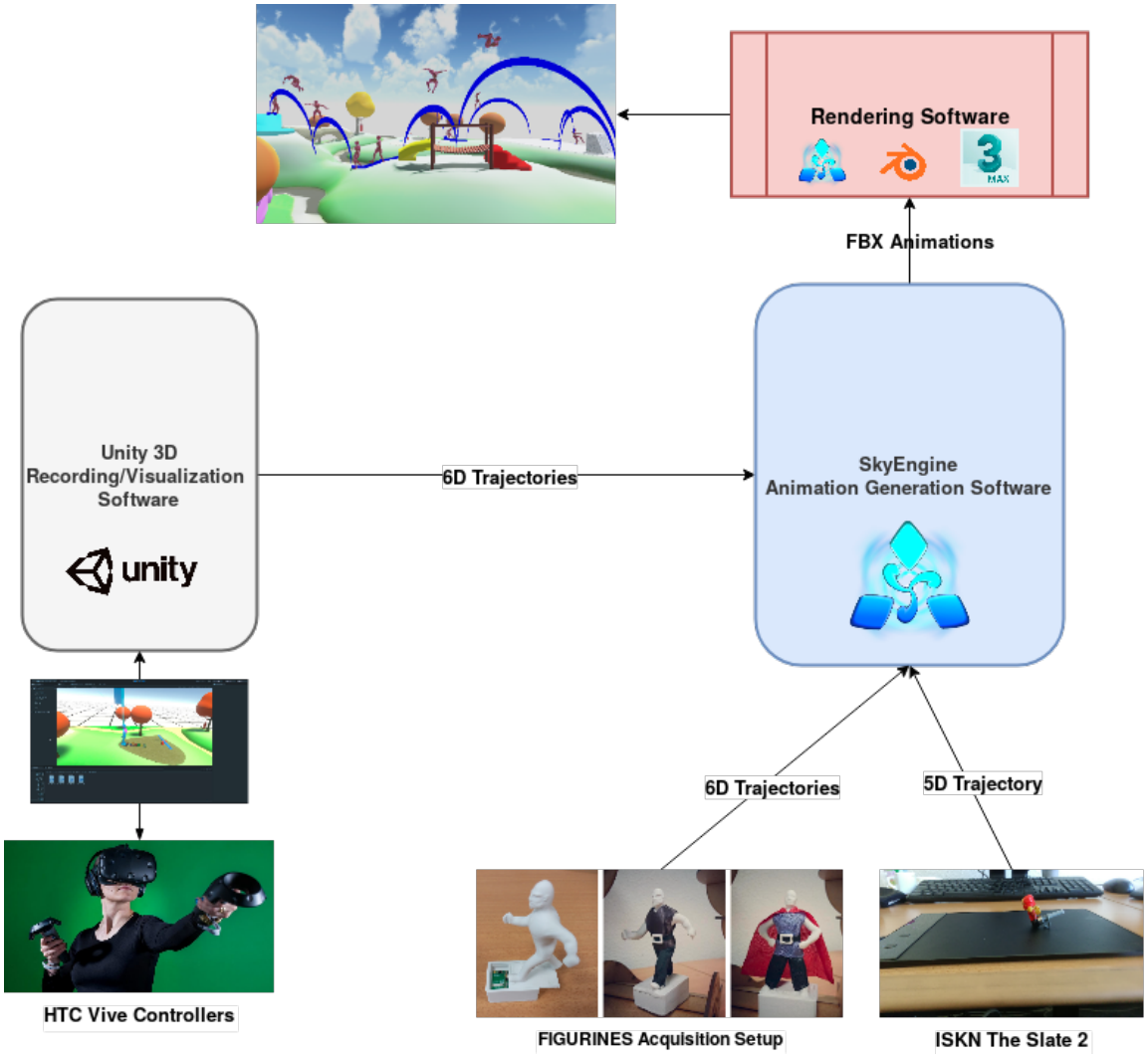
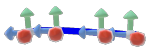
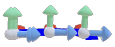
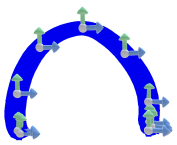
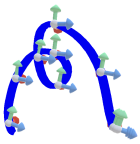
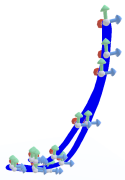
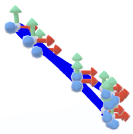
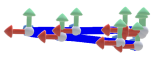
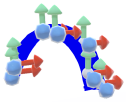
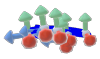
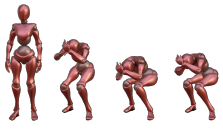
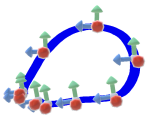
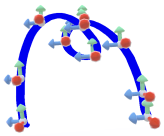
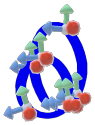
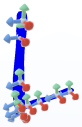
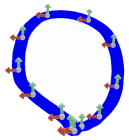


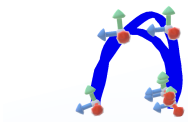
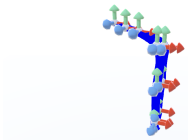
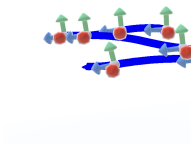
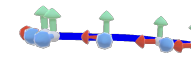
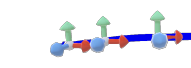
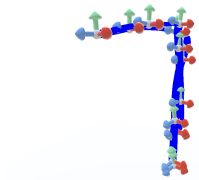
Figure 7.1: Performance transfer software/hardware workflow. Our current system can receive trajectory data from 3 different setups: our FIGURINES acquisition setup, the Slate 2 (by ISKN) and the HTC Vive controllers (and trackers). We use Unity 3D as the acquisition software receiving 6D trajectories from the HTC Vive controllers (and trackers) which also display a virtual environment in which the user can see the virtual version of the controllers presented as rigid 3d models. The FIGURINES setup 6D trajectories are recorded using a stand alone application while the Slate 2 is directly interfaced with the SkyEngine. When receiving trajectories, our main SkyEngine application processes them and produces output animations that can be directly rendered inside the same application (real-time) or exported toward rendering software like 3DS Max or Blender for more realistic renderings.

Appendix 2: Animation Database

Action	Gesture	Regular Expr	Animation
Walk		$(T\_Z\|T\_X\_Z\|T\_N\_X\_Z)+$	
Walk Back		$(T\_N\_Z\|T\_X\_N\_Z\|T\_N\_X\_N\_Z)+$	
Jump		$((T\_Z)*(T\_N\_Z)*)(T\_Y)+$ $((T\_Y\_Z)*(T\_Z)*(T\_N\_Y\_Z)*)$ $(T\_N\_Y\|T\_Y\_Z)+(T\_Z)*(T\_N\_Y\_Z)*(T\_N\_Y)*$	
Flip		$((T\_X\_N\_Z)*(T\_Y\_N\_Z)*)(T\_Y\_N\_Z\|T\_Y)+$ $((T\_Y\_Z)*\|(T\_Y\_Z)*(T\_Z)*$ $(T\_N\_Y\_Z)*(T\_N\_Y)*(T\_N\_Y\_N\_Z)*)(T\_Y\|T\_Z)+$ $((T\_N\_Y\_Z)*(T\_N\_Y)*(T\_N\_Z)*$ $(T\_Y\_N\_Z)*\|(T\_Z)*(T\_N\_Y\_Z)*\|(T\_Y\_Z)*)$ $(T\_Z\|T\_Y\|T\_N\_Y)+$ $((T\_Z)*\|(T\_N\_Y\_Z)*)(T\_N\_Y\|T\_N\_Y\_Z)*$ $(T\_N\_Y\_N\_Z\|T\_N\_Y)*((T\_N\_Z)*(T\_Y\_N\_Z)*)$ $(T\_Y)*(T\_Y\_Z)*(T\_Z)*(T\_N\_Y\_Z)*(T\_N\_Y)*$	
Kick		$(T\_N\_Z)+(T\_Z)+((T\_Y\_Z)*)$ $(T\_Y\|T\_Y\_Z)+(T\_N\_Y\|T\_Y)+$ $((T\_N\_Y\_N\_Z)*(T\_N\_Y)*)(T\_N\_Y\|T\_N\_Y\_N\_Z)+$ $(T\_N\_Y\_N\_Z)*(T\_N\_Z)*$	
Block		$((T\_Z)*)(T\_N\_X\_Y)+((T\_Y)*)$ $(T\_N\_Y\|T\_X\_N\_Y)+((T\_X\_N\_Y)*(T\_X\_Z)*(T\_X)*)$ $(T\_X)*((T\_X\_N\_Y)*)(T\_X)*$	
Raise One Arm		$(T\_X)+(T\_N\_X)+$	

Action	Gesture	Regular Expr	Animation
Raise Arms		$(T\_Y) + (T\_N\_Y) + (T\_Y) + (T\_N\_Y)+$	
Despair		$((T\_X) * \|(T\_X)*)(T\_N\_G) + (T\_N\_G\|T\_N\_X)*$	
Get Angry		$(T\_N\_X) + (T\_X) + (T\_N\_X)+$	
Dodge		$(T\_X\_Y\|T\_Y) + ((T\_N\_X\_N\_Y) * (T\_N\_Y)*)$ $(T\_Y\|T\_N\_X\_Y) + ((T\_X\_Y) * (T\_X) * \|(T\_Y\_Z)*)$ $(T\_N\_X\|T\_N\_G\_B\|T\_N\_Y)+$ $(T\_N\_X\_N\_Y) * ((T\_N\_Y) * (T\_Y)*)$ $(T\_X\_Y) * (T\_X) * (T\_X\_N\_Y) * (T\_N\_Y)*$	
Dismiss		$(T\_Z) + (T\_N\_Z) + (T\_Z) + (T\_N\_Z) + (T\_Z)*$	
Duck		$(T\_Y) + ((T\_N\_X\_Y)*)(T\_N\_X\|T\_N\_Y)+$ $((T\_Z) * (T\_X)*)(T\_N\_X\_N\_Y\|T\_Y)*$ $(T\_N\_Y) * (T\_Y) * (T\_X\_Y) * (T\_X) + (T\_X\_N\_Y) * (T\_N\_Y)*$	

Action	Gesture	Regular Expr	Animation
Cast Fireball		$((T\_N\_Y\_N\_Z)*)(T\_N\_Z)+$ $((T\_Y\_N\_Z)*\|(T\_Y)*(T\_Z)*\|(T\_Y\_N\_Z)*)$ $(T\_N\_Y\_Z\|T\_Y)+((T\_Y\_Z)*)(T\_Y\_Z\|T\_Z)*$ $(T\_Z\|T\_N\_Y\_Z)*(T\_N\_Y\_Z)*(T\_N\_Y)*)$	
Back Flip		$(T\_Y)+((T\_Y\_N\_Z)+)((T\_N\_Z)+)((T\_N\_Y\_N\_Z)*)$ $((T\_N\_Y)*)((T\_Z)*)$ $((T\_Y)+)((T\_Y\_N\_Z)+)$ $((T\_N\_Z)+)((T\_N\_Y\_N\_Z)+)((T\_N\_Y)+)$	
Yell		$(T\_Y)+(T\_N\_Y)+(T\_Y)+(T\_N\_Y)+(T\_Y)+(T\_N\_Y)+$ $((T\_Y\_N\_Z)*(T\_N\_Y\_Z)*(T\_N\_Y)*)$	
Uppercut Jab		$((T\_Z)+)((T\_Y\_Z)+)((T\_Y)+)((T\_Y\_N\_Z)+)$ $((T\_N\_Z)*)((T\_N\_Y\_N\_Z)*)((T\_N\_Y)+)$ $((T\_N\_Y\_N\_Z)*)((T\_N\_Z)*)((T\_Z)*)$ $((T\_Y)+)((T\_Z)*)((T\_Y\_N\_Z)+)((T\_N\_Z)+)$ $((T\_N\_Y\_N\_Z)+)((T\_N\_Y)+)((T\_Y)+\|(T\_N\_Y\_Z)+)$	
Stop		$(T\_N\_Z)+(T\_Z)+(T\_Y)+(T\_N\_Y)+$	
Stomp		$((T\_N\_X)*)(T\_Y)+(T\_N\_Y)+$	
Crescent Kick		$((T\_N\_X\_Y)*\|(T\_N\_X\_N\_Z)*(T\_N\_X)*(T\_N\_X\_Y)*)$ $(T\_Y\_Z\|T\_N\_X)+((T\_X\_Z)*(T\_X)*)$ $(T\_N\_Y\_N\_Z\|T\_N\_X\_Y)+(T\_N\_Y\|T\_Y\_Z)+$ $((T\_Y)*\|(T\_X\_Y\_Z)*)(T\_X\_Z\|T\_X\_Y)*$ $(T\_X)*(T\_X\_N\_Y)*$ $((T\_N\_Y)*(T\_N\_Y\_N\_Z)*)(T\_N\_Y)*(T\_N\_Y\_N\_Z)*$ $((T\_N\_X\_N\_Y\_N\_Z)*)$ $(T\_N\_X\_N\_Y\_N\_Z\|T\_N\_X\_N\_Y)*$ $((T\_N\_Y\_N\_Z)*(T\_N\_Z)*)(T\_N\_Y)*$	

Action	Gesture	Regular Expr	Animation
Pick up		$(T\_Y) + ((T\_Y\_Z) * (T\_Z) * (T\_N\_Y\_Z) * \|(T\_Y\_Z)*)$ $(T\_Z\|T\_N\_Y) + (T\_N\_Y\_Z\|T\_Y)+$ $((T\_Y\_N\_Z)*)(T\_N\_Y\|T\_N\_Z)+$ $((T\_N\_Y\_N\_Z)*)(T\_Y\|T\_N\_Y) + (T\_Y\_N\_Z) * (T\_N\_Z)*$ $(T\_N\_Y\_N\_Z) * (T\_N\_Y)*$	
Greet		$(T\_Y) + ((T\_N\_X\_Y)*)(T\_N\_X) + (T\_X)+$ $((T\_X\_N\_Y)*)(T\_N\_Y)+$	
Punch		$((T\_X\_Z)*)(T\_N\_Z) + (T\_Z) + (T\_N\_Z)+$	
Side Walk Right		$(T\_N\_X)+$	
Side Walk Left		$(T\_X)+$	
Throw		$(T\_Y) + (T\_Z) + (T\_N\_Z) + ((T\_Y\_N\_Z)*)$ $(T\_N\_Y) + ((T\_N\_X\_N\_Y) * (T\_N\_Y\_Z)*)$	

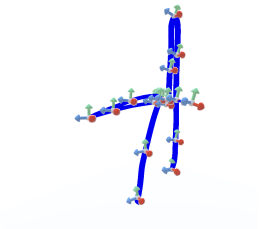
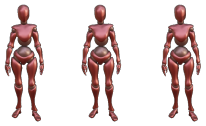
Action	Gesture	Regular Expr	Animation
Dance		$(T\_Y) + (T\_Z) + (T\_N\_Z)+$ $(T\_X) + (T\_N\_X) + (T\_X) + ((T\_N\_X\_N\_Z)*)(T\_Y) + (T\_N\_Y)+$	
Idle		$(T\_STILL)+$	

Table 7.1: Actions of our library, among the 28 actions 16 of them were added by users for the experiments. The corresponding regular expressions were learned from user provided examples.