MASTER THESIS:

MOTION ENRICHING USING HUMANOIDE CAPTURED MOTIONS

STUDENT: SINAN MUTLU
ADVISOR: ANTONIO SUSÌN SÀNCHEZ
SEPTEMBER, 8TH 2010

COURSE: MASTER IN COMPUTING
LSI DEPERTMANT
POLYTECNIC UNIVERSITY OF CATALUNYA

Abstract

Animated humanoid characters are a delight to watch. Nowadays they are extensively used in simulators. In military applications animated characters are used for training soldiers, in medical they are used for studying to detect the problems in the joints of a patient, moreover they can be used for instructing people for an event(such as weather forecasts or giving a lecture in virtual environment). In addition to these environments computer games and 3D animation movies are taking the benefit of animated characters to be more realistic. For all of these mediums motion capture data has a great impact because of its speed and robustness and the ability to capture various motions.
Motion capture method can be reused to blend various motion styles. Furthermore we can generate more motions from a single motion data by processing each joint data individually if a motion is cyclic. If the motion is cyclic it is highly probable that each joint is defined by combinations of different signals. On the other hand, irrespective of method selected, creating animation by hand is a time consuming and costly process for people who are working in the art side. For these reasons we can use the databases which are open to everyone such as Computer Graphics Laboratory of Carnegie Mellon University.

Creating a new motion from scratch by hand by using some spatial tools (such as 3DS Max, Maya, Natural Motion Endorphin or Blender) or by reusing motion captured data has some difficulties. Irrespective of the motion type selected to be animated (cartoonish, caricaturist or very realistic) human beings are natural experts on any kind of motion. Since we are experienced with other peoples' motions, and comparing each motion to the others, we can easily judge one individual's mood from his/her body language. As being a natural master of human motions it is very difficult to convince people by a humanoid character's animation since the recreated motions can include some unnatural artifacts (such as foot-skating, flickering of a joint).

Our work is focused to reuse the original motion by the following ways:

- Changing path of a single motion to get another motion which moves on a different path.

- Analyzing each joint's motion by signal processing and by changing coefficients of each signal for defining a different motion.
- Blending a walking/running character with different moods (such as sad, happy or relaxed).

It is possible that all the methods mentioned above end up with some artifacts such as foot-skating. To overcome this problem we proceed in our work to find a way for clearing unnatural parts of a motion by:

- Detecting the foot planted times of a moving character.
- Clearing the foot-skates by using Analytic Inverse Kinematic library.

We implemented all of these by benefiting from the asf/amc viewer of Computer Graphics Laboratory of Carnegie Mellon University which was able to import only one character.

Keywords: computer graphics, computer animation, mocap, motion capture, motion editing, inverse kinematics, motion graphs, quaternions, blending, Fourier Transform, signal processing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

In the most general sense, *"animate"* means "give life to" and includes live-actions puppetry such as that found on *Sesame Street*. In the early days of the animation history before the era of cinema and John von Neumann computer architecture people invented some tools to create some basic animations. With the arise of today's computers in digital medium 3D artists and researchers start to produce animated characters and process motion data, which we are used to see in computer games digital animation movies and in strategic simulators.

Today one of the biggest challenges in computer games, film industry and all other fields in which humanoid motion has a place to visualize, is creating animated character from scratch by using some spatial tools or reusing the existing motions by processing them. Apart of being very realistic or cartoonist results, recreated/processed humanoid animations have to be believable. Since human observers are very sensitive to motion as been natural experts, it demands tremendous skills to satisfy them.

There are different ways of creating humanoid animations such as procedural animation (using ragdoll physics and forces to move the character), key-frame animation (giving the positions of a trajectory for some key time intervals and then interpolating these positions for undefined times), and motion capture (mocap – recording the movements of a real object/human by using some devices). Among these examples of methods mocap has become more popular since its robustness, easy to capture realistic motions with different styles of motion.

Whatever method is selected, it is still challenging to create a motion, for example if our character has to pass from one type of motion to another such as from walking to running or from straight walk to sudden turns to left/right or back. In these transitions it

is difficult to keep the continuity of joints' movements. Another example is: if a motion data which goes forward and if we need same motion (depressed walk) following another path, it will be waste of time to design it from scratch. For this kind of example in low level animator has to select appropriate key-frames in the existing motions to construct a new one. Then he/she has to smooth the new generated one. Since this process gets a lot of time it can be a bottleneck in development of the applications.

1.1 Goals

In this study we tried to develop an environment which is processing the existing motions (mocap files in asf/amc format) by using various techniques to get a new motion. The techniques we used for this purpose are:

a) **Analyzing motion with signal processing methods**: If a motion is cyclic, it gives the chance to analyze each joint data using the techniques in signal processing. In a normal cyclic walking mocap data for each degree of freedom (DOF) of a joint we can produce other behaviors for that joint. For instance if we consider one DOF data of a joint as an original signal, most probably this is the sum of other signals. Therefore by the help of Fourier analysis behaviour of that joint can be changed. As a result of different filters motion of a joint can be exaggerated as well as the fact that it can be smoothed. Or changing the coefficients of existing signals in the original one will define another motion. Furthermore Fourier analysis and autocorrelation methods can be used to detect the period of the cyclic motion.

b) **Motion Path Editing:** A path is an abstraction of positional movement of a character, which is different from, but related to, the orientation of the character as it moves along the path. This abstraction leads to the idea of representing a motion relative to the path, allowing the path to be altered and the motion to be adjusted accordingly (*M.Gleicher*). For instance, a character can move in a straight line, so by editing this path it is possible to make the character to move in a curved path by preserving the detail and nuance of the original motion. Dancing characters can be another good

example for this method. With this method it is possible to change the path of very complicated motion data by changing the positions of control points. Applying this method can cause some visual problems such as foot-skating. If the distance between two consecutive control points is not preserved, we can end up with foot-skating problems. At this point inverse kinematics (IK) algorithms can be applied to overcome this problem. Actually in literature there are various methods which aimed to correct this kind of problems by using analytic inverse kinematics.

c) **Motion Blending:** When we have different motions for the character, we want to make a transition from one type of motion to another type, or to make a combination of motions by blending two or more of them with different ratios. By this method it is possible produce procedural animations, for instance if we have an aggressive walking animation and if we blend this motion with sadness we can have sad-aggressive motion. Or if we have a normal walk and another hand waving motion, we can blend the upper bodies of these two motions to get a walking-hand waving motion. For blending the motions we have to know/detect the foot stances, if we try to blend two motions when one of them is on the left foot stance and the other one is on the right stance we can have an unnatural motion.

d) **Motion Transition:** A motion transition is also using blending. The difference lies in the fact that transition is temporarily a blend. It will be used to pass from one motion to a new one. To detect visually good results for this problem there is a tremendous research in this field emerged in last decade. In the case of path editing method, if the two consecutive control points are moved too much, analytic inverse kinematic algorithms may not succeed to give satisfying results because the distances can be very far for the legs to reach to the new constrained end points. To fill such discontinuities motion graphs can be used. In this method we also need to detect the foot plant intervals to avoid bad wrong transitions such as from left stance to left stance.

e) **Foot Plant Position Detecting:** *Foot plant* occurs when the character strikes on the ground. The foot continues to contact and be stationary on the ground for the duration of foot plant. This is important to get a visually convincing results from the processing methods mentioned above. On the other hand ignoring foot plants during motion blending, motion transition and path editing will produce visible artifacts and unnatural behavior.

CHAPTER II

BACKGROUND

The state of the art in humanoid character animation in 3D dates back to more than twenty years ago, on the other hand, the history of animation started more than one century ago. In this chapter we will start with a short literature overview of animation and then we will review related research in character animation.

2.1 Early Days of Animation

Persistence of vision and ability to interpret a series of stills as moving image were invented in the 1800s[1]. Perhaps the simplest device designed to animate figures is *Thuamatrope,* a flat disk with images drawn on both sides and having two strings connected opposite each other on the rim of the disks. Other best known device is *Zeatrope*, or *wheel of life*. It has a short flat cylinder that rotates on its axis of symmetry. Around the inside of the cylinder there is a sequence of drawings each one slightly different from the ones next to it. The cylinder has long vertical slits cut into its side between each adjacent pair of images so when it is stun on its axis each slit allows the eye to see the image on the opposite wall of the cylinder. In the last decades of the 19th century devices like *"Magic Latterns"* (an image projector powered by candle or lamp) became very popular for moving images [2]. Then, in 1888 in Edison Lab *"Kinetograph"* was developed mostly by William Kennedy and Laurie Dickson. It was an innovative motion picture camera with rapid intermittent, or stop-and-go, film movement, to photograph movies for in-house experiments and, eventually, commercial Kinetoscope presentations [3]. The earliest use of camera goes back to 1896, George Mèliès accidently discovered the stop trick, then he used stop-motion technique to make objects appear, disappear, and change shape[4][25]. His well known film is *A Trip to the Moon*. After him in 1900 J. Stuard Blackton animated "*smoke*" in a scene and made his first animation cartoon *Humorous Phases of Funny Faces* (1906) [5]. The first animated character with an identifiable personality was *Flex the Cat*, drawn by Otto Messemmer of Pat Sullivan's studios (1919) [6]. Towards the end of 1920s Walt Disney

came into stage. He advanced the animation as an art form. He started to use storyboard to review the story and pencil sketches for reviewing the motion. Following these he started to use colors and sound in his movies. To advance his work he studied live-actions for making the animations more realistic [7]. In U.S in the following years other animation techniques also tried such as clay and puppet animation, a close comparison can be made with computer animation. Some examples of this method are "*Wallace and Gromit"* and "*Nightmare Before Christmas"*. In 1989 Pixar made first computer animation movie *"Tin Toy"* which was the first computer animation movie awarded with the Academy Award [8].

Figure 2.1: Thuamatrope, Zeatrope and Kinetoscope

2.2 Character Animation

Creating an animated character involves modeling the characters geometry and creating specific motions for the character [9][10]. The visible geometry of the skin can be created by a variety of techniques, differing primarily in the skin detail and the degree of representation of underlying internal structures such as bone and muscles. Geometry for hair and clothing can be simulated with a clear trade-off between accuracy and computational complexity. Different methods can be applied for light interaction with skin and clothes depending on visual requirements [11], [24**]**.

In the recent years many approaches have been developed for creating and representing the geometry of a virtual human's body. Thanks to the developments so far in computer graphics and computer animation, geometric modeling became mature ([12], [13]). Animators have wide range of private/open source tools for preparing the whole

animation and geometry of a character [14], [15], [16], [17]. In some of these tools it is also possible to prepare a dynamic humanoid character animation [18], [19].

Traditional animation also referred as hand-drawn animation is the oldest and most popular form of animation. In this type of the animations animators were used to draw each single frame by hand. Nowadays computer assisted animation replaced all of the manual production line of traditional animation. With the increased power computation and advances in computer graphics, it became possible to increase the realism and the motion editing and reusing existing data.

Creating character animation involves a lot of steps, such as modeling the geometry, shading the character, defining humanlike motions…etc. All of these steps are costly and requires high skills. Therefore in these days one of the biggest challenges is developing algorithms for increasing the reusability of the existing animations.

In the literature there are few basic methods for creating a character animation such as: Key frame, motion capture animation, behavioral animation and procedural animation. Each of these methods can combine other methods for decreasing the cost of character animation production. While the combined methods allow rich control to the animators, the single methods lacks the reusability and control.

2.2.1 Key Frame Animation

This type of animation mimics the hand-drawn animation. We give the system the extreme positions and rotations of an object, in the case of the character animation, poses are defined for some of the key frames. Then in-between key frames are calculated using spline interpolations [20].

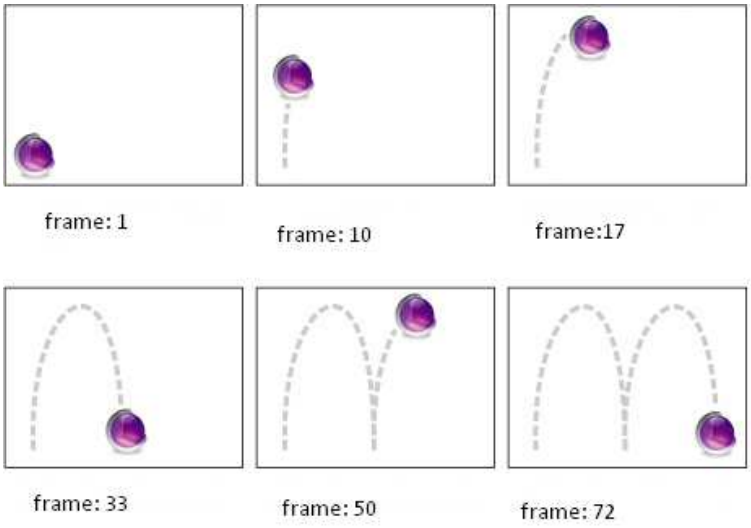


Figure 2.2: Defining main key frames

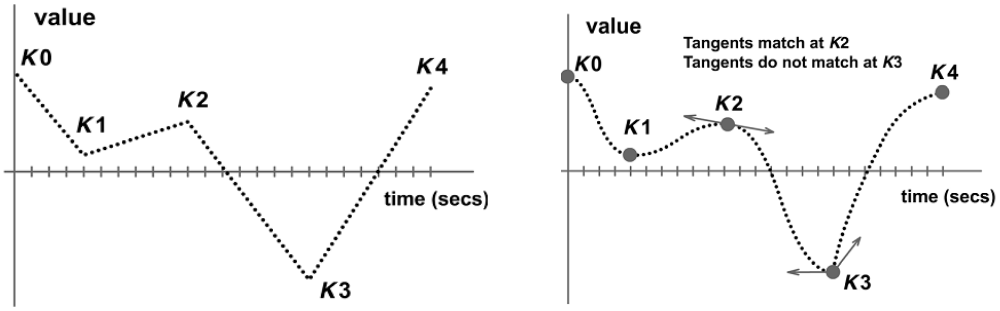


Figure 2.3: Linear interpolation (left), cubic spline interpolation (right)

If we apply linear interpolation to our main key frames to calculate the in-between frames we will have a motion like on the left diagram on the above figure. In this method between any key frames, *K2 and K3*, the value of a point *P* at time *t* can be found by: *P = K2.value +((t-K2.time)/(K2.time-K1.time))*(K2.value – K1.value)*

It is obvious that the result is not satisfying. For this reason cubic spline curves give better results. But still there are problems in using splines. Since not all of them are preserving the first order continuity (C1) an appropriate type of them has to be selected. For instance Hermite curves can give good results for C1 continuity. Taking into account all of these we can say that key framing offers more control while sometimes it can be overwhelming with increasing number of DOF [21]. In this sense this method requires high level experience and skills from the animators.

2.2.2 Motion Capture Animation

Recording and a mapping of a motion of a real object to a syntactic object is called *motion capture*. It involves sensing digitizing and recording of an object's motion, not its visual appearance [22]. Using this method has some advantages such that it is rapid and even real time, the amount of work does not vary with complexity or length of the performance, the amount of animation data that can be created within a given time is extremely large when compared to traditional techniques, complex movements such as physical interactions can be recorded. Conversely, by using other techniques, such as key-framing, inverse kinematics and forward kinematics, trying to get the same result which we can get from motion capture is frustrating. This method is mimicking the key-framing technique since the motion of an actor is sampled many times for a second. Still it has some difficulties and disadvantages, such as, specific hardware and special programs are required to obtain and process the data, the cost of the software, equipment and personnel requirement can potentially be prohibitive for small productions, the capture system may have specific requirements for the space it is operated in, depending on camera field of view or magnetic distortion. These are just some of the disadvantages of these systems.

In this technology, there are two approaches used so far: electromagnetic sensors and optical trackers. In the former type of approach sensors are placed at each joint that they are transmitting their positions and orientations back to central processor to record their movements. For doing this, sensors have to use cables or wireless transmitters. With some latency the orientation and positions of sensors can be rendered, on the other hand the drawbacks relate to the range and accuracy of magnetic field and the restricted movements. Optical markers have much larger range and performers only have to wear reflective markers on their clothes. But this approach does not provide real time feedback, also being error prone (occluded markers) and being noisy are the other drawbacks. In recent years there are some approaches for reducing the number of optical markers and using inverse kinematics for detecting the position and orientation of the joints which are not represented by markers [23].

Still there are a lot of researches for making motion capture systems more accurate and cheaper. But thanks to groups who make their data free to access, researchers can create their own databases by processing these pre-recorded animation data.

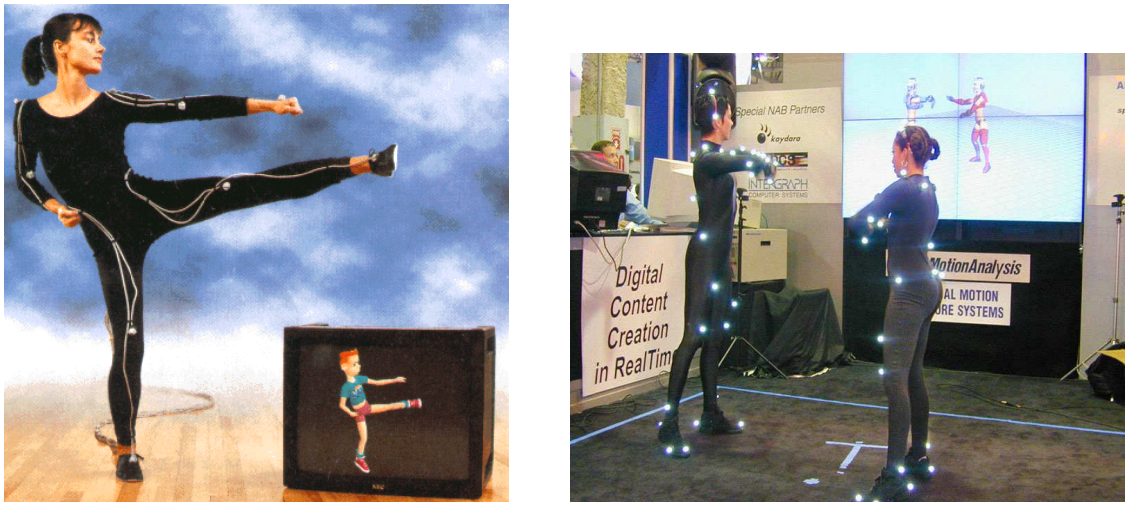


Figure 2.3: Mocap with electromagnetic sensors (left), with optical markers (right)

2.2.3 Procedural Animation

Procedural Animation refers to generation of motion based on some sort of procedures rather than being pre-processed. It is a pretty huge subject and contains many different techniques. Animation is produced either by running a physically based simulator or finding the poses of an animated character considering some geometric constraints. In the former concept different results can be obtained by changing the forces which are applied on the animated character. In the latter concept it is possible to get a variety of results with changed speed, acceleration and positions of the joints. Using these methods gives less degree of control to the animator compare to the key-framing technique but on the other hand it is much faster method for creating animations. Depending on the solvers and constraints we defined realism can be improved.

In the literature for creating some locomotion models some motions models are used. Here we will describe a few of them:

a) Dynamic Simulation: In *physically based animation* forces are applied to each limb of the character that they have different mass values. Some forces may not relate to physics at all-they may model constraints that animator may wish to enforce on

the motion. There are two main types of dynamic animated character control: *forward dynamics control* and *inverse dynamics control*. The former method requires the forces and torque to be specified. For instance these forces can be applied if an event or stage is triggered, such as if the system detects maximum forward extension of a leg, it will change the forces and torques which are applied so far on the character's leg [26], [28].

The latter method, *inverse dynamics,* introduces calculating the forces and torques to make a trajectory following a path [27], [18], [19]. To model the Newtonian physics and momentum rules can be used. For this kind of animation Simbicon [27] is one of the best examples. In Simbicon walking locomotion and running locomotion are represented as a finite state machine. Each state defines different poses of the character (left stance, right stance), during the simulation by applying torques on each joint, character is getting the next state's pose after a while. For each state there is a defined time interval that simulated character has to go from one state to the next state, actually these time intervals define speed of the animated character. If the state finished earlier, system triggers the next state. In Simbicon rotating the joint/trajectories are done by proportional derivative controller: $\tau = Kp(\theta d - \theta) - kd^* \theta$, which is the other type of virtual springs. Here *"θd"* is the desired angle and "*θ"* is the current angle value of the joint. But if we just apply these forces, there is no guarantee to have a balanced character /ragdoll, for this reason a balance feedback is added into the system. $\theta d = \theta d0 + Cd^*D + Cv^*V$, is defining the balance feedback for the swing foot replacement. Where *"*θ*d"* is the target angle used for proportional derivative (PD) at any time, *"θd0"* is the default fixed target as described in the finite state machine (FSM), *"D"* is the horizontal distance from the stance ankle to the centre of mass (COM), and *"V"* is the velocity of the COM. *"Cd and Cv"* are gain parameters which have to be tuned by hand. For us this part is the drawback of the system, since the animator should fit these values. These values are depending to the simulation environment (type of the terrain) and type of the character motion (normal walk, drunk walk, running). Mimicking the motion capture data is one of the advantages of the system. If the mocap data is periodic a FSM can be constructed, so this data can be imported into the system. Most of the full dynamic systems still have poor locomotion results. They seem like more robotic movements instead of human or cartoon character movements.

Another example to this kind of systems is DANCE [19]. This open source software has its own PD controller. We can define the key poses for the ragdoll by importing a

mocap file or by using the inverse kinematic framework, after using these key poses we can prepare a state machine. For example jumping from a building, or protecting face from a coming object are a bit complicated simulations that just needs a few key poses in DANCE. To move from one pose to another system uses spring damper model. There are some simulations that can be done easily; on the other hand simulations such as walking locomotion, running locomotion or continuous movements are difficult since there is no internal feedback controller [29].

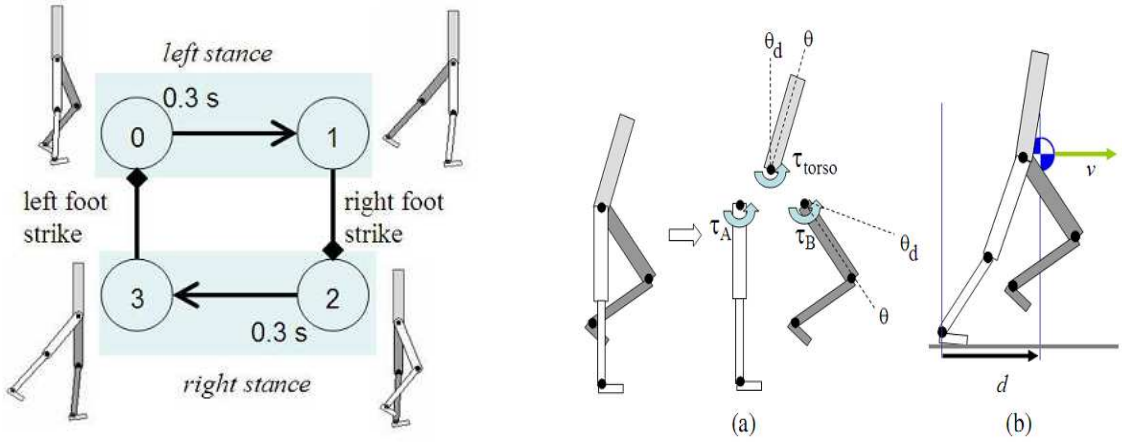


Figure 2.4: Finite State machine of walking cycle in Simbicon(left), elements of balance control strategy(right), (a) - relationship between torso, stance-hip, and swing-hip torques; (b) - centre of mass position and velocity [27].

b) Kinematic Simulation: For articulated humanoid character kinematics simulation is the study which does not consider mass and forces acting on it, mostly it involves positions, angles, speed and acceleration of the limbs of the character. So far in the literature mainly two methods are used: forward kinematics and inverse kinematics. In the first method world coordinates of a joint can be calculated by taking into account all the joints in the hierarchy including the starting one. Motion capture data information is an example for the forward kinematics that we just have each joint's local rotation and root's world coordinates. But without using data such as mocap, getting a figure to a configuration by specifying the joint parameters values can be tedious. For this kind of work *Versatile Walk Engine* is a good example [32]. In this work the findings of Neurophysiology and Biomechanics are used [33]. In the engine walking locomotion is represented as a cycle. You can see a normalized walking cycle in Figure 2.5.

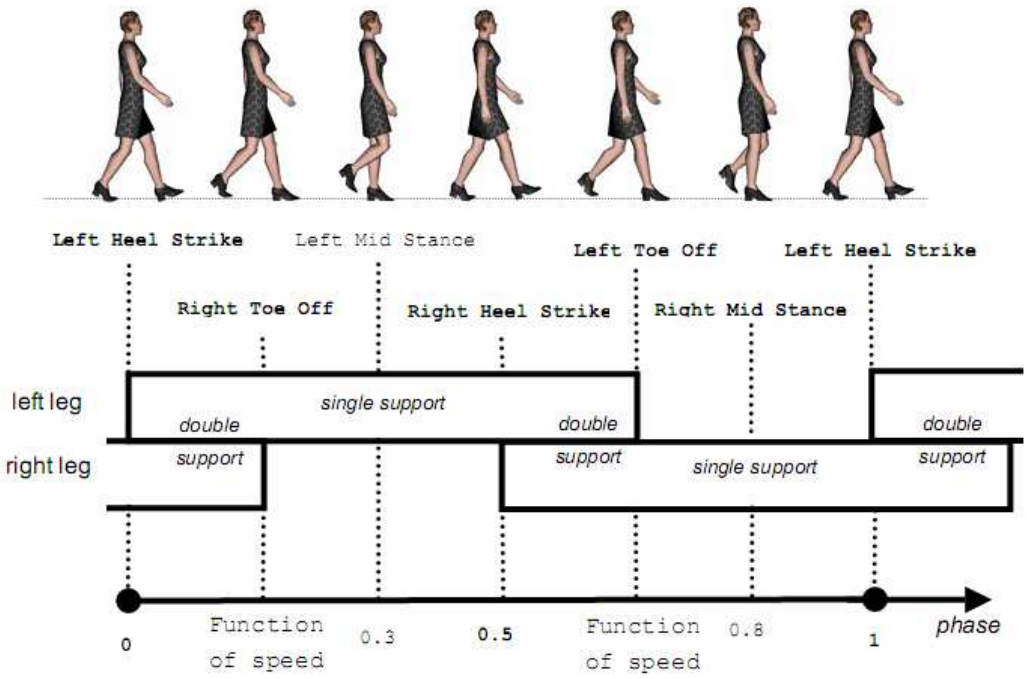


Figure 2.5: Normalized Walk Cycle [32]

In this system the toe phases are depending on the speed of the character which changes between [0.1 and 0.25], 0.10 is for the fastest speed and 0.25 is for the slowest speed. In the system for character speed and walk cycle frequency *Inman law* [33] is used, in which the cycle frequency is equal to $0.743\sqrt{v}$. For the hips, knees, feet and spine movements some results of biomedical observations are used [Appendix B]. The system has a good performance for large crowd simulations, but if we consider characters individually they seem like perfect walking without a characteristic (mood).

Conversely in the latter method, *inverse kinematics*, only the end positions for hierarchy has to be specified, and systems are calculating the required angles for each joint to get a configuration. In this method, if the system has a few constraints and many solutions then it is *under-constrained*. Conversely if the system has so many constraints and no solution, the system is called *over-constrained*. In the literature so far configuration of the hierarchy are calculated by using *Analytic Inverse Kinematics* and *Jacobian*. In the first one the joint values of a final desired pose can be determined by inspecting geometry of the linkage. In the *"FootSkate Solver"* library this method is used to find the constrained configuration of a character. Input system requires the limb lengths, root positions and the constrained end effectors' positions. Then it tries to find angles of joints in a reachable distance [30].

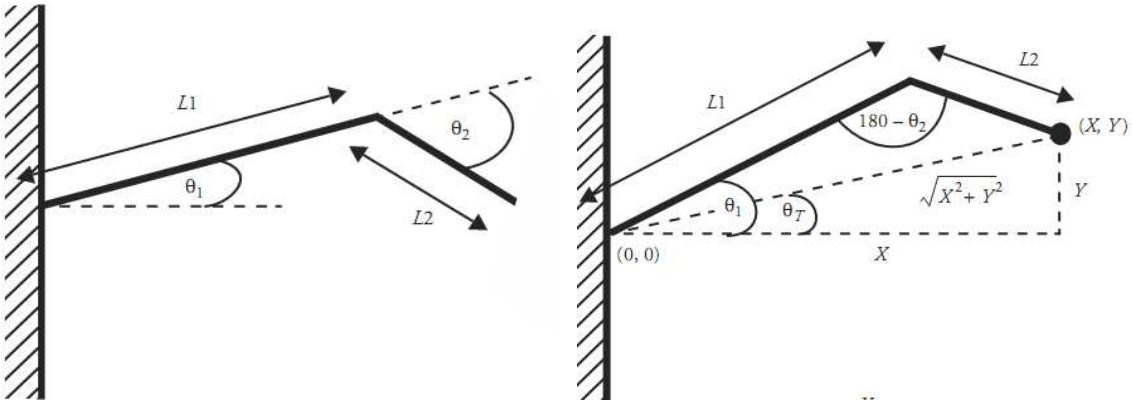


Figure 2.6: New configuration (left), applying cosine rules (right)

As it is shown in the above pictures if the user gives a new position *(X, Y)* for the hierarchy which was in the initial configuration, first and second angles can be found by cosine rules.

In computer animation most mechanisms are too complex to allow an analytic solution. For this as in *inverse dynamics* the motion can be incrementally constructed. In each time step angles of each joint are recomputed to move the end effector to the goal position [23]. Most of the methods in inverse kinematics are using *Jacobian* [10]. If the computed *Jacobian* matrix is square matrix, and it has an inverse matrix, this inverse matrix can be used to get the desired joint angle. But if the inverse matrix does not exist, the system is singular and there is no solution for this configuration. As a solution we can change the configuration of the linkage a bit to avoid the singularity. But when the configuration is close to singularities even small changes in the angles can give numerical errors and trembling results. If the number of DOF is more than constraints we can define another matrix by multiplying Jacobian matrix by its inverse in the case of its rows are linearly independent, "*pseudoinverse"*. This approach will give a square matrix. While using Jacobian if big steps are taken in joint angle space the end effector may not reach to the goal position. Still "*under-constrained"* cases have some problems with singularities. As a solution in the literature there are few approaches proposed so far by adding more controls [31]. In one of these methods, *damped least square* [31], user is supplying parameters to reduce the sensitivity of the *pseudoinverse.* Performances and complexities of different inverse kinematics methods can be found in ([34]). Among these solutions *selectively damped least square method* has best results

in the case of multiple end effectors. In this method a filtering is applied to all singular vectors and damping is defined in terms of the difficulty of reaching to the target [34]. On the other hand Jacobian transpose method in the case of single end effector is very fast and easy to implement. According to the survey results of [30] in most of the methods still there are trembling problems when the goal point is very close to singularities. Producing animated hierarchies with inverse kinematics is difficult, since it can give solutions which do not seem like human movements. Moreover, if we put some degree constraints to the joints, there is no guarantee for the system to give a (humanlike) result. For instance in [23] the system gets the position of wrists, ankle, hip and head positions. Then it computes the whole body configuration by using sequential inverse kinematics, normally this will be a waste of time for an animator. But still the results lack of small details.

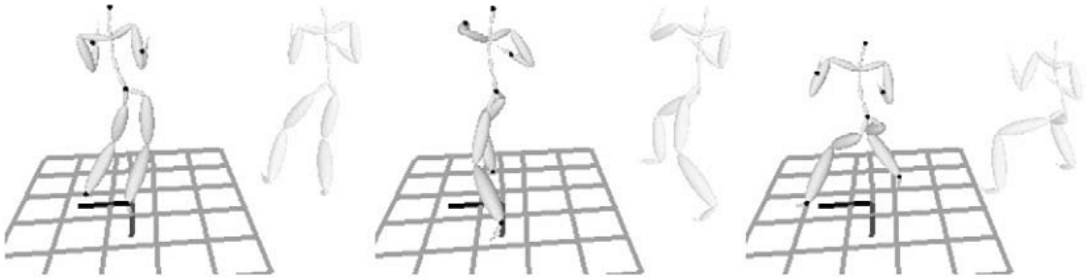


Figure 2.7: Sequential IK results for a boxing character [23].

Animations that are created by dynamics are more realistic than the ones generated by kinematics. The main problem of dynamic simulation of humanoid character is that it is computationally very expensive. The other drawback is that animator has to tune parameters for different type of motions. In kinematics motions adding small detail (sadness, happiness) can be cumbersome. In both cases there is a lack of artistic talent.

2.2.4 Behavioral Animation

In this type of animation, mental process of the humanoid character is simulated. To achieve the goal a locomotion system has to be implemented. On the top of the system there is a *preceptor*, depending on the triggered orders from here, *behaviors and goal actions* are loaded to the character. Depending on the selected behaviors character can

choose different styles of action, and these actions can be simulated by one of the animation type: key-framed, procedural or mocap [35], [36]).

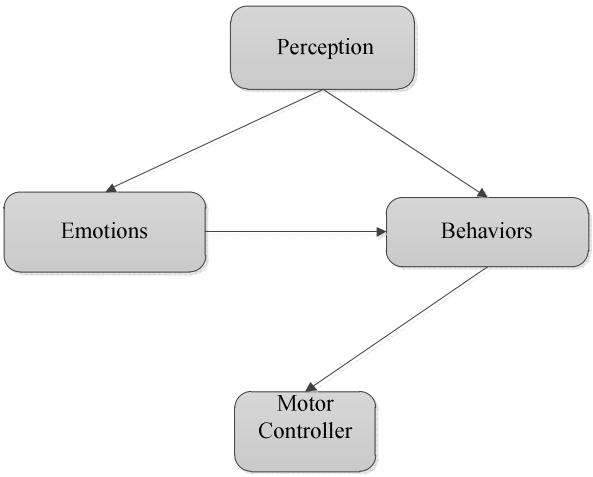


Figure 2.8: Structure of behavioral model

As it can be seen from the above diagram virtual character is animated by a "*motor controller"* which gets the goal information from the AI interface on top of it.

Before closing the "character animation types" section, we can say that each technique has its own place in the literature, depending on the needs of the applications. Procedural animation fits to physically realistic animation requirements; behavioral animation fits to crowd simulations where the detail of the motion is not so much important. Key-framing animation can be used to have artistic results (cartoon character movements), and motion capture data can be used whenever high realism is needed. But as stated in behavioral animation section, different kind of character animation techniques can be used considering the requirement of the realism.

2.3 Hierarchical Structure

Hierarchical modeling is the representation of the organization of a group of objects in a tree like structure. In these structures such as the one used in motion capture system, root changes the world coordinate and orientation of all the system. Other joints are rotating around their *ancestors*. In human like hierarchical representations, each joint can have only one ancestor. To compute *global coordinates* of a point on a link, all the

joints starting from *root* up to the *current one* have to be taken into account. Different local rotation angles, degree of freedom (DOF), direction vector and length values for each joint have to be considered.

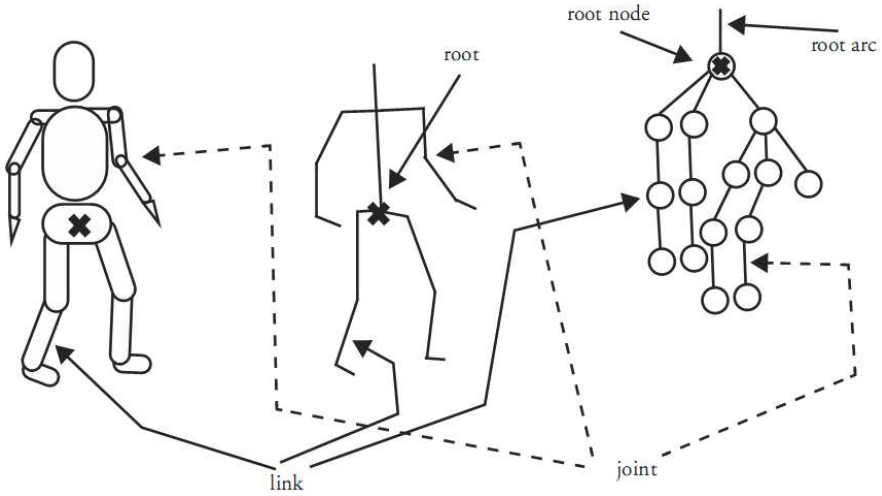


Figure 2.9: Example of a hierarchical representation [10]

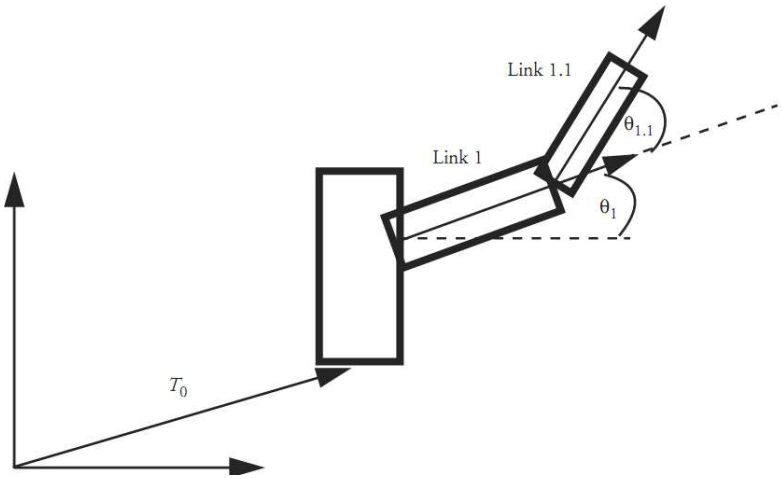


Figure 2.10: Rotations at joints [10]

To locate a vertex of *Link 1* in *world space* first it has to be transformed by multiplying it with joint's local rotation matrix, after this step result is multiplied with transformations, which were computed for up of the hierarchy.

$V_1' = T_0 * T_1 * R_1(\theta_1) * V_1$

For a vertex on $Link_{1.1}$: $V'_{1.1} = T_0 * T_1 * R_1(\theta_1) T_{1.1} * R_{1.1}(\theta_{1.1}) * V_{1.1}$

2.3.1 ASF/AMC File Format

Asf file is containing information about joint data field (length, axis, direction, DOF, limits) and organization of the hierarchy. First part holds information about each joint and in second part parent-child relation of joints is given [37]. As a starting point it should be known that *asf* file defines the *default pose* of the motion. Each pose defined in the *amc* file is calculated considering this default pose. In this way we don't have to hold any information about previous frames. *DOF* data of each joint is given as *"rx, ry or rz"*. The order of these keywords in a joint field defines the order of rotation for the joint. The absence of *dof* field means the joint is not rotating. Nonetheless absence of *rx, ry or rz* means joint never has a rotation in that axis. This field contains the axis of rotation. Apart from root joint all other joints in the hierarchy contain rotation values and order in the *axis* field. Root axis field only depicts the order of rotation. For this reason, *axis* field has to be involved into global/local rotation computations.

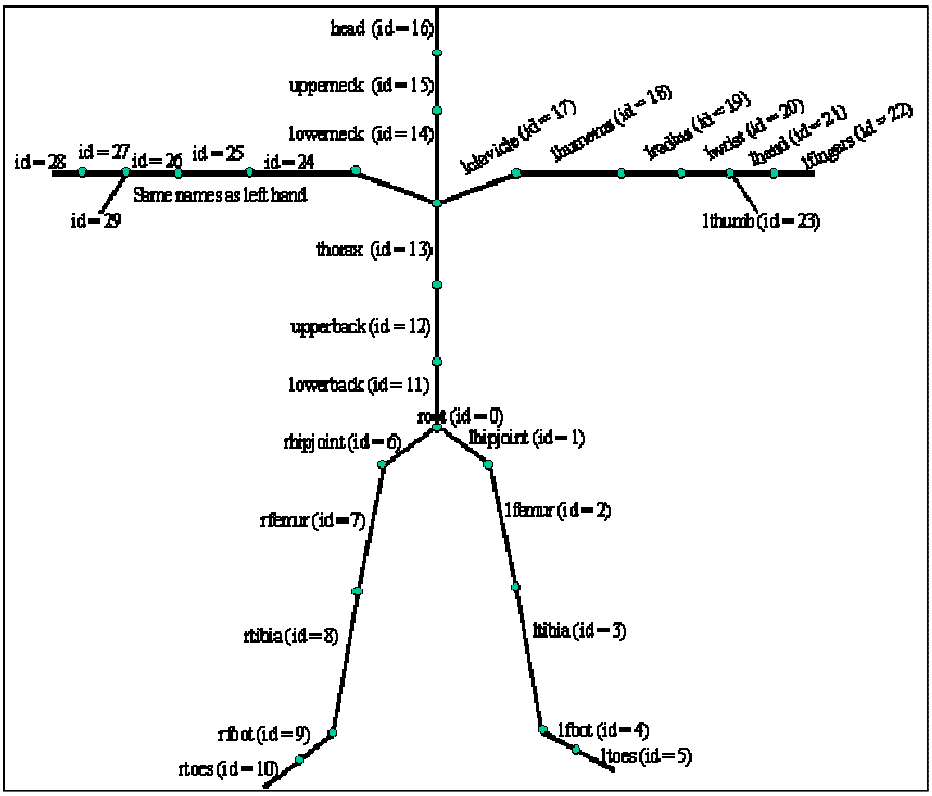


Figure 2.10: asf file humanoid hierarchical representation

In *amc* file for each frame the *global position* and *direction* of the character are given after *"root:"* keyword. Other joint name fields hold the local rotation values for each key-frame. To compute the global rotation/position of a joint at a key-frame, a matrix "*C"* has to be created using the *axis* values. In the asf file the order is given left to right. After creating this matrix, we have to create another matrix "*M"* for local rotation of the joint using the order defined in *dof* field. Another matrix ,"*B"*, contains the composite rotation from root up to the joint. Another variable, K, holds the global position of parent joint. For the root joint the position vector is directly used. The global rotation, *R*, of the joint will be:

$$R = B*Cinv*M*C$$

Global position of the *end point* of joint, *P*, will be:

$$P = K + joint\_length*joint\_direction$$

These are the methods how we find the position of end points of each limb such as *left toe, right toe, left hand* and *right hand.*

2.4 Rotations in 3D

In the amc file joint rotations are given in *Euler* format. In *Euler angle* representation, the axes of rotation are the axis of local coordinate system that rotate with the object, as opposed to fixed angle representation. For each axis the rotations is defined as different matrix. For further information about Euler angles and rotation matrices [38] can be checked. Despite the easy representation of *Euler angles* it has some drawbacks compared to quaternions.

In mathematics, quaternions are a number system that extends the complex numbers. For more information about quaternions theory and mathematics [39] can be checked. In computer graphics and animation quaternions are used for computing the rotations. A quaternion consist of variables, [w, x, y, z], first one represents angle information and the second is for rotation axis. It can be represented as [w, v] since the second one is a vector. So any rotation in 3D space can be represented via quaternions. It can be shown that quaternions are equivalent to Euler angles rotation. For instance, q = (w,v) = (W, X, Y, Z) = (cos(θ/2), sin(θ/2)*(x,y,z)):

$$\begin{bmatrix} 1-2Y^2-2Z^2 & 2XY-2WZ & 2XZ+2WY & 0 \\ 2XY+2WZ & 1-2X^2-2Z^2 & 2YZ-2WZ & 0 \\ 2XZ-2WY & 2YZ+2WX & 1-2X^2-2Y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the above matrix is equal to: $R_z(Z)R_y(Y)R_x(X)$. In our implementation for all kind of computations unit quaternions are used. To get back the rotation angles from a quaternion the above matrix can be used. But in this conversion some spatial cases can be met. Or in direct conversion from quaternion to Euler angle can occur [40]. For the first case: the value in M_{33} of the rotation matrix is equal to "$-\sin\theta$" where "θ" is the rotation angle around "y" axis. But "*sin30*" and "*sin150*" has the same values. This kind of spatial cases are handled in ([41]). In the second case, since we are using *arctan* for the conversation, some singularities may occur close to *90* degrees [40]. We used the algorithm in ([40]) to detect this kind of singularities. Quaterions have lots of advantages compare to Euler angles representation because of the following reasons:

- **Better Accuracy:** When more rotations are added to the same matrix, the rounding error of floating/double point math will add up and some visual artifacts may occur. Since quaternion representation has less mathematical operations to represent a composite rotation, it will have more accurate results.
- **Interpolation of Rotations:** Interpolating the rotations is very important in the case of motion blending. It is very hard to do when rotation matrices or Euler angles are used ([39]).
- **Complexity:** When two 4x4 matrices are multiplied 64 multiplications and 48 additions will be done. On the other hand in multiplication of two quaternions 16 multiplications and 12 additions will be computed.
- **Gimble-Lock:** Gimble lock problem occurs when two axes of rotations overlaps. In this case rotation around one of these axes can be cancelled by the other one [39].

In our framework we imported *FootSkate Solver* library, which is using quaternions. We added conversion from quaternions to Euler angles to this library. We used quaternions mostly for motion morphing.

2.5 Fourier Transforms

The idea of Fourier series is that any periodic function on the interval 2π *f(x)=f(x+2* π *)* can be decomposed into the contributions from *sin(nx)* and *cos(nx),* the general Fourier series may be written as:

$$f(x) = \frac{a_0}{2} + a_1\cos(x) + a_s\cos(2x) + a_3\cos(3x) + ... + a_n\cos(nx) \\ + b_1\sin(x) + b_2\sin(2x) + b_3\sin(3x) + ... + b_n\sin(nx)$$

In this equation, $\cos(nx)$ and $\sin(nx)$ are periodic on the interval 2π for any integer n. In other words, a function can be represented either in time domain or in frequency domain. In the above function each $a_k\sin(kx) + b_k\cos(kx)$ represents a frequency. If our function is continuous, time domain and frequency domain relationship can be represented as:

$$F(f) = \int_{-\infty}^{\infty} f(t)e^{2\pi ift}\,dt \ ,$$

$$f(t) = \int_{-\infty}^{\infty} F(f)e^{-2\pi ift}\,dt$$

These functions mean that one can go from time domain to frequency domain without loss of information or vice versa. If the *t* is measured in seconds, then *f* in these equations is in cycles per second or *Hertz* (the unit of frequency). Also these equations are working with other units. In this sense we can change *f* parameter with angular velocity (ω).

$$\omega = 2\pi f \quad \mathrm{F}(\omega) = \left[\mathrm{F}(f)\right]_{f=\frac{w}{2\pi}}$$

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{i\omega t}\,dt$$

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{-i\omega t} d\omega$$

For more details of the Fourier Transform, and relationship between *convolution* and *correlation* theorems please check [42].

What happens if our data is sampled, as in the data we read from mocap files? For any sampling interval, there is a *Nyquist critical frequency (f_c).*The *Nyquist* frequency represents the highest frequency that can be represented by something sampled at intervals of Δ, that is, a frequency having wavelength of 2Δ. Alternatively, if one is seeking to describe a function by a set of discrete values, we must sample the function at 2 times the highest frequency in the function [42].

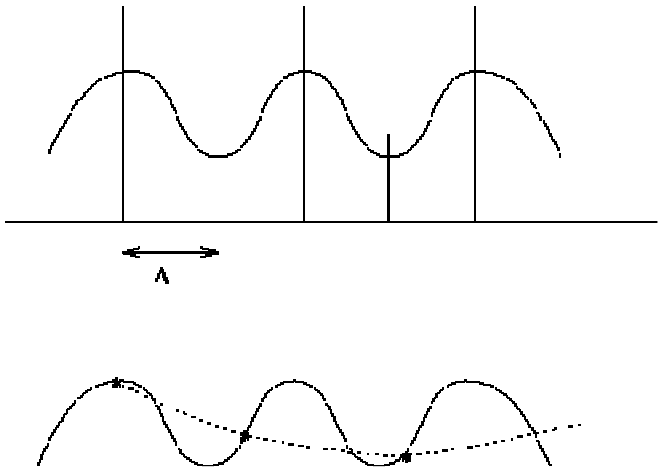


Figure 2.11: Critical sampling of the sine wave. Sampling at a lower rate will result loss of sine wave property.

In mocap files the sampling rate is very high, so critical frequency is not the case that can be a critical point in this kind of data. Discrete Fourier Transform can be represented as:

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

For further information about the theory of discrete Fourier transform (*dft*) please check ([41]). By using the methods told so far complexity of Fourier Transform for big number of samples is very high ($O(N^2)$). In mid-1960's, an algorithm developed by J.W. Cooley and J.W. Tukey to reduce the complexity of *dff* , it is called *fast Fourier transform*. In our framework fast Fourier transform implementation idea is taken from

[41]. This algorithm also has some tricks which is using the symmetry of Fourier transform. For the details of this algorithm [41] is advised to be checked. For a single *dof* in a time interval, key-frames interval, a window has to be selected which should be: $2^{n-1} \le frame\_number \le 2^n$ If the total frame number is less than 2^n we put *"0"* to those sample values to be able to calculate *fft*. After computing *fft* we can select some frequencies from the spectrum depending on the requirements (for smoothing the motion, slowing the motion…etc).

2.6 Autocorrelation

For cyclic motions we can use fast Fourier transform to find the period of the motion, to get the hidden motions in the original one, or to smooth it. To find the period of a motion we need to consider the *primary frequency/fundamental frequency*. In a periodic signal *fundamental frequency* is equals to the inverse of the period. In other words, it is the *lowest frequency* among the signals which are composing our signal/motion. When we consider this frequency as the *fundamental frequency* (f_0) in our periodic motions, in most of the cases we will have the wrong one. Some of the reasons are: noise in the motion, not being totally periodic, or there can have multiple pitches. For the details of the existing methods on detection of f_0 [43] and [45] can be checked. In our study we used *autocorrelation* method which is used in the literature on the purpose of solving same kind of problems. It is the *cross-correlation* of a signal with itself. Cross-correlation is a measure of similarity of two waveforms as function of time-lag applied to one of them. For continuous functions, *f* and *g,* the cross correlation is defined as:

$$(f * g) = \int_{-\infty}^{\infty} f^*(\tau) g(t+\tau) d\tau$$

Where f^* denotes *complex conjugate* of *f.* Similarly, for discrete functions, the cross correlation is defined as:

$$(f * g) = \sum_{-\infty}^{\infty} f^*[m] g[m+n]$$

For the continuous case τ denotes time lag, and for the discrete case *n* is denoting the window size we are shifting the signal. When we apply this method to a *dof* in our moving character we will have pitches at the frames where periods are matching each

other. Quality of the results depends on to our motion. For instance if we have more cycles and if the time intervals of each cycle are very close to each other, we will get better results. This method also can be used for the foot-plant detection for walking/running periodic movements.

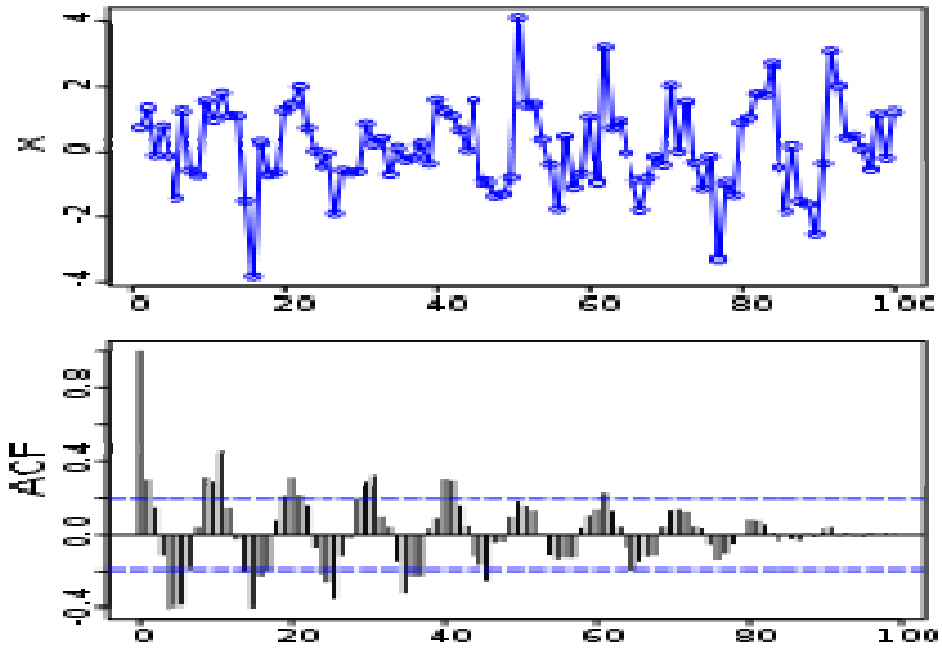


Figure 2.12: A plot showing 100 random numbers with a hidden sine function and auto-correlation of this data is on the bottom image.

2.7 Foot Plant Detection

Foot-plant occurs when an animated character steps on the ground. For a time interval foot stays stationary on the ground. But in this time interval depending on the joints integrated to the foot structure, some of them cannot be planted. For instance for right foot in a normal walking cycle as depicted in the *figure 2.5,* first *right heel* is planted and following this event then *right toe* joint becomes planted. After a while first *right heel* becomes off and *right toe*. For detecting foot planting time interval we consider each of these instances. Ignoring *foot plants* during motion blending will result in visual artifacts such as foot sliding. Moreover, even in motion capture data there can be some problems about foot planting due to the noise. In literature there are some works on this topic such as [46], [47], [30].

Foot-plant problem can be solved using the current vertical distance and translation speed of the ankle and toe joints. For each frame in the animation, these thresholds are checked. The foot is considered to be stationary when its position and linear velocity is lower than the specified threshold. But this method can give inconsistent results for the noisy or badly calibrated motion capture data. In other cases e.g. if your motion is processed or blended with another motion we can have foot skate artifacts. For this kind of reasons in [47] adaptive positional threshold is used. In this method, different kinds of walking cycles and running cycles are normalized. For *left heel* and *right heel* depending on the style of the locomotion plant intervals are predicted. Since this method is working considering the prerecorded normalized cycles it can be a drawback if a new style of motion is loaded. On the other hand in ([4]) *k* nearest-neighbors classifier is used. This is an interactive method that at the beginning the user gives some key intervals to the system, or at the beginning system can predicts some of the key frames for the *heels* and *toes*. Then, depending on the result user corrects or eliminates some of the predictions done by the system, system continues learning over the remaining predictions. Its interactivity can be the disadvantage but for the motion data which is not including noise or foot skates, this interaction can be in the minimum level. Since detection foot plant intervals for noisy data is hard to be detected this algorithm can save a lot of time of an animator. One example to traditional methods is [48]. Here in vertical positions and joint speeds are used for detection. Since this system cannot work for non-clean motions, some constraints are given via user interface.

Without knowing these intervals most of the motion processing cannot be accomplished. In [30] foot skates are cleaned by using this knowledge. In normal asf/amc motion capture file for the foot only ankle and toe joints are given. To use library of [30] we have to have to provide to the system *heel* and *ball* joints.

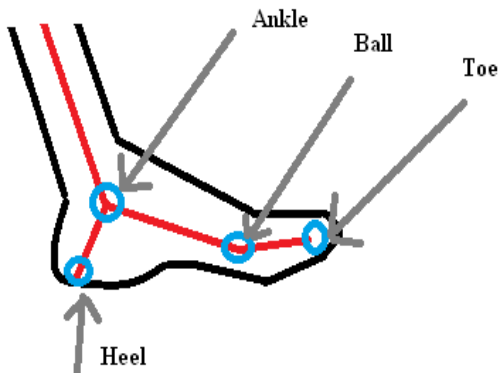


Figure 2.13: Foot joints

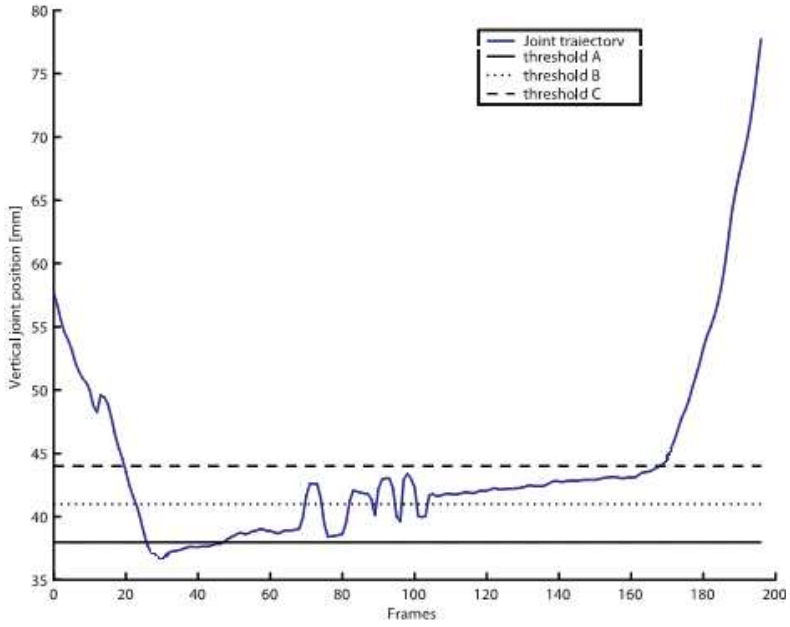


Figure 2.14: An example for vertical position of a ankle joint, as it can be seen from the figure it is difficult to decide on a threshold value [47].

In our system we used vertical distances and user feedbacks for predicting foot plants. In addition to this, on a normal walking cycle since both feet cannot be in plant phase, it is checked in our system. Details of our implementation will be given in the next chapter.

2.8 Motion Graphs

A motion graph is created in order to encapsulate connections among a database. In this graph, edges correspond to motion clips (sequence of motion frames) and nodes to choice points (specific frames) connecting these graphs. After selecting different frames, or windows of frames, and creating graphs of connections, a walk along the graph allows us to create new motions. Figure 2.15 illustrates the construction of a motion graph. Initially we have three clips of motion from database. There are different types of graph constructions; here we illustrate each frame as a node. In the same clip we do not consider frames for evaluation which are very close to each other in the same stance. If we evaluate all frames with each other it will take a lot of resources and time. There are different methods used to check how close the poses are of a character in two frames. In

[49] to calculate the similarity of two frames/poses weighted sum of squared distance between point clouds of poses are used.

$$\min_{\theta,x_0,z_0} \sum_i w_i \left\| \mathbf{p}_i - \mathbf{T}_{\theta,\mathbf{x}_\theta,\mathbf{z}_\theta} \mathbf{p'_i} \right\|$$

In the above formula $\mathbf{T}_{\theta,\mathbf{x_0},\mathbf{z_0}}$ is for finding a coordinate system for the two point clouds. It rotates a point **p** about the vertical (y) axis by θ degrees and then translates it by (x_0, z_0). The weights w_i may be chosen differently for some of joints to increase their importance. To calculate θ and x_0, z_0 values please check [49].

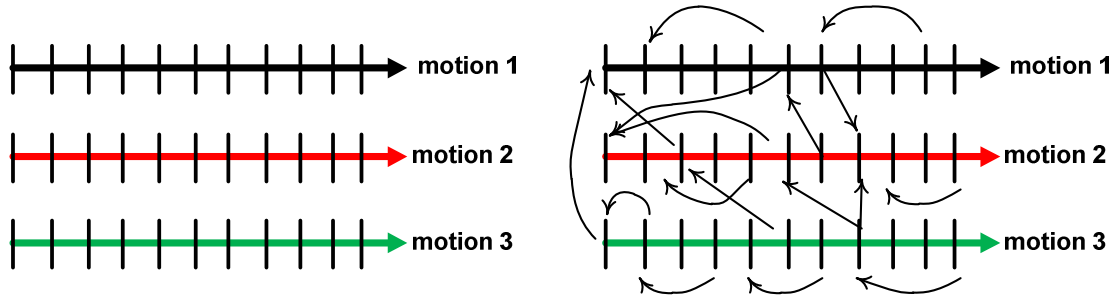


Figure 2.15: Constructing a directed graph using similar poses [50]

First-order Markov can be adopted for checking the similarities of two motions. A transition from one state to the next one only depends on the current state [50]. To check the similarities it is possible to take into account the joint angles and angular velocities of joints [50]. If this model is used it is possible to eliminate transitions from redundant velocities.

$$d(p_i, p_j) = \left\| p_{i,0} - p_{j,0} \right\|^2 + \sum_{k=1}^{m} w_k \left\| \log(q_j^{-1} q_{i,k}) \right\|^2$$

Where $p_{i,0} \in R^3$ is the translation position of the character at frame *i*, $q_{i,k} \in S^3$ is the orientation of joint *k* with respect to its parent in frame *i*, and joint angle differences are summed over *m* rotational joints. The value of $\log(q_a^{-1} q_b)$ is a vector **v** such that a rotation of $2\|\mathbf{v}\|$ about the axis $\frac{v}{\|v\|}$ takes a body from orientation q_a to orientation q_b.

Since we want to find the similarity of two poses, we do not need to take into account first part of the formula. First part of the formula is concerning the global position closeness of root joints. In this formula we can set different weights to give more importance to some of the joints. It is better to give higher values to shoulders, elbows,

knees, hips and spine. If we have previous knowledge on motion clips, such as foot stance intervals, we can use it to reduce the computation time. For instance, if a pose is in left stance interval and if the other one is in the right stance interval there cannot be a transition between these two frames. These two methods told so far are computationally expensive. The matrix of probabilities P_{ij} computed one of these formulas needs $O(n^2)$ storage space for n motion clips. If the motion database is huge it will be a problem since the computation can take a lot of time. For these reasons we have to avoid checking the distances of some of the frames. One option is skipping dissimilar contact states and other one is avoiding dead ends. After computing the graph, strongly connected subcomponents can be found. Probably we will have one very large strongly connected component and relatively small components. If the small components are very small we assign zero to the probabilities in these small ones.

The optimal decision on a motion graph which satisfies user constraints can be computationally expensive. Running two searches simultaneously from start state and from end state of the final motion (A* search) will increase the search performance by factor of two. Moreover it will reduce the memory allocation [51]. Another option for deciding on the optimal transitions using clusters. In [50] clusters are computed looking to the similarities of the frames. After grouping the frames into the clusters, and having the motion graph for each of the frame a cluster tree is constructed for different behaviors or choices. In the paper isometric Gaussian mixture model is used in which each cluster is an isometric multivariate Gaussian distribution with variable standard deviation. The aim of this model is keeping similar motion states within the same cluster [50] [52].

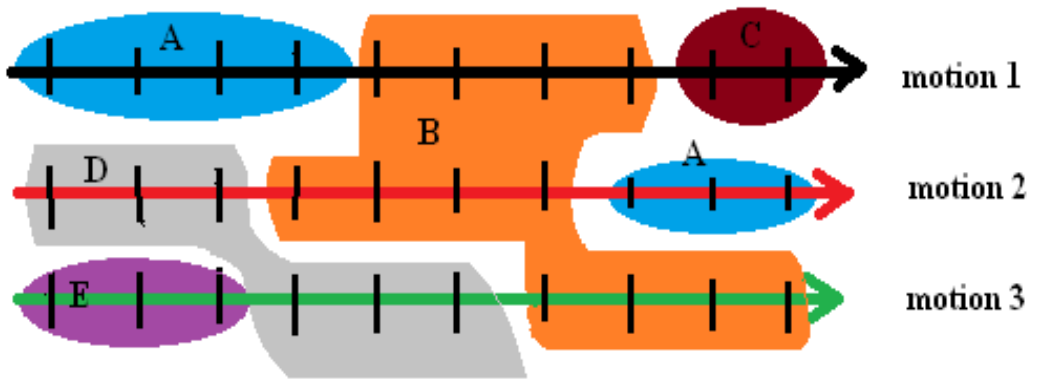


Figure 2.16: Similar poses clustered into same groups [50].

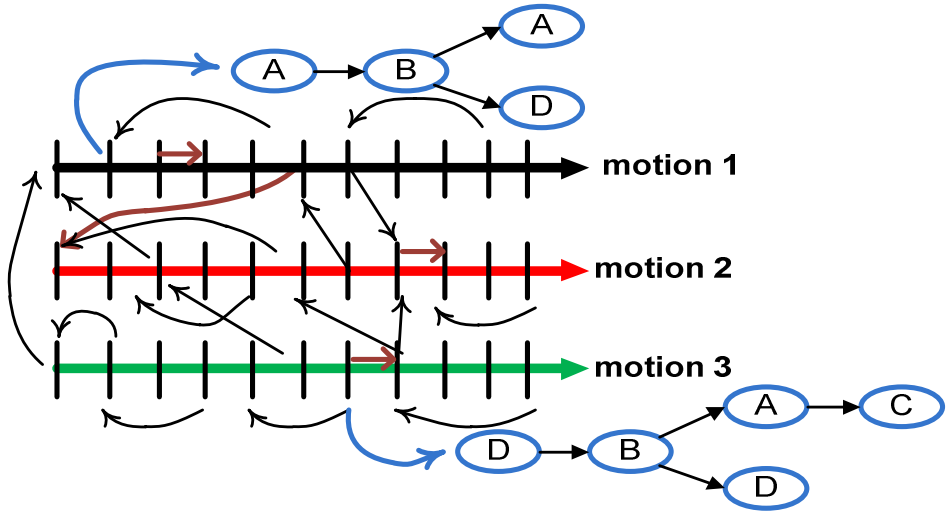


Figure 2.17: A cluster tree is constructed for a motion frame by traversing the motion graph to identify clusters reachable within a given depth (6 transitions for this figure) [50].

After detecting the transition path for a pose to the other one we have to smooth the transition by blending.

2.9 Motion Blending

By blending several motions it is possible to have multiple motions simultaneously. This means that the final motion is a blend of all active motions. Each frame in the final motion is found by blending each joint of the character with the corresponding joints in the other motions (eg. shoulders are blended with shoulders, etc). Motion transition is also using "*blending*", the difference lies in the fact that a transition is a temporary blend. When the decision of transition from one motion to another one is given a motion blend is started, but only for a certain transition length. For the starting frame the weight of the first motion is at highest *("1")* value and the second motion weight is lowest *("0")*, and first of these values is decreasing while the second one's weight is increasing when we progress on the transition interval.

As mentioned above we wish to blend corresponding joints of the motions. If we have two motions: $\mathbf{m}_i$ and $\mathbf{m}_j$, which are to be blended, former one has weight w_i and the latter has weight wj. Then the blend position of the two motions ($\mathbf{J}_{ik}$ and $\mathbf{J}_{jk}$) are found by

$$\mathbf{J}_{\mathbf{kpos}} = \frac{\mathbf{J}_{\mathbf{ik}} \cdot w_i + \mathbf{J}_{\mathbf{jk}} \cdot w_j}{w_i + w_j}$$

Equally for *N* animations blended together the position of the *k'th* joint can be computed by

$$\mathbf{J}_{\mathbf{kpos}} = \frac{\sum_{i=0}^{N} \mathbf{J}_{i_k} \cdot w_i}{\sum_{i=0}^{N} w_i}$$

The rotation of a joint is represented by quaternions, thus it is possible to do linear blending by using *spherical linear interpolations* (Slerp). Having a starting quaternion $\mathbf{q_s}$ and a quaternion $\mathbf{q_b}$ to be blended, we can find the linear interpolation $\mathbf{q_l}$ by

$$\mathbf{q_l} = \mathbf{q_s}(\mathbf{q_s}^{-1}\mathbf{q_b})^{\mathbf{w_b}}$$

Where $\mathbf{w_b}$ is the weight between 0.0 and 1.0 that determines how much $\mathbf{q_b}$ should be weighted according to $\mathbf{q_s}$. For blending more than two quaternions, we start by finding $\mathbf{q_l}$ for the first two quaternions and then setting $\mathbf{q_s}$ = $\mathbf{q_l}$ and performing the above equation for the next quaternion. This is repeated until all quaternions blended.

In [53] signal processing methods have been used for blending the motions, by introducing multi-resolution motion filtering to be applied to existing motion data. By adjusting gains of bands for each joint's angles a reliable blend is constructed. This method suffers from being non-automatic and only working with two motions. In addition to this, general linear blending do not consider if one motion speed is faster than the other one. On the other hand, methods told in section 2.8 solve these problems by finding the closest transition frames and considering angular velocities of joints. But these methods are working for detecting the closest frames for transitions poses, but what happens if we want to blend two whole motions such as a *happy walk* and *a sad walk*? Since the speed of each motion can be different we have to find the closest frames in each stance intervals. In Bruderlin and Williams work [53] *timewarp* algorithm is used. The solution is decomposed in two steps: finding the optimal sample correspondence between the two signals, and applying warp. The *vertex correspondence* problem is defined as finding the globally optimal correspondence between the vertices (samples) of two signals: to each vertex of a signal, assign (at least) a vertex in the other signal such that a global cost function measuring the "difference" of two signals is

minimized. In this sense the problem is similar with shape blending and it is solved by dynamic programming optimization techniques [53].

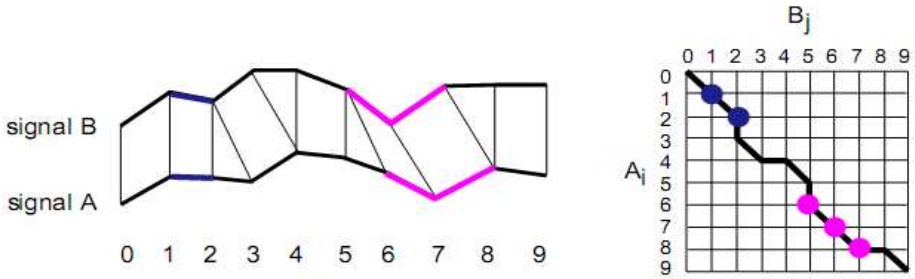


Figure 2.18: Vertex correspondence problem and cost functions [53].

In this work Sederberg's shape blending algorithm is used [54]. It measures the differences in "shape" of two signals by calculating how much work it takes to deform one signal into the other. The cost function consists of the sum of local stretching and bending work terms, the former involving two, the latter three adjacent vertices of each signal. Intuitively, cost depends on the distance difference between two adjacent vertices of one signal and two vertices of the other signal. Similarly for angle difference between three adjacent vertices on each signal defines the cost. After finding the lowest cost correspondence between vertices on two signals or motion curves, second part is to apply the warp for the optimal blending. As in speech recognition [55], three cases are applied: substitution, deletion and insertion. The warped signal is denoted by $\mathbf{B}w$. If $\mathbf{A}i$ and $\mathbf{B}j$ are related by substitution it follows that $\mathbf{B}w_i = \mathbf{B}j$. In the case of multiple samples of **B** correspond to one $\mathbf{A}i$, $\mathbf{B}w_i$ will be equal of the *mean* of the corresponding **B** signal values. In the case of multiple **A** sample values correspond to a single **B** signal value $\mathbf{B}w_i$ is determined by computing a cubic B-spline distribution around the original $\mathbf{B}j$ value.

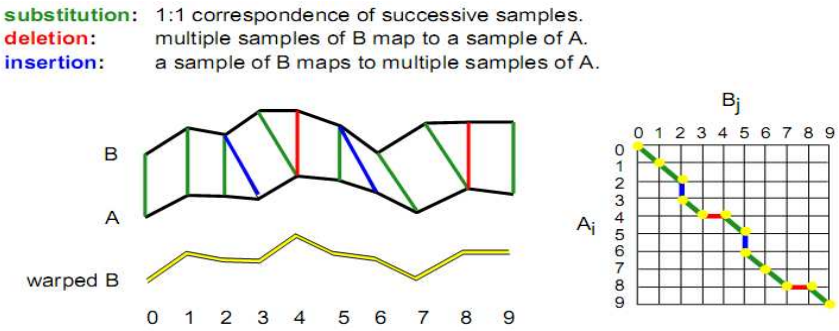


Figure 2.19: Applying time-warp [53].

Glardon and Boulic based their time-warp algorithm ([56]) on considering different support phases and their durations (left heel stance, left toe stance, etc). According to their algorithm we want to blend two motions: motion A, where frames f_{a1} and f_{a2} delimit a footprint phase, and motion B, having frames f_{b1} and f_{b2} delimiting a similar footprint. According to the elapsed time Δt corresponding frames of f_a and f_b has to be computed.

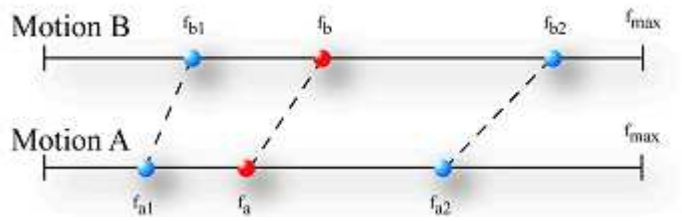


Figure 2.20: Frame alignment between two motions [56].

Frequencies F_a and F_b are associated to motion A and B. Since motions may have different frequencies F_a chosen as the initial frequency F, and phase variable φ_{curr} as described in their previous work ([57]). Then frame f_a is determined using equation 2, where $f_{\max}$ is the number of frames of each normalized motion sequence (walking running or jumping).

$$\varphi_{curr} = \varphi_{prev} + \Delta t F \qquad (1)$$

$$f_a = \varphi_{curr} f_{\max} \qquad (2)$$

The frame f_b is computed by a linear interpolation between f_{b1} and f_{b2} with the blend weight parameter p, varying from "*0*" (motion A) to "*1*" (motion B), defined in equation 3.

$$p = \frac{f_a - f_b}{f_{a2} - f_{a1}} \qquad (3)$$

Finally, the frame to be displayed is computed by applying quaternion spherical interpolation (Slerp) between f_a and f_b, using parameter p.

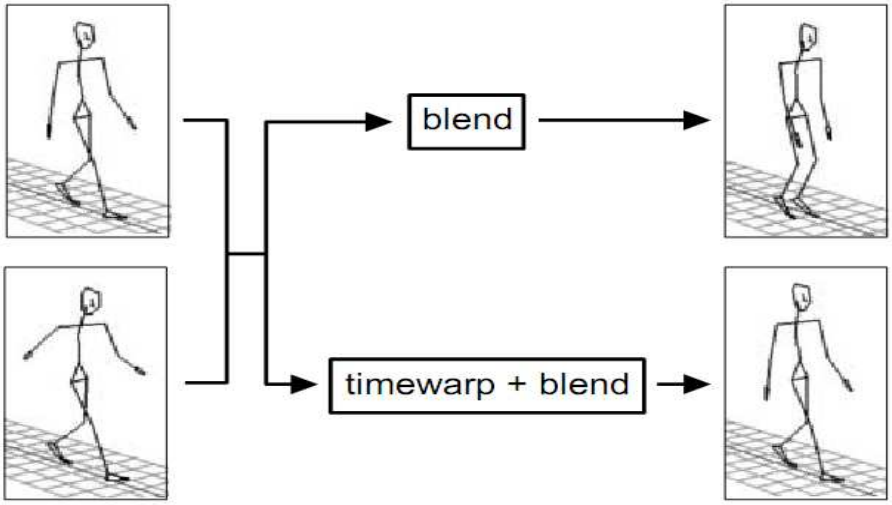


Figure 2.21: Blending two types of walk with (bottom) and without (top) considering time-warp [53].

2.10 Motion Path Editing

Given a path corresponding to an initial motion we can factor the motion into the path and residual by placing the motion in the moving coordinate system defined by the path. Then path curve can be replaced by another one that animated character is following the new path [58].

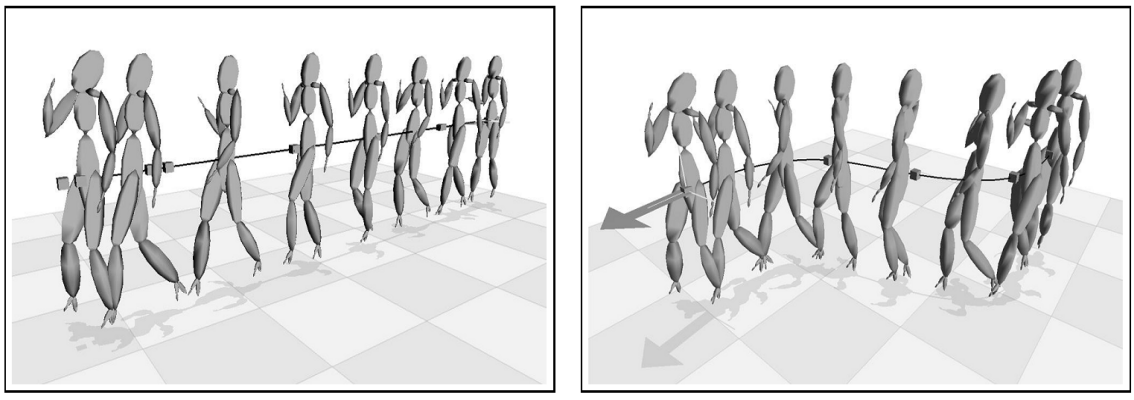


Figure 2.22: Editing a path of normal walk motion, left image shows the normal original motion, right one shows the edited one [58].

In this work path transformation preserves certain aspects such as motion. For instance joint angles are unaffected by the process, but because of the transformation of the root joint the positions of the end effectors are not preserved [58]. When arc length between two control points is extended foot/feet will start slipping along the floor. For this

reason before processing the original motion foot plant intervals are needed to fix the new motion. Using the previously known foot stances new motion can be cleaned by using IK method [30]. But if the arc lengths are extended too much, foot skating intervals can be impossible to be fixed using IK method (*Foot Skate Solver*) since its legs cannot be extended that much to reach to the constrained positions. In this case such intervals can be filled using motion graphs, or filling this gaps by using blending methods.

This work is extended to be used for variety of motions in [59]. In this algorithm all the motion is partitioned to smaller clips, each clip defines different type of motion such as walking (one clip), holding a box (one clip), carrying the box (one clip). In this way this method becomes very general and user selects the clips he wants to change. Still foot skating and unreachable leg extensions have to be handled.

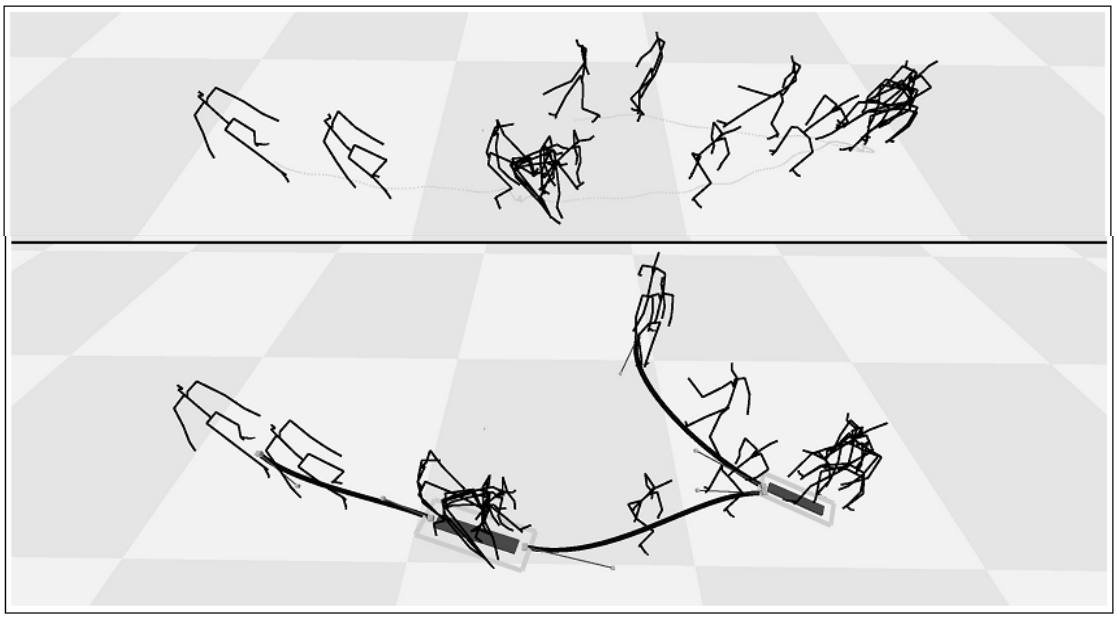


Figure 2.23: The top shows the original motion of a character which is picking up a box and carrying it. The lower one shows the motion after user manipulated the walking path. Note that rectangles are defining the inflexible regions (tiles) which are connecting the Bèzier splines (flexible regions) [59].

Before closing the chapter we have to denote that quality of the re-used motions depend on physical correctness. Safonova aabd Hodgins ([60]) made a survey on physical correctness of interpolated human motions. They considered 1) linear and angular momentum during transition, 2) foot contact, static balance and friction with the ground during stance and 3) continuity of position and velocity between phases. Since blending

phase is the bottleneck of the re-using and processing the motions, there is a need for robust methods. Their proposed methods also can be used in dynamic character animations.

CHAPTER III

IMPLEMENTATION METHODS

In this chapter we will give a brief description of the methods which are implemented in our frame-work. And then shortly we will describe our system architecture.

3.1 Methods implemented in the Work

a) **Foot planting detection:** Our *foot planting detection* algorithm automatically finds the strike key-frame intervals for each of the left heel, left toe, right heel and right toe when the motion file imported into the system. Detected results are returned back to the user. If there are some missing intervals user can feed the system by providing some seeds by means of the scripting window. The success of the implementation depends on how much clean is the original motion (having less noise and no foot skating).

b) **Fast Fourier Transform:** If a motion is periodic we can use fast Fourier analysis for processing the motion. For instance we can use fast Fourier transform analysis to change the speed of the original motion. Getting the benefit of Fourier transform and autocorrelation methods we tried to detect the period of a cyclic motion. In this way we can get the characteristic of motion and we can import this data to dynamic humanoid simulators such as Simbicon[27] which is capable of imitating the real motion.

c) **Motion Signal Processing:** Image multi-resolution filtering technique can be applied to the articulated figures. The lower frequencies of signal are considered to represent the base activity (e.g., walking) while the higher frequencies are the idiosyncratic movements of specific individual. Frequency bands can be extracted and manipulated by the user to define another motion.

d) **Path Editing:** When we import the motion capture data we already have the animated character's root position. By using the points on this path we can define an "*n degree of Bezier curve"* to have the main direction of the root path. Original motion can be altered by manipulating the control points on this curve by calculating the change in the orientation of this curve and multiplying this change will also change the character's orientation.

e) **Motion Blending:** We used some cyclic motions which are taken from Graphics Group of Carnegie Mellon University. We chose motions such as sad walk, happy walk, relaxed walk, etc. Then we detect the stance interval of these motions. After calculating the average motion for each foot stance intervals, we saved the data in a defined file format. These files contain averaged motion information for left stance and right stance. These files can be blended by among themselves with different ratios and also they can be applied to a normal mocap data for which we already know the foot plant key frames.

3.2 System Architecture

This section describes our framework which visualizes motion capture files and creates new motions by processing them. Our framework is an extended version of asf/amc viewer which was developed by Computer Graphics Group of Carnegie Mellon University, can be found in "*Tools"* section of motion database website. Our extended framework comprises of the following components:

1. **Motion Capture File Pre-processing:** The motion file pre-processor for each read file (only asf/amc formats) saves the initial degree of freedom values of joints. Then it calculates the local/world translation matrices for the hierarchy for each key frame. Following this step we try to detect the foot plant intervals, by taking into account positions of feet with respect to position of root joint. These differences will help us to understand if the feet are motion or stepping. After this step the results are printed on the screen, if the user is not satisfied the plant intervals system can be feed with seed intervals via the command line. If the

result has visual artifacts or noise, user is also able to give the exact intervals.

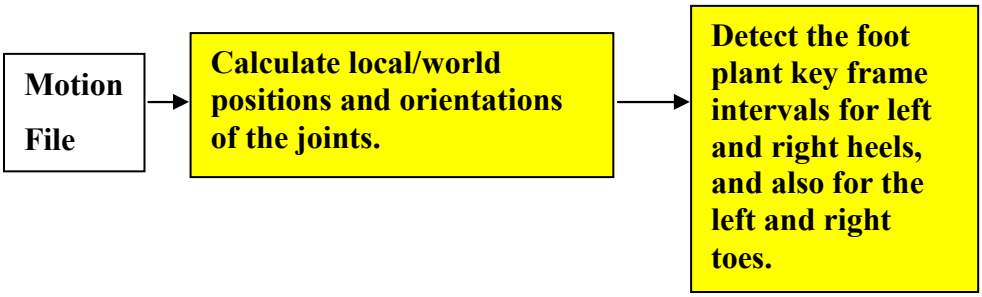


Figure 3.1: Pre-processing of motion capture files

2. **Fast Fourier Transform and Signal Processing:** When the motion files are imported into the framework, each one can be selected by double clicking on the characters in the 3D environment or by clicking the motion file names in the graphical interface. When a motion is selected, by using the interface Fast Fourier Transform (fft) or signal bands can be calculated for all the joints of a character. After playing with the coefficients of signal bands or applying different fourier filters the changes for each DOF can be followed from the *"DOF Curve Window"*. These step can result in some problems such as foot skating. Via the scripting window user can try to clean the result by calling "*FootSkate Solver*"[30].

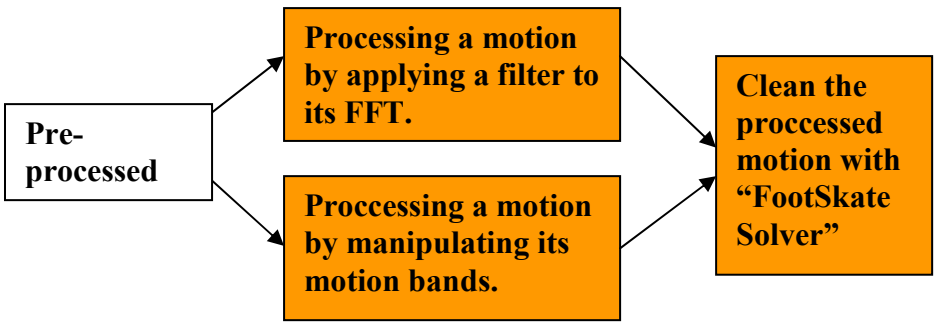


Figure 3.2: Fourier analysis and Signal Processing

3. **Motion Blending:** After importing a cyclic motion, pre-processing stem is applied to it. Then we calculate the average of the left and right stances. After this step, the result is saved to a file. Later these files can

be imported for blending other walking humanoid character. For blending a walking humanoid character we have to know the foot plants.

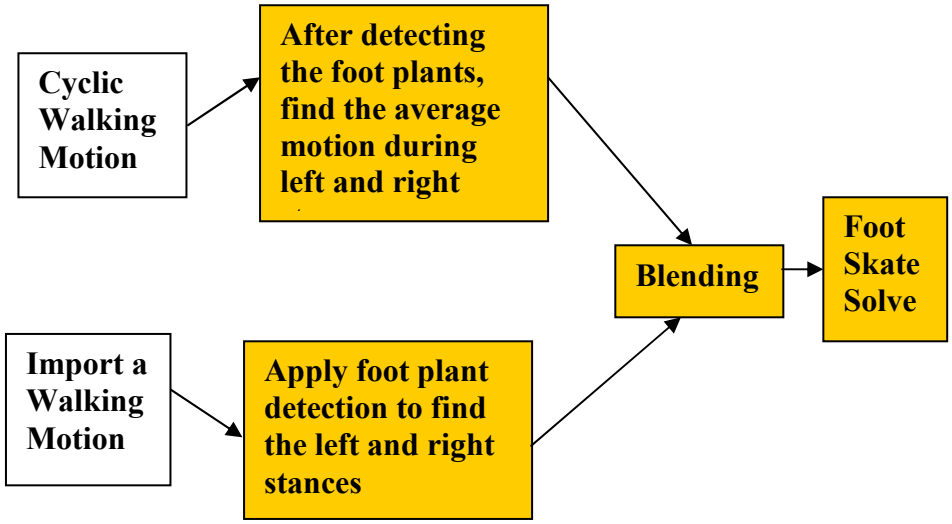


Figure 3.3: Motion Blending Diagram

4. **Path Editing:** If there is a pre-processed motion in the system, using the scripting interface we can query to the system to construct the smoothed motion path. After having it we can manipulate the path by changing the positions of the control points. When there is a change in the path, system calculates the new motion, and following this *"FootSkate Solver"*[30] takes its part to fix the foot plant problems.

Figure 3.4: Path Editing Diagram

CHAPTER IV

RESULTS AND IMPLEMENTATION DETAILS

In this chapter we will discuss the techniques we used in our framework for reusing motion capture data. For each of the method most crucial part is detecting the foot stance intervals. For instance when we edit the path of the motion it is highly probable to have new motion with foot sliding. For correcting this kind of artifacts we need to know when foot plant is occurring in the original motion. Then we will describe other methods such as motion blending, using fast Fourier transform for processing the original motion.

4.1 Foot Plant Detection

As it described in the previous chapter foot plant occurs when the animated character steps on the ground or on the object. During the time of the stepping foot is stationary and continues to contact to object/ground. Ignoring this time intervals will result in visual artifacts ranging from foot skates to unnatural motion where foot is not in contact with ground when it is supposed to be. As stated in previous chapter foot skate means foot is sliding on the ground instead of being planted at a position. In motion capture data we can also detect *footskate* artifacts due to the noise. Automatic identification of foot plant for the general case is difficult. Most of the algorithms fail since human beings have variety of motions. In our frame work we tried to detect foot plant intervals by using different methods. Here we will give our method and results. Moreover if the user is not satisfied with the results he/she can give the exact intervals to the system by using scripting interface. Our first method is checking the foot displacement and velocity by using a threshold. Second method is using the data taken from the user as clues for the planted intervals. And third method is checking the curves of both knees when the character is walking. In a foot plant interval there are four phases as in the figure 3.1: heel-strike, toe-strike, heel-off, toe-off.

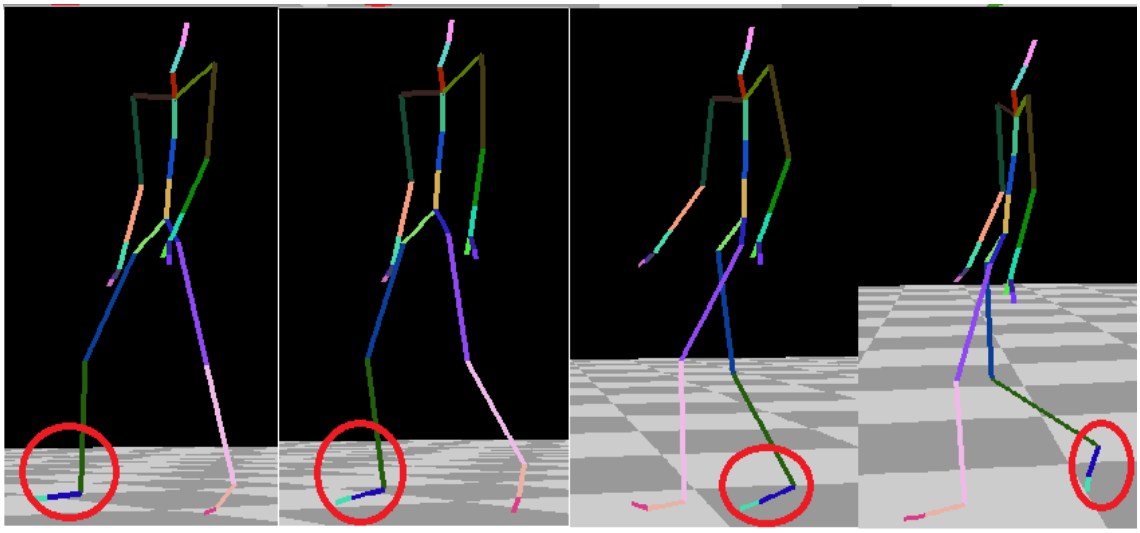


Figure 4.1: Example of an right foot plant interval: right-heel strike (first), right toe-strike (second), right heel-off (third), right toe-off (fourth)

4.1.1 Using Displacements and Velocities

When feet are planted they are supposed to be stationary and contact with floor. But because of the sensor noise or calibration errors during capturing the motion there can be small movements in the plant time interval. So for this reason, most of the time heel and toe displacement won't be zero. For detecting these intervals we put a *threshold* for the vertical displacements of the heel and toe, and another *threshold velocity* for foot joints.

Inputs: $\mathbf{d}_{vert}$ Vertical displacements of each joint (left heel, left toe, right heel, and right toe)

$\mathbf{v}$ Velocities of the joints at each frame

$\mathbf{v}_{threshold}$ Velocity threshold for each joint

$\mathbf{w}_{search}$ Search window size for detecting the noise.

Step 1: Compute absolute joint displacements and joint velocities for each frame.

Step 2: Evaluate joints considering only vertical displacement and joint velocities.

Step 3: Check the noise intervals using a fixed size window.

Step 4: Eliminate the foot strike intervals which are totally inside of the other one.

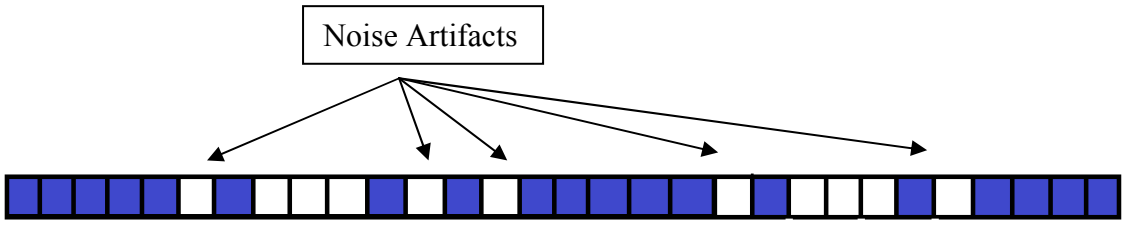
First and the second steps are straight forward to compute. When these steps are completed we will have two evaluation tables: one for the vertical displacement and one for velocities of the feet's joints. These tables consist of four columns: *left heel, left toe, right heel, and right toe columns* as shown in the figure 3.2. In second step for each frame we compare the each joint's *vertical displacement* and velocities with the given threshold and the result is saved into one of these tables.

Table 4.1: Threshold Evaluation Table Structure

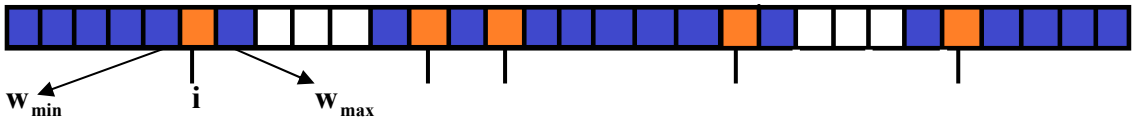
Frame	**Left Heel**	**Left Toe**	**Right Heel**	**Right Toe**
1	True	False	False	False
2	True	True	False	False
….	…	…	…	…
355	False	False	True	True

After having these tables we check each row for possible noises as shown in figure 3.2 (step 3). For this step we use different sizes of windows. We observe that a 30 frame size window is giving good results. We put this window to a *seed key frame* (each of the *non-stance* key frames). The percentage of *marked frames* (*stance*) around this *seed frame* is computed. If the result is lower than *thirty percent* then it is accepted as *non stance* key frame for that joint. But if it is higher, *un-marked* key frames between the *marked* ones are made marked.

Row Heel Stance (Noisy Data)



Detecting Noise Artifacts (Detecting and fixing noise intervals with a window)



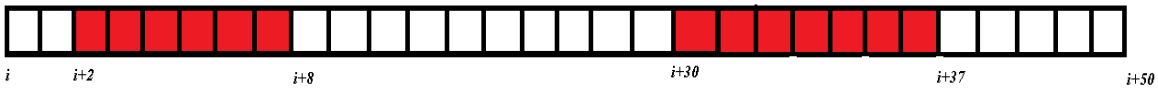
Noise Free (Fixed Data)

Figure 4.2: Steps of detecting and removing noise from data

If we check an example normal real walking data as depicted in figure 2.5 we will realize that *left foot* and *right foot* stance intervals are crossing each other. But by using the method which is told so far we can find *some intervals* which are small compare to the other *stance interval* which is including it/them. This kind of events cannot occur in a normal walking/running cycle. To avoid this problem we compare the intervals which are computed up to the last step. Another possible result that we can have: small consecutive stance intervals having small gaps between each other and not overlapping with other foot stance intervals. These gaps can occur because of the threshold we used is not valid for each of the stance intervals or motion can be very noisy (Figure 3.4).

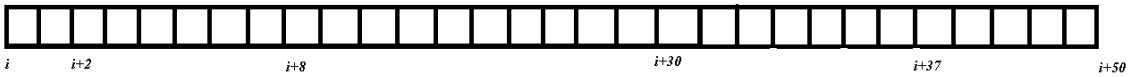
Left Foot Stance Interval

Right Foot Stance Interval



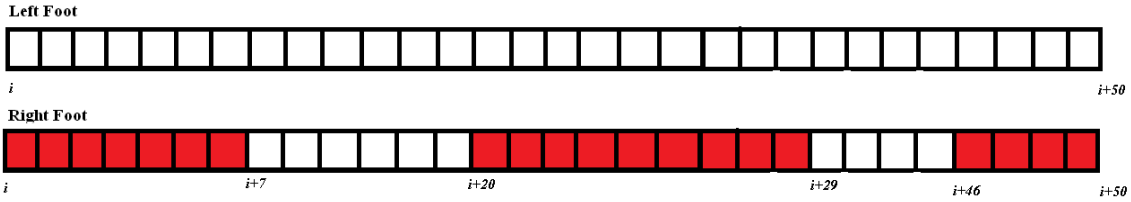
Left Foot Stance

Cleaned Right Foot Stance Intervals

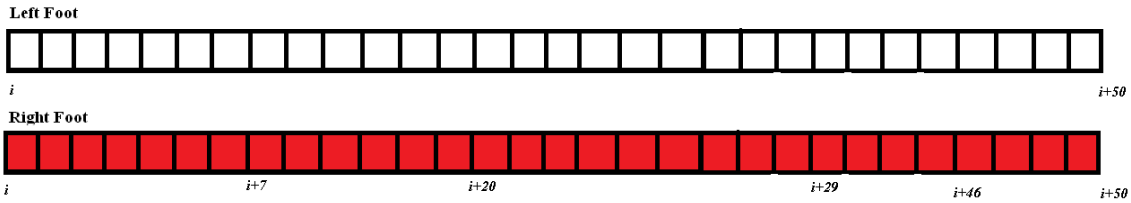


Cleaned *Right Foot Stance* intervals *wrt* left foot

Figure 4.3: Process of Step 4



Stance intervals of left foot and right foot. Red = stance, White = no stance.



Stance intervals merged for right foot.

Figure 4.4: Merging small stance intervals.

With the brute force method (just using the vertical displacements in the above algorithm), for some of the motions we have good results but some kind of the motions this algorithm gives poor results. For instance we tried a normal cyclic walking data which is taken from Carnegie Mellon University's database with *07_01.amc* name. This motion has 315 key frames in total. You can check the result of the algorithm for this data from the below Tables.

Table 4.2: For *07_01.amc* automatic foot plant intervals detection results (left) and manually checked results (right)

Threshold	0.001
Window Size	21frames for *heels*, 11 frames for *toes*
Left Heel Plants	61-110, 192-243
Left Toe Plants	71-135, 202-263
Right Heel Plants	2-46, 127-165, 257-310
Right Toe Plants	7-57, 138-189, 262-312

Left Heel	63-113, 197-243
Left Toe	71-138, 201-263
Right Heel	0-47, 131-167, 259-310
Right Toe	7-68, 136-195,265-315

Relaxed walking is another motion data which we downloaded from same database. It was under the *Stylized Walk* subject. This motion has more than 2500 frames. We partitioned it to small portions since it is very big and we just need a small portion of it for blending and other processing methods. So our small relax walking portion is called *relaxed_01.amc* in our dataset. It has *670* key frames in total.

Table 4.3: Automatic foot plant intervals detection for *relaxed_01.amc* (left) and manual detection (right)

Threshold	0.001
Window Size	21 for *heels*, 11 for *toes*
Left Heel Plants	129-177, 313-354, 484-558
Left Toe Plant	134-203, 317-397, 498-567
Right Heel Plant	3-98, 201-286, 385-466, 563-661
Right Toe Plant	20-108, 229-305, 413-485, 592-669

Left Heel	102-176, 293-369, 473-546
Left Toe	1-21, 113-215, 313-397, 485-580
Right Heel	0-97, 206-276, 385-464, 566-645
Right Toe	23-118, 215-310, 394-487, 585-670

From Carnegie Mellon University's database under the same section we imported *marching* walking motion. We partitioned it in to small sections. We saved them in our dataset as: *marching_01.amc* and *marching_02.amc*. We tried the first one with the using the above algorithm by checking vertical displacements.

Table 4.4: Automatic foot plant detection for marchin_01.amc (left) and manual detection (right)

Threshold	0.0155
Window Size	21 for *heels*, 11 for *toes*
Left Heel Plants	48-57, 149-206, 250-259, 286 309, 352-361, 390-412
Left Toe Plant	183-207, 286-309, 400-412
Right Heel Plant	6-58, 107-155, 209-256, 310-357
Right Toe Plant	8-57, 108-156, 209-256

Left Heel	52-87, 156-176, 258-276, 360-410
Left Toe	1-7, 50-103, 153-204, 254-302, 355-413
Right Heel	6-40, 107-135, 207-226, 306-331
Right Toe	0-54, 102-154, 204-255, 303-356

As it can be figured out from the results in the last table, sometimes just checking the vertical displacement is not enough. This *marching* animation is a bit complicated since toe strikes occur before heel strikes. And in this data there are some problems about left foot displacement. If we check the table we can realize that *right foot* stance intervals are quite successful compared to the left foot. In general, for most of the motions *0.001* is enough for detecting these intervals considering noise effect. But for marching animation this threshold should be updated up to *0.0155* to get a relevant result. On the other hand if we add the velocity condition into *step 2* the result will be much more different. If one foot is *stationary* and the other one is moving, it does not matter that if the planted one has a noise and moves a bit during planted interval. As expected the other leg should have a bigger velocity. So if we take into account the velocity factor the result for *marching* animation becomes like:

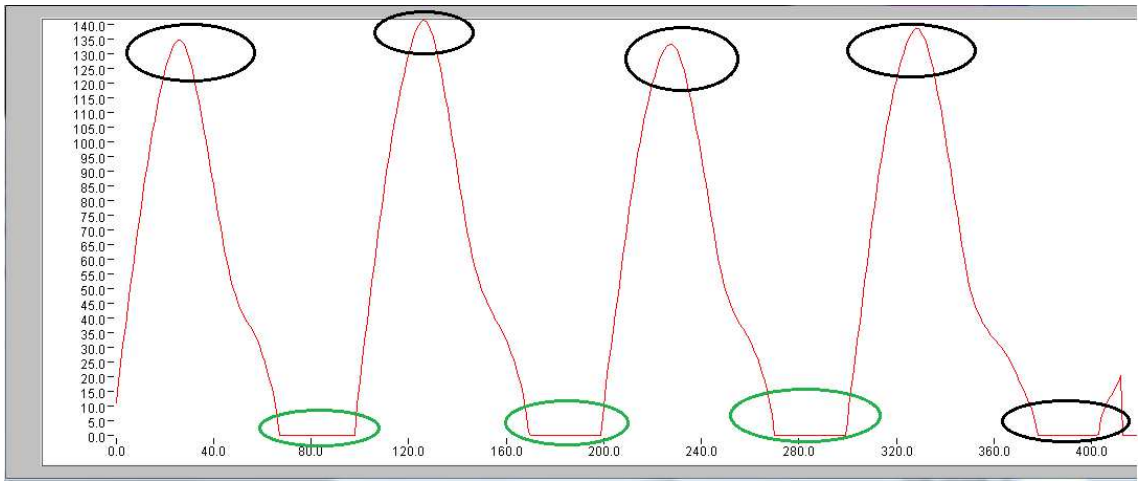
Table 4.5: After applying velocity cooperation to joints

Threshold	0.001
Window Size	21 for *heels*, 11 for *toes*
Left Heel Plants	52-108, 156-209, 256-300 , 389-412
Left Toe Plant	82-108, 181-206, 285-311, 389-412
Right Heel Plant	7-46, 107-148, 209-249, 310-350, 412-412
Right Toe Plant	9-52, 108-155, 209-254, 412-412

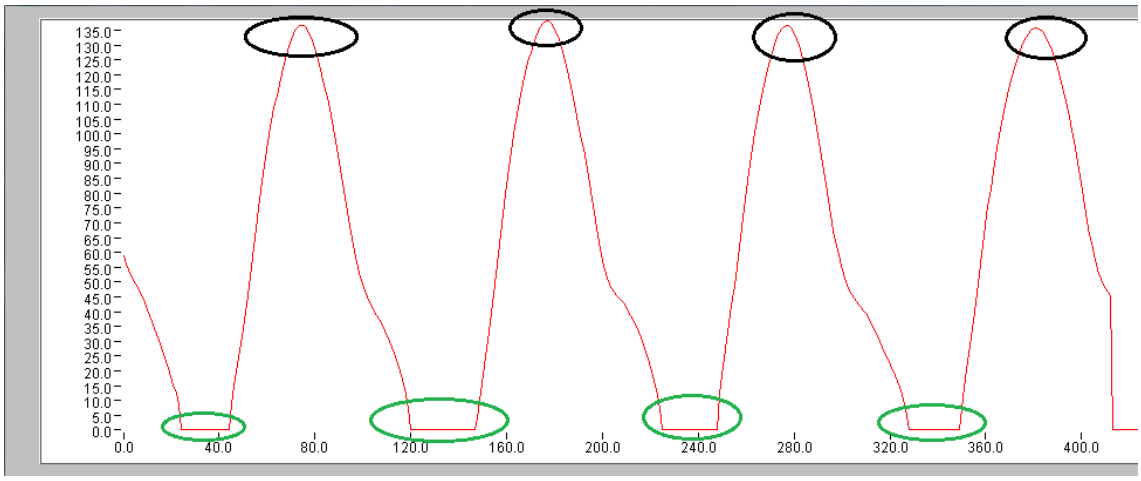
As it can be observed from the above table results are quite better but still there are some problems due to noise. Velocity check step compares the magnitude of joint's velocity with the other joints: for instance if left heel velocity is slower than the right heel and the right toe, then left heel is probably in a stance phase. But *left foot* can be in *stance phase* because of *left toe,* on the other hand *left heel* can be off. For this reason we can use threshold to partition a *foot stance* interval to *heel stance* and *toe stances*. Still using more constraints can give poor results since this method is based on using a threshold for vertical displacement and instantaneous velocities of joints. These values are usually noisy.

4.1.1 Using Knee Rotation Angles

For a cyclic motion we realized that it is possible to use knee movement to predict foot stances. In asf/amc file format (as it is in real life) knee has only one DOF which is rotating around *"x"* axis. For instance in a normal cycle at the time when left leg stretches most, *right heel* strikes to the ground and after a short while *right toe* stance occurs. In addition when left knee has the maximum *"x" axis* rotation value, both of *left foot joints* are in stance phase. It will be the same for the left leg: when left knee has the minimum *"x" axis* rotation value, right foot enters the *right heel stance* phase. And a few frames before this left *heel off* event occurs. Following the *right heel stance,* a few frames later *right toe stance* and *left toe off* will occur. And when left knee comes to its maximum rotation value both of the joints in right foot are supposed to be planted. The maximum rotation degrees can be used to predict the middle of the stance interval.



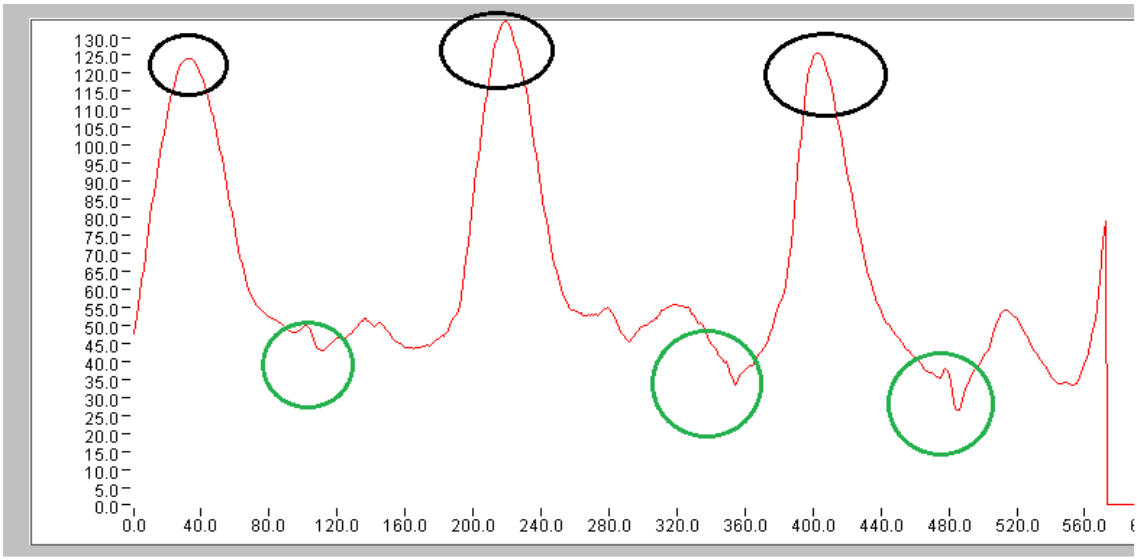
Left Knee



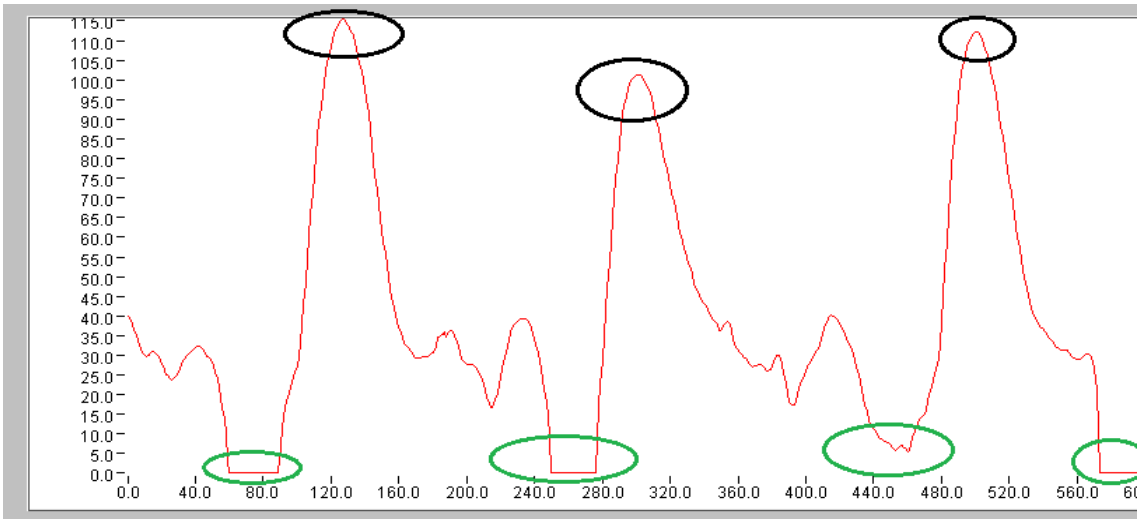
Right Knee

Figure 4.5: Knee rotation values in X axis for marching_01.amc motion

As it can be seen from figure 4.5, detecting the *foot stance* intervals is very simple. But still *marching* motion is a bit complicated as stated in the previous section that *toe strikes* occur before *heel strikes.* We believe that when we are blending motions or fixing the foot skating artifacts with IK algorithms [30] replacing *toe strikes* with *heel strikes* (if they come true before *heel strikes)* will not create remarkable artifacts. Since we believe that heels are already in contact with the ground but they are a bit higher, this artifact is negligible compare to foot skating problem. More over in the *FootSkate Solver* [30], if the end effector cannot reach a point limbs are extended up to %3 of their initial lengths. Length change in human limb is undetectable even for observers [61]. But for the most of the motion types curves of knees do not have perfect shape as figure 3.5.



Left Knee



Right Knee

Figure 4.6: Knee X Axis rotation degrees for drunk_01.amc with 584 frames.

From figure 4.6 we can see that detecting maximum picks is straight forward but it is not that much easy for *minimums (maximum stretch of leg)* since between two picks there can be several local minimums. For detecting the maximums we put a window let's say with a size of *l, l/2* key frames on the left and right checked for proving if our seed is the local maximum or not. If it is not, for the next iteration we continue from the biggest value key frame which is on the right of our seed. If it is we save it to the *maximum candidate list*. For detecting local minimums the method is similar: we put a fixed size of window and checking the values on the left and right of the window center. When a local minimum is detected then this key frame is saved in to the *mini-mum candidate list*. After this step we match each *maximum candidate* with a *minimum candidate* based on the idea a *local maximum* has to follow a *local minimum.* Following this a new list is created which holds *local minimum and maximum* pairs. A threshold (taking the mean of the differences) is applied to this list and the pairs which have lower difference values are removed from the list. In addition to this, more constraints have to be added into the candidate elimination step for the case of having a data like *marching_01.amc.* In this motion (figure 3.5) all the maximums and minimums are what we are looking for. So mean value of local maximums will be very high and most of our picks will be removed. Because of this case, we used the simplest way by putting a threshold such as *15 degrees* between maximum pick pair and *mean value*. If the difference between maximum and mean value is lower than 15 degrees, candidates which are lower up to *30 degrees* from maximum one are selected. But this can give poor results for the motions which are including several types of walking (marching and elderly). But since we are just using one type it is successful enough. For this kind of motions pairs can be grouped and the order of the local maximums and minimums can be considered for the solution.

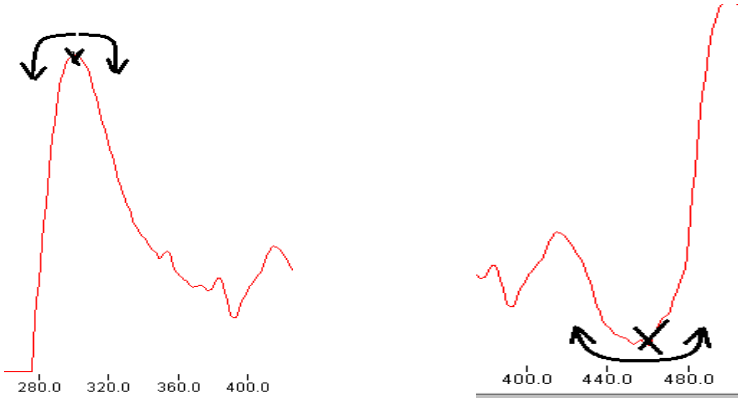


Figure 4.7: Finding local min-max by a window centered at a key frame, drunk_01.amc

We realized that first applying Fast Fourier Transform to the knee joints and removing high frequency signals will increase the correctness of above algorithm.

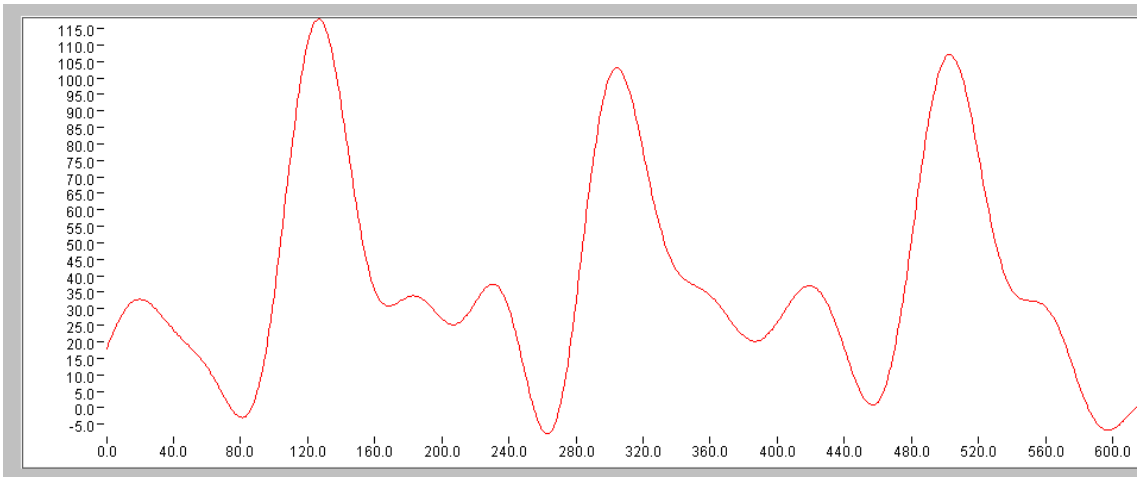


Figure 4.8: Right Knee rotation of drunk_01.amc after removing small frequencies

As it can be seen from figure 3.7 there were some small cavities around frames 350 and 400. After applying *fft* we removed small frequencies. We put a %10 threshold for removing high frequencies. This threshold compares the magnitude ratios on the frequency spectrum. As it can be seen from the figure 3.8 we have less noise and a better configuration.

Since we have a user scripting interface we tried to give example sets to the system for predicting the stances. And considering this intervals system tries to predict a threshold for vertical displacement. But the results are not much better than what we got from automatic prediction algorithm which is told in the first section.

>left_stances/right_stances motion_name strike_key_frames

eg: left_stance drunk_01.amc 0 8 15 25 50 52 110 115 120 130 160…

Using the scripting interface it is possible to correct *automatic foot plant detection* results such as:

>left_heel_strikes motion_name intervals

Eg:>left_heel_strike drunk_01.amc 0 52 110 160 222 270

>left_toe_strikes …

>right_heel_strike …

>right_toe_strikes …

4.2 Motion Blending

As it is described in *chapter 1* and *chapter 2* it is possible to blend two motions when the stance intervals are the same. For blending we merged *heel* and *toe stances*. Then using *left foot stance* and *right food stance* intervals motion is interpolated to a *single left stance* interval and *right stance* intervals. By using the scripting interface user can load these saved *averaged motions* and blend each of them with *other averaged motions* in different ratios. After a motion capture data is imported and stance intervals are set, by scripting interface user can blend *motion capture* data with averaged motions with different ratios. For interpolating the values we used quaternions and spherical linear interpolations (SLERP). We consider the motion clips are compromised from similar motion patterns. For instance the *sad motion* should have similar patterns for each joint during the whole motion. Since each one of the stance intervals can be in different size, we have to interpolate these intervals by mapping on each other.

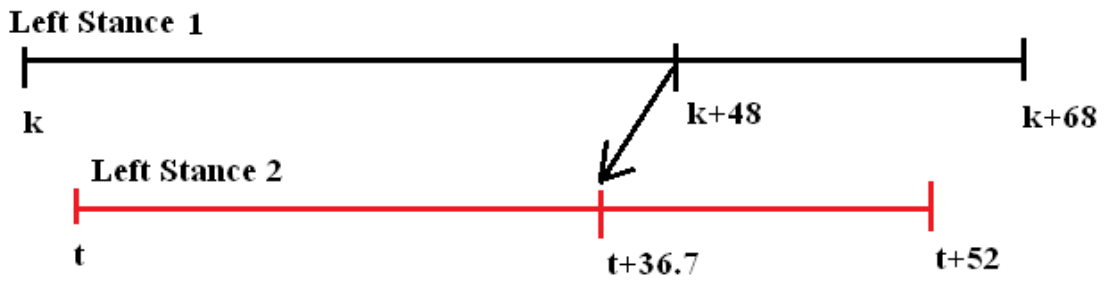


Figure 4.9: Mapping one stance interval onto the other one

After computing the equivalence of one key frame from one stance on the other stance which is being interpolated with the first one, if it is not an integer we have to interpolate the key frames on its left and right to find its value at that time. For instance the ratio for this interpolation in the above figure for *t+36.7* is *0.7* (slerp(t+36, t+37, 0.7)).When this value is computed the result is interpolated with the key frame on first stance interval (slerp(k+48, t+37.6, 0.5)).

In our implementation two lists are created: one for *left stance* intervals, and one for right stance intervals. Each interval is a list of quaternions for each joint. Consequently, these interval lists are interpolated as explained in the previous paragraph. Then the result is saved into a file. For blending one of normal animation motion with one of these files user has to enter the *ratio* and the body partition. Amc skeleton is partitioned

in our implementation for this work into: *lower body (LOWER), upper body (UPPER), left leg(LEFT_LEG), right leg (RIGHT_LEG), left arm(LEFT_ARM), right arm (RIGHT_ARM), spine(SPINE), head(HEAD).* The upper capital words are the keywords for our scripting interface. The configuration of the amc/asf skeleton can be checked from figure 2.10. Partitioned skeleton configuration can be found in figure 3.10.

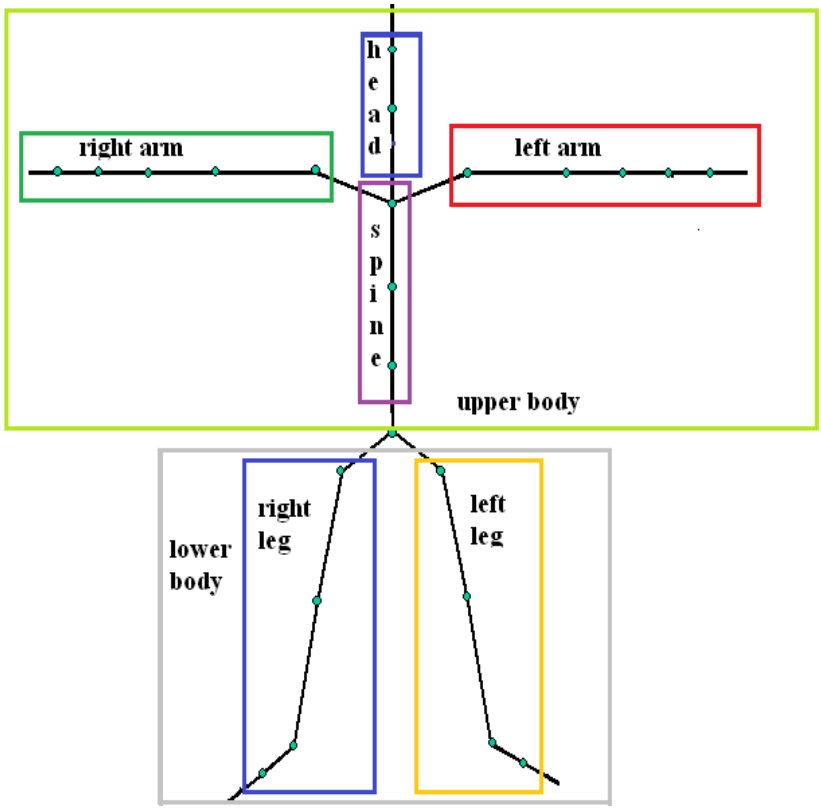


Figure 4.10 Partitioning skeleton into groups

Figure 4.11: Marching animation (left_stance, double stance, right_stance)

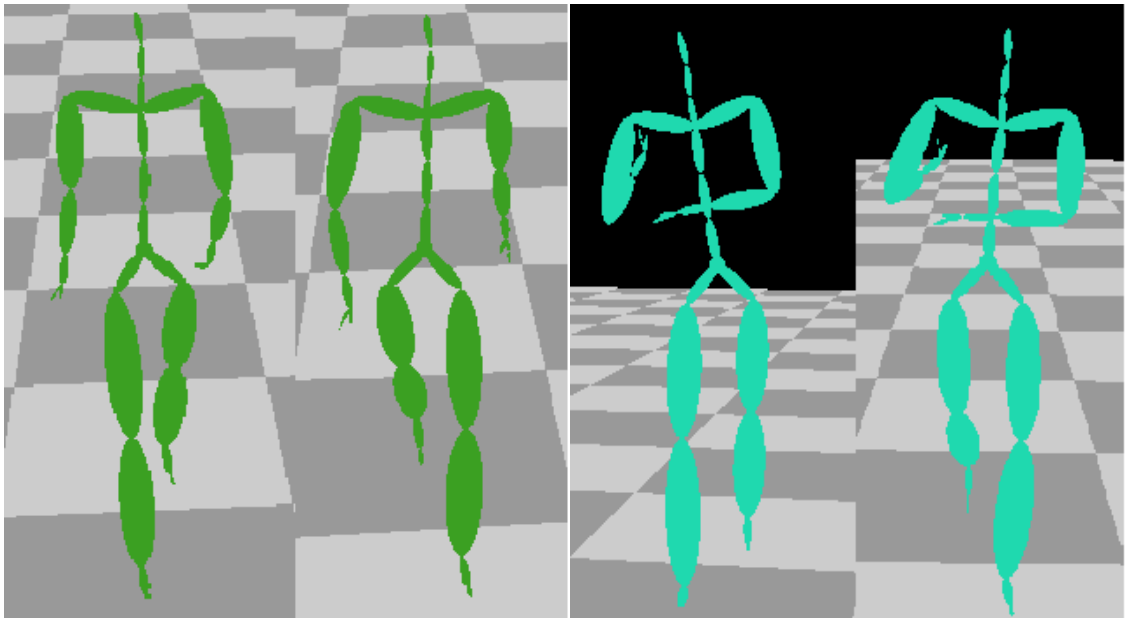


Figure 4.12: Normal walking cycle (left), blending result of marching and normal walking cycle (right)

Since we are blending a character considering the average of an animation for a cyclic walking animation there can be some fast/odd changes during the foot plant transitions. This is due to the fact that in original motion the stance intervals are not equal and movement in each stance can be quite different from each other. To smooth the movement of the joints (spatially on the foot stance changes) after the blending, interpolation (Slerp) is applied to the result animation. For improving the smoothness quaternion splines can be applied to the final motion [62]. Conversely if the original motion, which is averaged, repeating itself perfectly on each cycle, blending other motions with this kind of *averaged animations* will increase the visual quality.

If lower body is included to the blending, root rotation and translation has to be considered. If in the *averaged motion* root's total translation in a single step is bigger than other motion (which is being blended) then both of the legs will seem that they are coming back of the root. Conversely, if total translation of root joint during a single step is shorter than blended motion's root translation than root will always come back after the feet. Because of this for the root displacement in a step we consider the ratio between *averaged motion* and *blended motion* as a multiplicand for the displacement between each frame. Moreover we add another multiplicand which is given as blending ratio. A part from this root rotation has to be considered as a spatial case. Its rotations

are defined considering the sagittal, coronal, and traverse planes [63]. If the root rotations are directly interpolated (without using Slerp) in the result animation character will be facing into another direction which is different from its moving path. Mostly it will result in abnormal rotations such as tilting all body to left/right or to back/forward. For this reason at the beginning we hold the changes in x, y and z orientations. Then we interpolate these differences, after having the interpolated differences for each *key-frame* of the root joint, we add them incrementally one by one to the initial orientation. Because of the fact that root is not interpolated correctly in this we have had a tilting character during the whole animation. This is due to the fact that the rotation difference between consecutive key-frames around any axis can be very distinct in two motions.

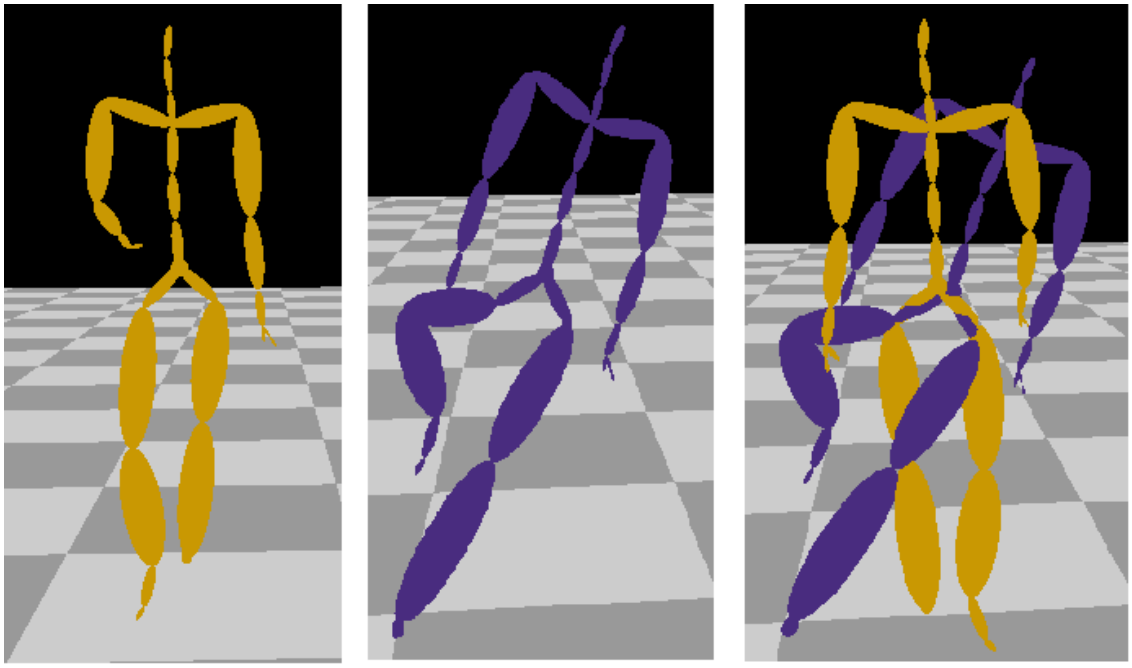


Figure 4.13: Normal walking character (right), blending result for lower body [$slerp(lower\_body_{normal\_walk}, lower\_body_{marching\_walk}, 1.0)$] (middle), both animations at the same key-frame (right)

For the above result we interpolate all the lower body joints using *slerp* except root joint. Root joint rotations for each key-frame are computed with the method which is explained in the previous paragraph. After checking the marching root joint, we observe that x and z axis rotation angles are changing so much between consecutive key-frames compared to normal walking animation. Since these differences are interpolated and added to the initial rotation of the root joint, character will be unbalanced in x and z axis.

On the other hand with this method works *y* axis since char-acter's orientation is overlapping with the path it is following.

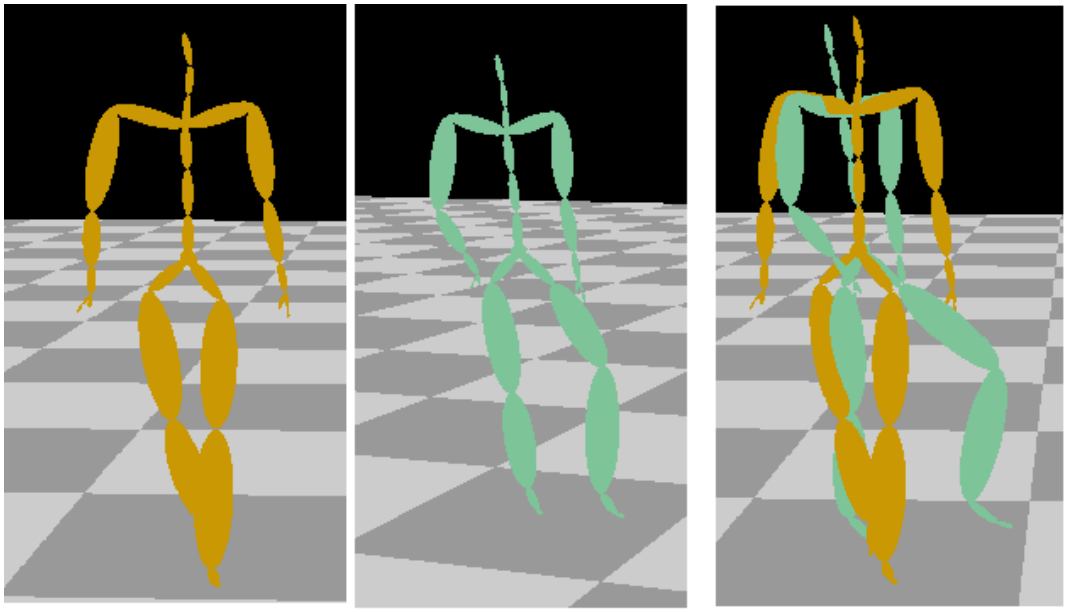
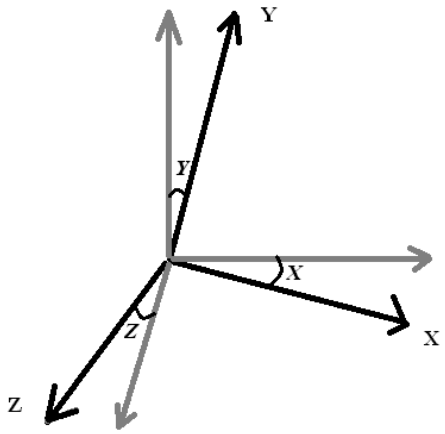


Figure 4.14: Normal walk (left), blending result for lower body [$slerp(lower\_body_{normal\_walk}, lower\_body_{marching\_walk}, 0.5)$] (middle), both animation at 46th key-frame (right)

After computing the interpolation using *slerp* for root joint, whole body tilting artifacts are disappeared. But now character orientation and path are not overlapped as in the last figure. For this reason at each *key frame* we compute rotation differences in *y* axis between the original motion and the blended motion. Then root joint is rotated that much in *y* axis. For computing this rotation:



First we compute a rotation matrix where local frame of root aligns with vertical axis of the global frame.

$$new\_Z\_axis = [0.0,1.0,0.0] \times old\_X\_axis$$

This cross product will give us new *Z* axis.

$$new\_X\_axis = [0.0,1.0,0.0] \times new\_Z\_axis$$

Using the new Z axis we can compute the new *X* axis. So this frame should have a rotation only around *global Y* axis. This rotation can be calculated as:

$$\sin\theta = \frac{\|[1.0,0.0,0.0] \times new\_X\_axis\|}{\|1.0,0.0,0.0\| \|new\_X\_axis\|}$$

Let's call this rotation (only around global y axis) matrix K and root's global rotation matrix M. Another matrix, which aligns root local frame with this frame, can be computed by $KR = M \Leftrightarrow R = K^{-1}M$. To find the angle that the blended motion's root has to rotate another coordinate frame has to be computed for the original motion. Using these two coordinate frames, for which vertical axes are aligned with global vertical axis, the angle can be computed as on the above with cross product. Let this rotation matrix be R_y. All the matrix multiplications should be like this $M \times R \times R_y \times R^{-1}$.

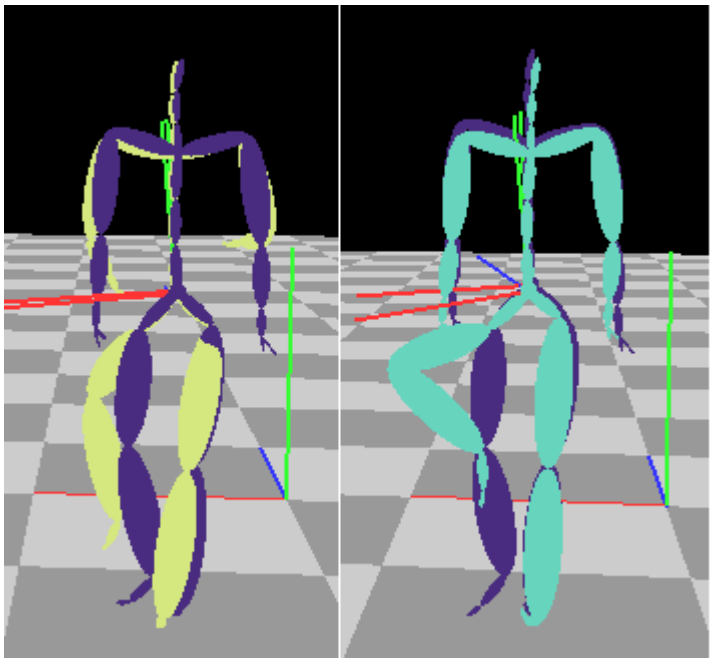


Figure 4.15: Blending all body of a normal walk with marching ($slerp(all\_body_{normal\_walk}, all\_body_{marching\_walk}, 0.5)$ (left), blending lower body $slerp(lower\_body_{normal\_walk}, lower\_body_{marching\_walk}, 1.0)$ (right)

After correcting the root rotation, if the lower body is included to the animation, foot skating artifacts have to be handled. For foot skating correcting *FootSkate[30]* Solver can be used. The command window keys for the blending are:

>right_stances/left_stances motion_name first_footstrike_keyframes

eg: left_stances marching_01.amc 65 195 320

>set_name averaged_motion_id new_name

eg: set_name 1 sad_walk

>print_averaged_motions

>blend_motion motion_name(mocap_name) averaged_motion_name body_part ratio

eg: blend_motion marching_01.amc sad_walk ALL 0.5

All body partition keywords are given in page 54 which are: *LOWER, UPPER, LEFT_LEG, RIGHT_LEG, LEFT_ARM, RIGHT_ARM, SPINE, HEAD.*

4.3 Motion Path Editing

Given the path of an initial motion it can be associated with a curve which is the smoothed version of the original path [58]. This curve can be altered to make the motion following the new one. To define this curve we used Bézier curves. In cubic Bèzier curves four points are used, $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. Curve starts at $\mathbf{P}_0$ going toward $\mathbf{P}_1$ and arrives at $\mathbf{P}_3$ coming from $\mathbf{P}_2$. Usually, it is not passing through $\mathbf{P}_1$ or $\mathbf{P}_2$; these points are only there to provide directional information. The distance between $\mathbf{P}_0$ and determines "how long" the curve moves into the direction $\mathbf{P}_2$ before turning towards $\mathbf{P}_3$. The parametric form of the curve is:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3, \quad t \in [0,1]$$

Using Bezier cubic Bezier curves have some disadvantages. Trying to represent all of motion path by a single cubic spline will cause loss of detail and control. On the other hand if we use more than one cubic spline to have more control on path adjusting we will have C^1 discontinuity problem between the consecutive curves at the points they connect to each other. For this reason Hermite Curves can be used. But in this type the splines curve is passing through all the control points which won't be smooth [10]. To have a smooth curve we selected generalized Bèzier curves. Given points $\mathbf{P}_0, \mathbf{P}_1 \ldots \mathbf{P}_n$ the Bezier curve is defined as:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i = (1-t)^n P_0 + \binom{n}{1} (1-t)^{n-1} tP_1 + \ldots\ldots + \binom{n}{n-1} (1-t) t^{n-1} P_{n-1} + t^n P_n,$$

$t \in [0,1]$, where $\binom{n}{i}$ is the binomial coefficient.

To define this curve we select each 15th key frame as a control point. If our first frame is *k* (P_0) and last one is *k+r* (P_n), we have to compute $B(t)$ for every key frames in this interval. After this step we will have the corresponding 3D coordinate of the path per each key-frame. Following this, to manipulate the path of the motion we implanted Hermite curve on the motion path, and for every *400 frames* we put *five* control points.

When the user changes one of the control points, *motion path* is updated. And we compare each point on the *motion path* considering transformations in *X* and *Z* axis. The displacements in *X* and *Z* axis are added to the root positions for the corresponding key-frame. To find the rotation around vertical axis (*Y*) we used a similar method as in the previous section. For each key-frame we compute two local frames using vertical axis (0, 1, 0) and tangent vectors for the key-frame on previous path and updated path. Cross product of the vertical axis and tangent vectors will give the *Z axes* in two local frames. *X axes* can be found by cross product of *Z and Y axes*. Using these axes we can construct the global rotation matrix for these two different reference frames. To find rotation around Y axis: $\mathbf{M}_1\mathbf{M}_{\mathbf{rot\_y}} = \mathbf{M}_2 \Rightarrow \mathbf{M}_{\mathbf{rot\_y}} = \mathbf{M}_1^{-1}\mathbf{M}_2$ is computed, where M_1 and M_2 are global rotation matrixes and M_{rot_y} has component of *Y axis* rotation.

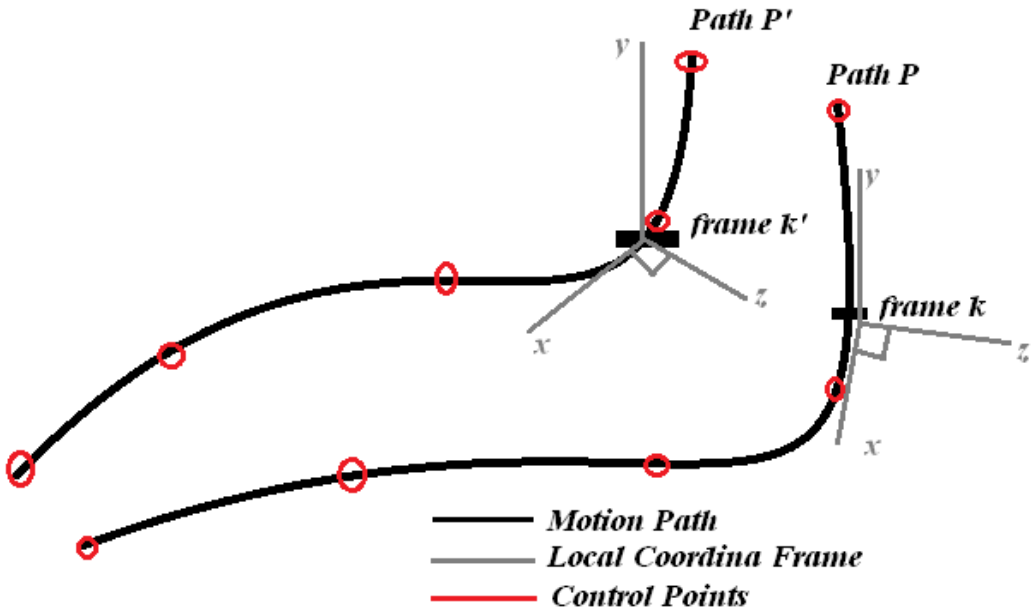


Figure 4.16: Changing motion path with Hermite splines

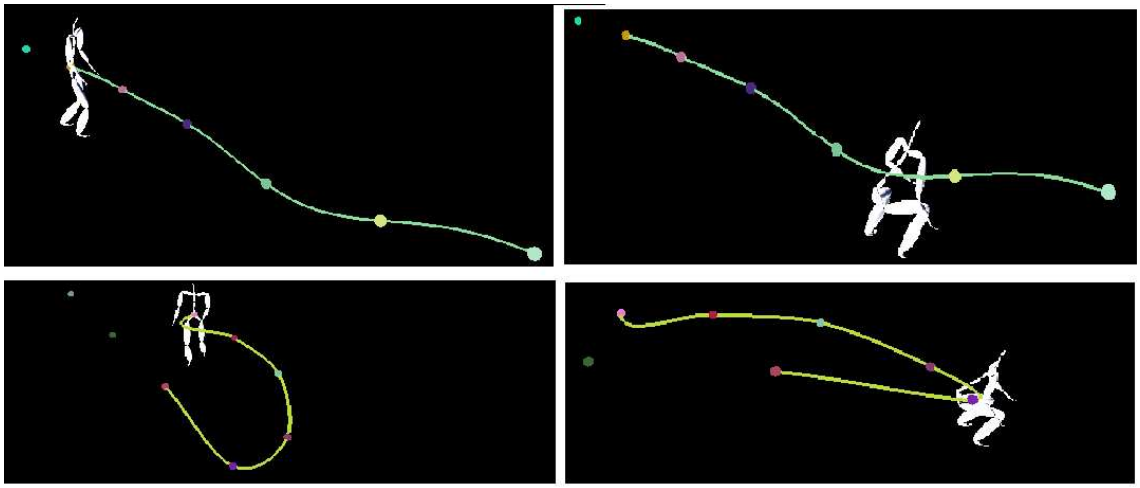


Figure 4.17:First raw depicts the skateboard duck under motion for initial frame and for frame 235. On the second raw motion path altered using the control points (colored ones).

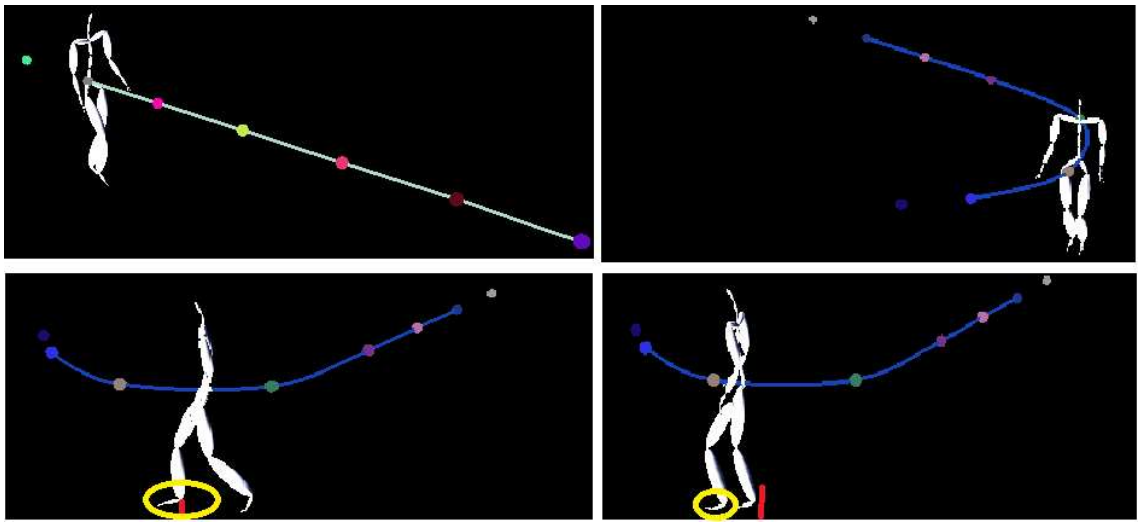


Figure 4.18: Changing the path of normal walk motion (first raw), Right foot stance (second raw). Red line shows the right heel strike position. Right foot is put inside a yellow circle.

In the above figure we have the result of altered path for the normal walk motion. Between key-frames *324* and *384* right foot plant occurs. In the initial motion there is no *foot sliding* artifact in this interval. Since there is a distance change between two consecutive control points it causes *foot-skating* artifact. For this reason we tried to use a library, *FootSkate Solver[30]*, to get rid of this problem when a walking motion's path is edited. To use this library the *foot-plant* intervals for the initial motion have to be known. In our framework this intervals can be entered manually via command window or there is an automatic check for each imported motion. In this library eight constraints can be defined, one for each heel, for toes, for wrists and for fingertips.

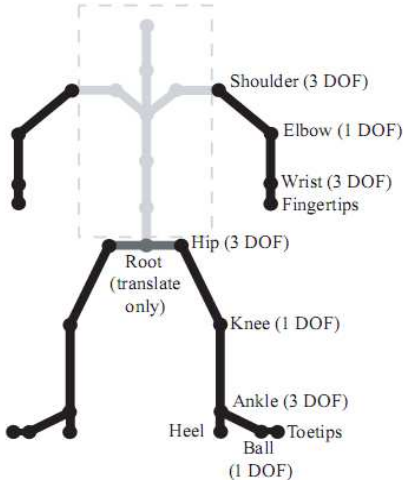


Figure 4.19: Skeleton Model for End Effector Cleanup [30]

As in the figure 3.19 the hierarchy of the skeleton in *Foot Skate* solver has more DOF for *wrist* and *ankle* joints compare to the skeleton of the asf/amc motion capture data. As it can be seen from the last figure this library is just trying to correct the lower movements of arms and legs. So we can't put any constraints to the spine and head. In our case we are using it to correct the lower body motion problems, so we do not need to put any constraints to the arms. When a motion is imported into our framework initially all the global rotation and global position of the joints are computed. And after any processing on the animation with path editing or motion blending methods these information is recomputed again. When there is a need to use this library this information is directly copied to FootSkate solver. The root orientation is not changed. Foot-Skate solver chops the constrained joints and puts them to the constrained position. After placing the constraint joints it fixes the root position using the length information of legs that they can reach. The position of the root is computed using spheres. More generally for every key-frame if *n* end effectors are constrained, then root has to be within the intersection of these spheres. If this is not true, then root must be projected onto the surface of the intersection region. Since we have constraints for left foot and right foot it will solve root placement problem using two spheres.

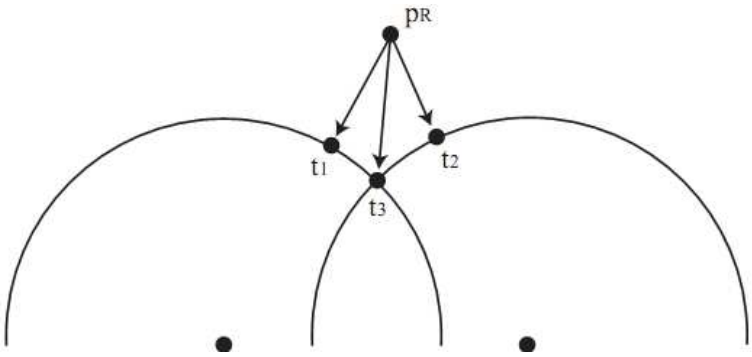


Figure 4.20: A constraint on a foot will make the root the projection closer to *t1*, if the other foot is only constrained one root's projection will be closer to t2. On the other hand if both of them are constraint root projection will be at t3 [30].

Since the root position is computed for each key-frame it will have some visual artifacts for this reason a Gaussian smooth applied to the result. After having the position of the root and position of the constraint legs it fixes the *tibia* and *femur.* We can imagine that there is a circle which diameter is fixed between the constraint joint and hip. We can find the orientation of this circle from the constraint joint. When we have the orientation we can find the knee rotation from *cosine law.*

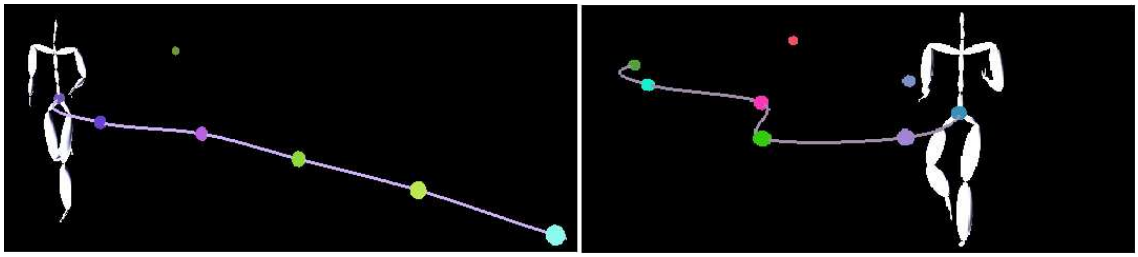


Figure 4.21: Straight running animation before path editing (left) and after path editing (right)

To use the path editing the following command can be used:

To compute the path of the animation:

>*root_path motion_name*

eg: root_path run.amc

To display the path of an animation if it is already computed:

>*display_root_path motion_name*

To set the constrained joints manually:

>*left_heel_strikes/left_toe_strikes/right_heel_strikes/right_toe_strikes motion_name intervals*

eg: left_heel_strikes run.amc 0 30 105 135…

4.4 Motion Editing with Signal Processing Methods

The first technique that we use to process the motion is fast fourier analysis method. Details of the implementation of *FFT* algorithm can be checked from [42]. The idea for using the *fft* was for detecting the period of a cyclic motion as it has been told in [27]. *Fundamental frequency* is the one which is the biggest in amplitude in the frequency spectrum. After detecting the fundamental frequency we can put a threshold to remove the frequencies which are low compared to *fundamental frequency.* In this way we can have smooth motion signal for a character with fewer local maximums and minimums motion signal for each joint. Then motion signal for some key joints can be exported for some tools which are imitating key poses such as *Simbicon* and *Dance*. In our framework we used *fft* to change the joints motion individually or to lower or exaggerate all body motion. After a motion is imported into our framework user can select to compute *fft* of all the body motion and then each DOF of joints can be manipulated from joints window. Or a threshold can be put into all DOF of joints from motion window. But any change to the lower body and root joint can result in visual artifacts. Since humanoid characters are represented as a hierarchy any change in the upper part of the hierarchy can cause abrupt motions in the lower parts. In this sense it is very difficult to reproduce a new motion by manipulating the signals of lower body that we have to find a good synchronization between the translation and rotation of root joint and legs' joints rotations by playing signals. Unsynchronized root translation and joint rotations will cause *foot-skating problems.* In addition to these *fft* can be used for smoothing the motion, spatially for the joints which have some noise at some key-frames.

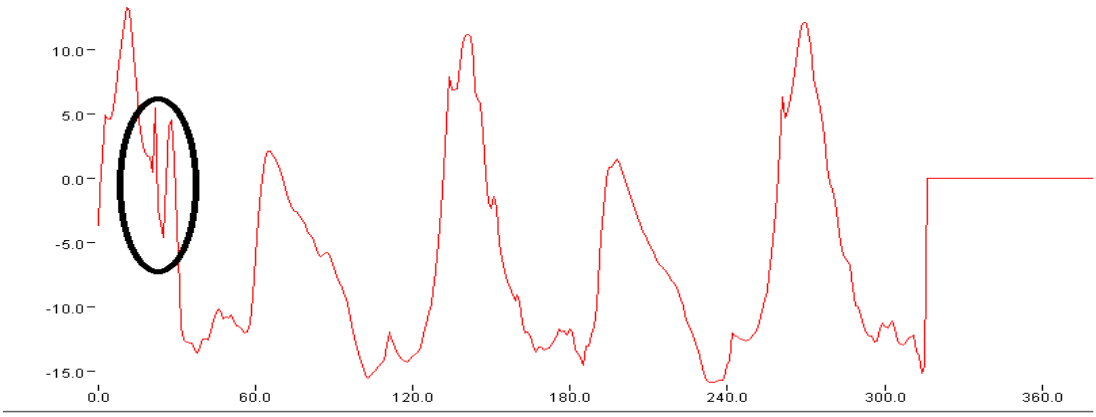


Figure 4.22: *X axis* rotation of foot joint in a normal walk animation taken from a motion of *CMU* motion database.

Above figure depicts to *X* axis rotation of a joint in a normal walk. There are some noise data which can be realized from the graph. But most of these abnormal movements are not realizable when we play with the motion. But on the above figure around key-frame *25* there is an odd change and it is obvious when we watch the character movement in real time.

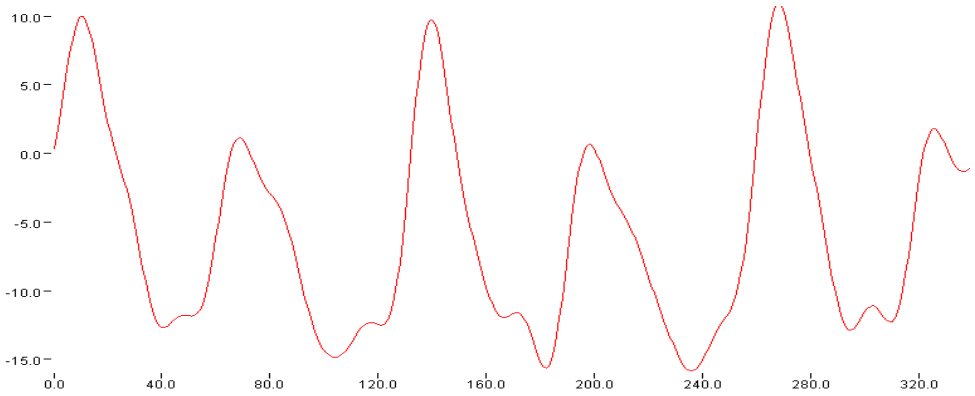


Figure 4.23: *X axis* rotation of the root joint after smoothing the signal.

We removed all the signals from the initial one which are smaller than %10 of the original motion's *fundamental frequency* in magnitude. After smoothing the signal as in the last figure the vertical distance between the local maximums and local minimums decreases. For instance in our example this will cause to foot to rotate less in the *x axis*. To fix the rotation to previous range remaining frequencies in the spectrum can be multiplied with a constant.

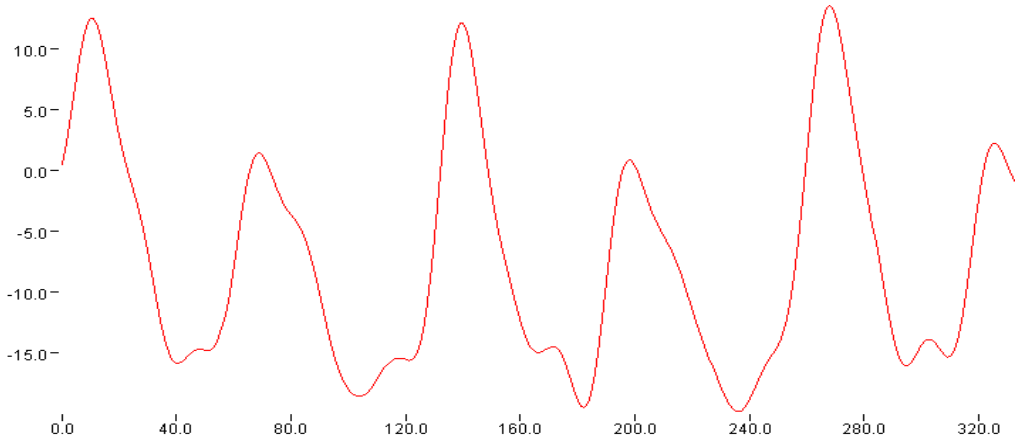


Figure 4.24: After multiplying all the frequencies in the new signal with 1.25. Local max and local min are closer to the original motion.

In addition to removing noise from the joints using *fft* we have also tried to change the motion by multiplying each joint DOF by a constant. This constant has to be same for every DOF. After this some joints have to be corrected. For instance when the root's transformation signals are multiplied, character will appear in a different vertical coordinate. Furthermore changes in the character root orientation will rotate and make the legs collide with each other when the one foot is moving and other one is planted.

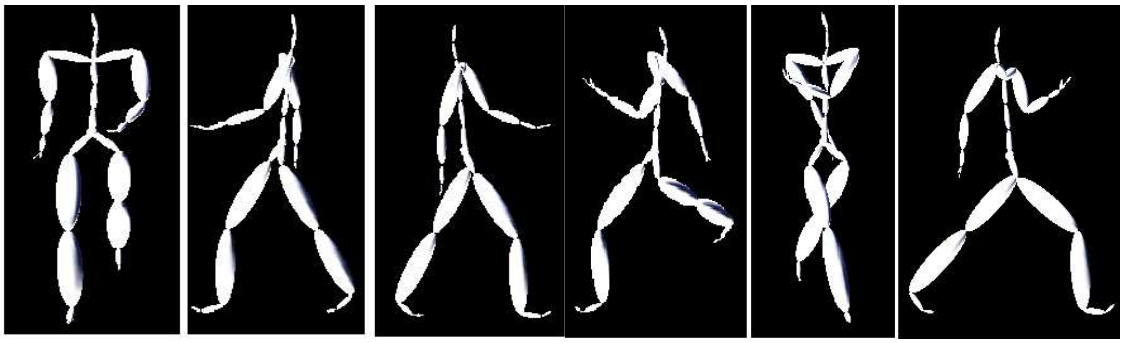


Figure 4.25: First three frames are the 1st key-frame of the initial walking animation. Second frames screenshots are the 1st key-frame from front, left and right when the initial motion signals are multiplied with *1.5*.

As it can be seen from the above figures femurs and shoulders of the character are rotated so much that during the animation limbs will pass through each other. For this reason when the motion weight is changed we correct the rotation values of femurs and shoulders. In general asf/amc file has a constraint for the rotation limits for each *DOF*. For instance from the observations of biomedical knee rotation limits is [-10, 170] and it has only one DOF. But after multiplying the knee *X axis* with different weights we can have new values that can be out of these limits. This is very disturbing when it occurs in elbows and knees. For this reason after each manipulation in the motion for each key-frame *X axis* rotations are checked to see if the limits are violated. When this case occurs then they are fixed to the limit angles for the corresponding key-frames.

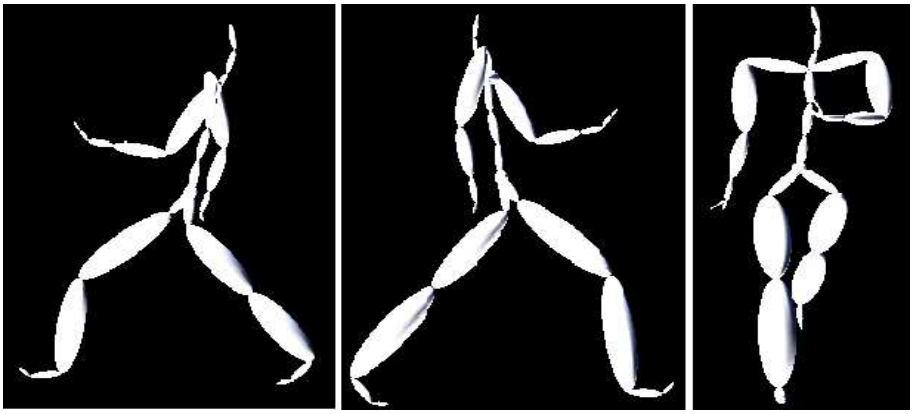


Figure 4.26: After the femur and shoulder joints are corrected.

Other method to enrich the humanoid motions as in [53] is using multi-resolution filtering. In the first step image lowpass pyramid is obtained by convolving the image with a *B-Spline* filter kernel and at each level this image is sub-sampled by a factor of two as in the below figure. Let's the original image be $\mathbf{G_0}$ and to final image be $\mathbf{G_n}$. You can see the lowpass pyramid in the below figure. This processing is repeated until the image size is reduced to one pixel, which has the average intensity. Then we compute the band-pass pyramid by taking the difference of two successive Gaussian pyramid images. Let's call these images $\mathbf{L_{n-1}} \ldots \mathbf{L_0}$. Then the original image can be constructed by: $\mathbf{G_0} = \mathbf{L_0} + \mathbf{L_1} + \ldots + \mathbf{L_{n-1}} + \mathbf{G_n}$. This is applied in [53] to the articulated motions. Here *DOF* treated as a one-dimensional signal from which low-pass (G) and band-pass (L) levels are computed. Let's say we have *m* frames then we should have *n* ($2^n \leq m \leq 2^{n+1}$).

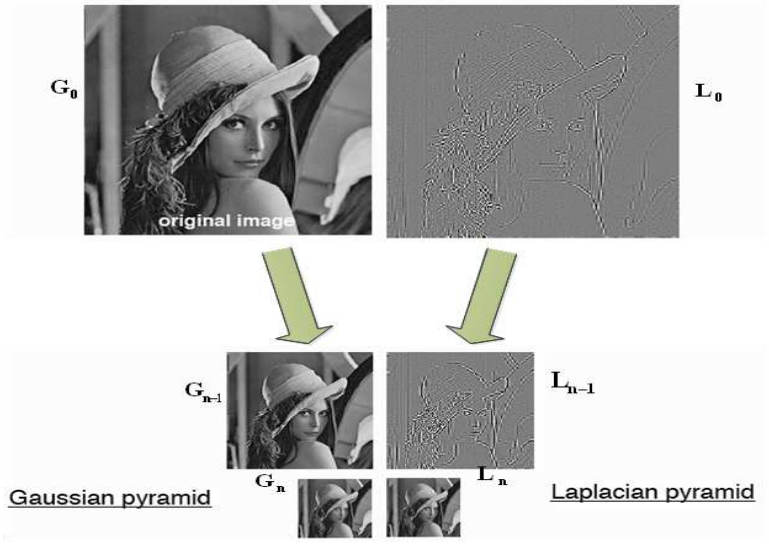


Figure 4.27: Low pass pyramid (left), band pass pyramid (right)

Instead of constructing a pyramid of low-pass and band-pass sequences where each successive is reduced by a factor of two, alternatively the sequences are kept the same length and the filter kernel (ω) is expanded at each level by inserting zeros between the value of the filter kernel (*a, b* and *c*). For instance with the kernel of width 5,

$$\omega_1 = [c,b,a,b,c]$$

$$\omega_2 = [c,0,b,0,a,0,b,0,c]$$

$\omega_3 = [c,0,0,0,b,0,0,0,a,0,0,0,b,0,0,0,c]$, where a=3/8, b=1/4, c=1/16.

Computing motion multi-resolution has five steps:

1. Compute low pass sequences considering *n*:

   $$G_{k+1} = \omega_{k+1} \times G_k$$

2. Compute the band-passes:

   $$L_k = G_k - G_{k-1}$$

3. Adjust the gains for each band (L_k).

4. Reconstruct the motion signal.

   $$G_0 = G_n + \sum_{k=0}^{n-1} L_k$$

After computing these bands we put sliders into the user interface to change the gains and for producing a different motion. According to our results it produces good motion for the upper body, but for the lower body spatially if there is a change in the femurs or root joint visual artifacts appears. This is mostly due to synchronization problems of the legs and the time when feet hit to the ground. In this phase Foot Skate solver can be used.

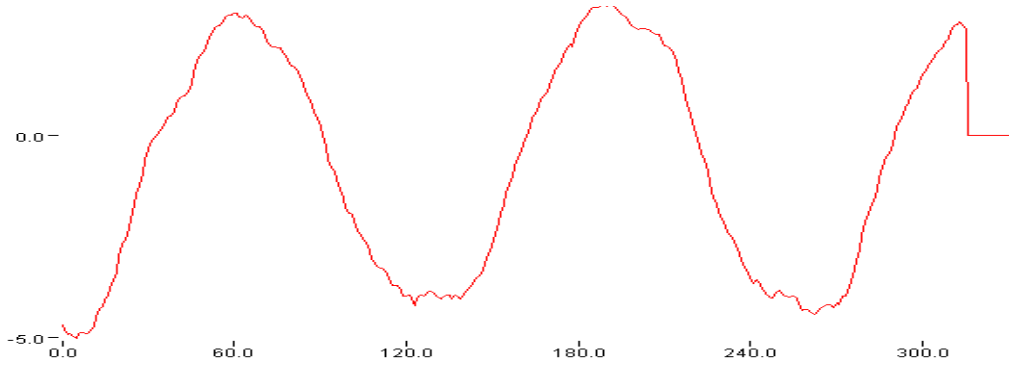


Figure 4.27: Rotation around *Y axis* of lower back joint in a normal walk.

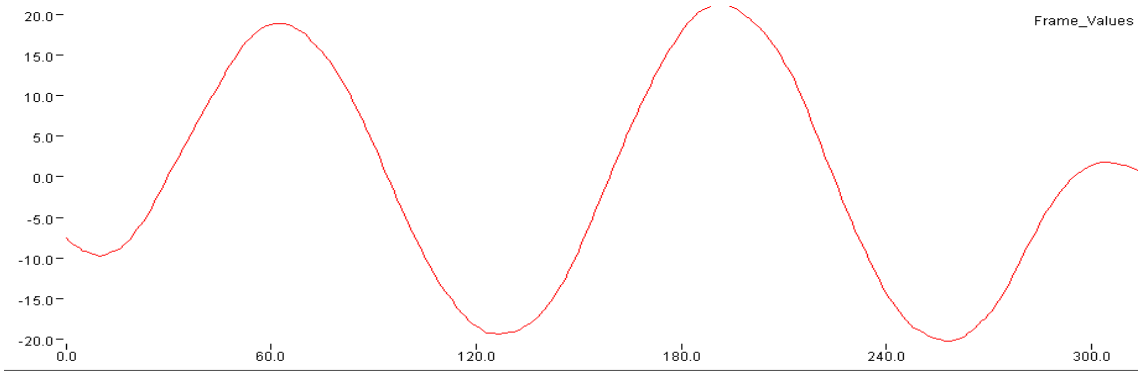


Figure 4.28: Rotation around *Y axis* of lower back joint when we put 7.58 to band 6, 7 and 8 in a normal walk.

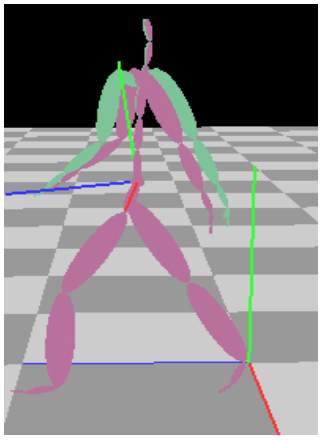


Figure 4.29: In the same key-frame normal walk (purple), and after manipulating the lower back joint band passes with about coefficients.

As it can be seen from the above figures this method gives a possibility to process the motion, but it is not straight forward. Animator has to play with all the bands to have a good result.

4.5 Predicting the Period of a Cyclic Motion

As it was told in the previous section fundamental frequency of a signal can be used for predicting the period of a motion. Fundamental frequency is the biggest one in the spectrum considering the magnitudes. In [27] they used it for detecting the period of a motion. But according to our experiments in most of the cases it is difficult to get a

correct estimation. For instance for a normal walk (07_01.amc in CMU database) we checked the fundamental frequencies for key joints:

Table 4.6: Fundamental frequencies for some joints of a motion with 316 key-frames.

Joint Name	**DOF axis**	**Fund. Freq.**
l_femur	*X axis*	4
r_femur	*X axis*	4
l_tibia	*X axis*	4
r_tibia	*X axis*	0
l_foot	*X axis*	8
r_foot	*X axis*	4
l_radius	*X axis*	4
r_radius	*X axis*	0
l_humerus	*Y axis*	4
r_humerus	*Y axis*	4

As it can be seen from the above table for some of the joints the fundamental frequency is different and it is more logical if we use knee and food joints to predict it. Even though the motion to prepare this table is almost a prefect cyclic motion we cannot predict the period from these table. For this reason it is told in the state of art chapter we used auto-correlation method to predict the period. When we multiply the signal with itself, second signal is sliding frame by frame, multiplication results are summed. After finishing all the iterations, results are normalized considering the biggest one. We get a result as in the below figure. In this graph we check the biggest drops between the local maximums and minimums. If there is a drop between the two big drops we eliminate it. According this method knee rotation of a different motion gives a result as in the below figure. The real period is around 140 and in the figure it is very close to it.

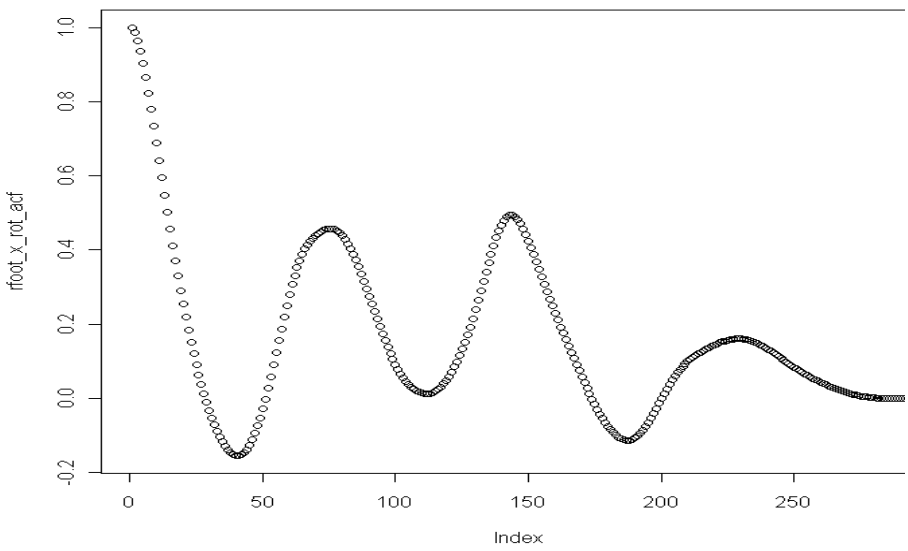


Figure 4.30: Result of autocorrelation method for knee rotation around *X axis*.

Table 4.7: Result of the autocorrelation method for same walking motion

Joint Name	**DOF axis**	**Period**
l_femur	*X axis*	126
r_femur	*X axis*	125
l_tibia	*X axis*	129
r_tibia	*X axis*	195
l_foot	*X axis*	129
r_foot	*X axis*	130
l_radius	*X axis*	123
r_radius	*X axis*	0
l_humerus	*Y axis*	119
r_humerus	*Y axis*	127

Table 4.8: Fundamental frequencies for some joints of a motion with 413 key-frames.

Joint Name	**DOF axis**	**Fund. Freq.**
l_femur	*X axis*	5
r_femur	*X axis*	5
l_tibia	*X axis*	5
r_tibia	*X axis*	5
l_foot	*X axis*	5
r_foot	*X axis*	5
l_radius	*X axis*	0
r_radius	*X axis*	0
l_humerus	*Y axis*	0
r_humerus	*Y axis*	0

Table 4.9: Result of the autocorrelation method for marching walking motion with 413 key-frames.

Point Name	**DOF axis**	**Period**
l_femur	*X axis*	101
r_femur	*X axis*	101
l_tibia	*X axis*	101
r_tibia	*X axis*	101
l_foot	*X axis*	101
r_foot	*X axis*	201
l_radius	*X axis*	0
r_radius	*X axis*	0
l_humerus	*Y axis*	0
r_humerus	*Y axis*	0

Table 4.10: Fundamental frequencies for some joints of a cyclic rush motion with 189 key-frames.

Joint Name	**DOF axis**	**Fund. Freq.**
l_femur	*X axis*	3
r_femur	*X axis*	3
l_tibia	*X axis*	3
r_tibia	*X axis*	0
l_foot	*X axis*	4
r_foot	*X axis*	3
l_radius	*X axis*	0
r_radius	*X axis*	0
l_humerus	*Y axis*	0
r_humerus	*Y axis*	0

Table 4.11: Fundamental frequencies for some joints of a rush motion with 189 key-frames.

Point Name	**DOF axis**	**Period**
l_femur	*X axis*	74
r_femur	*X axis*	71
l_tibia	*X axis*	73
r_tibia	*X axis*	41
l_foot	*X axis*	73
r_foot	*X axis*	73
l_radius	*X axis*	0
r_radius	*X axis*	0
l_humerus	*Y axis*	0
r_humerus	*Y axis*	0

As it can be seen from the above tables both methods are competitive. But most of the times it is difficult to predict the exact period with fundamental frequency. For instance for *fft we used* 256 width spectrum and from the table if we assume 3 is correct to predict it then our period has to be 256/3. But actually the period is 72 frames. In the marching walk animation we had zero values for upper body joints since these joints are not changing. According to our observations it is better to use lower body joints spatially the *femurs* and *tibias*.

CHAPTER V

CONCLUSIONS

We started by giving the definitions of different methods for processing the recorded animations and regenerating new motions. Since creating a new motion using the existing ones is saving a lot of time of animators we tried to develop and merge some of the exiting algorithms. On top of all of these methods there is a need to predict the foot plants. In our application motion path editing and behavior blending algorithms are applied considering the foot stances. For detecting the foot plants we checked the existing methods such as foot displacement and foot velocities. We have seen that this method in some cases gives poor results because of the existing noise in the data. And then we tried to improve the method by processing the knee rotations compared to each and as a result we got better result.

Afterwards we tried to enrich the motions by blending different motions with the current motion. For this method we need just one cycle of characteristic motion. These characteristic motions also can be blended with each other before blending a long motion with them. This method has good results for the upper body, but for the lower body generally it suffers from foot skating problem that can be corrected by using a library such as FootSkate Solver [30]. Since this method is not changing the motion path we implanted a method [58] to manipulate the character's motion path. We have seen that if the arc length between each control point increases foot skating problem occurs. Again for correcting the foot plants FootSkate Solver can be used or rotation of the femurs around Y axis can be weighted according the arc length change ratio. But both of these do not give a solution if the arc length changes are bigger than the foot extendable regions.

Following these methods we made a research on changing single motion by signal processing methods. First one is using Fourier transform. It gives a change to the animator to lower rotations of joints or exaggerate them. In addition to this we realized

that this method can be used to remove the noise in the motion which is realizable during the animation. Second method implemented by using multi-resolution technique that it created bands for each DOF. User can play with these bands and change the behavior of the joints. Higher bands represent the course grain motion signal. On the other hand these two methods suffer having a lot of parameters.

This study can be regarded as a preliminary work for motion processing. A number of practical issues, that are widely discussed in this thesis, need to be addressed in future work. An important one is the following: When a motion is modified with path editing method and if the arc lengths between the control points extended too much, inverse kinematics cannot give a solution to extend the legs to preserve the continuity. For this scenario motion graphs can be used as a solution.

CHAPTER VI

FUTURE DIRECTIONS

This thesis involves an overview of advantages and disadvantages of different methods for motion capture data processing. Beyond the analysis of existing algorithms, it proposes how to combine the good features of them to create a new motion from a single data or different motions by blending. Also it addresses a solution for estimating foot plants.

Considering our main results, this study will provide a background for a number of future directions.

Motions can be merged by improving our blending method. Since we know the intervals of the foot plants we can extend our implementation for detecting a good interval for transition from one motion to another. This can be done in a brute force way only considering the first motion foot stance at the end of the motion and the beginning foot stance in the second motion. But we think that this can produce some abrupt results. In this sense we need to use motion graph algorithms to find a good transition interval. We already tried to construct motion graphs but it is very slow if it is computed on CPU. Because of this we already started to extend our work by computing motion graphs on GPU.

There are many implementation issues that need to be addressed to have fast and robust algorithms to enrich the character animations. The methods implemented in chapter III and chapter IV can be seen as a starting work.

As it is discussed so far in this thesis we studied motion enriching problem. We believe that having motion graphs as future work will give more chance to enrich the humanoid motions.

REFERENCES

[1] “The Complete History of the Discovery of Cinematography”, http://www.precinemahistory.net/introduction.htm

[2] R. Balzer, *Optical Amusements: Magic Lanterns and Other Transforming Images, A Catalog of Popular Entirtainments*, Richard Balzer, 1987

[3] http://en.wikipedia.org/wiki/Kinetoscope

[4] http://en.wikipedia.org/wiki/Georges_Mèliès, http://www.videosurf.com/video/a-trip-to-the-moon-part-1-1218239?vlt=ffext

[5] http://en.wikipedia.org/wiki/J._Stuart_Blackton, http://www.videosurf.com/video/humorous-phases-of-funny-faces-9331028?vlt=ffext

[6] http://en.wikipedia.org/wiki/Felix_the_Cat

[7] http://en.wikipedia.org/wiki/Walt_Disney

[8] http://www.pixar.com/shorts/tt/index.html

[9]S. Roberts, *Character Animation in 3D*, Focal Press (2004).

[10] R. Parent, *Computer Animation: algorithms and Techniques*, 2nd edition, Morgan Kaufmann (2008).

[11] Florian Struck, *et al, Realistic Shading of Human Skin in Real Time,* Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualization and interaction in Africa, 2004, pp. 93-97.

[12] L. Kavan, et al, *Duel Quaternion for Rigid Transformation Blending*, Proceedings of the 2007 symposium on Interactive 3D graphics and games, 2007, pp. 39-46.

[13] Soha Moad, Saida Bouakaz. *Geometric Modeling of Moving Virtual Humans: A Survey of Various Practices,* Journées GTMG - Groupe de Travail en Modélisation Géométrique, 2004.

[14] Autodesk Maya, http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13577897

[15] Autodesk 3ds Max, http://www.autodesk.it/adsk/servlet/pc/index?siteID=457036&id=14642335

[16] Blender, www.blender.org

[17] Animeeple, http://www.animeeple.com

[18] Endorphin, www.naturalmotion.com/endorphin.htm

[19] Dynamic Animation and Control Environment, http://www.arishapiro.com/dance/

[20] Alexander Kort, "*Computer Aided Inbetweening*", In NPAR'02: Proceedings of the 2nd International symposium on Non-photo realistic animation and rendering, pages125–132. New York, NY, USA, 2002. ACM Press.
[21] *Real-time 3D Character Animation with Visual C++*, Focal Press (2001).
[22] *Understanding Motion Capture for Computer Animation and Video Games*, Morgan Kaufmann, New York, 2000.
[23] L. Unzueta, M. Peinado, R. Boulic, Á. Suescun, *Full-body performance animation with Sequential Inverse Kinematics*, Graphical Models. September 2008, pp. 87-104.
[24] Adam G. Kirk and O. Arikan, *Real-Time Ambient Occlusion for Dynamic Character Skins*, ACM SIGGRAPH Symposium, April 2007.
[25] "*Cinema: 100 Years of the Moving Image*" http://www.angelfire.com/vt/mhunt/cinema.html, 2000
[26] Granata, K.P., Brogan D.C., Sheth P.N., *Stable Forward Dynamic Simulation of Bipedal Gait Using Space Time Analysis,* In American Society of Biomechanics Proceedings of IV World Congress of Biomechanics, 2002.
[27] Kang Kang Yin, Kevin Loken, and Michiel van de Panne, *SIMBICON: Simple Biped Locomotion Control,* ACM Transactions on Graphics, 2007, Article No.: 105.
[28] M. McKenna and D. Zaltzer, *Dynamic Simulation of Autonomous Legged Locomotion,* Proceedings of the 17th annual conference on Computer graphics and interactive techniques, 1990, pp. 29-38.
[29] Ari Shapiro, Derek Chu, Petros Faloutsos, *The Dynamic Controller Toolkit*, Proceedings of the 2007 ACM SIGGRAPH symposium on Video games, pp. 15-20.
[30] L. Kovar, J. Schreiner M. Gleicher. *Footskate Cleanup for Motion Capture Editing*, Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA). July 2002, pp. 97-104.
[31] Samuel R. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*, Unpublished study, http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/index.html
[32] R. Boulic, B. Ulicny, D. Thalmann, *Versatile Walk Engine*, Journal of Game Development, 1(1), pp 29-52, Michael van Lent Editor, Charles River Media. www.jogd.com, 2004.
[33] Inman VT, Ralston HJ, Todd F, *Human Walking,* Baltimore, Williams & Wilkins, 1981.

[34] Samuel R. Buss, Jin-su Kim, *Selectively Damped Least Square for Inverse Kinematics,* In Journal of Graphics Tools, vol. 10, no. 3 (2005), pp. 37-49.
[35] Jean-Sébastien Monzani, Angela Caicedo and Daniel Thalmann, *Integrating Behavioural Animation Techniques*, EUROGRAPHICS 2001, Volume 20, Number 3.
[36] T. Conde, W. Tambellini, and D.Thalmann, *Behavioral Animation of Autonomous Virtual AgentsHelped by Reinforcement Learning, Lecture Notes in Computer Science*, vol. 272, pp. 175-180, Springer-Verlag: Berlin, 2003.
[37] *Acclaim ASF/AMC,* http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html
[38] *Euler Angles,* http://en.wikipedia.org/wiki/Euler_angles
[39] E .B. Dam, M. Koch, M. Lillholm, *Quaternions, Interpolation and Animation,* Technical Report DIKU-TR-98.
[40]*Maths - Conversion Quaternion to Euler*, http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/index.htm
[41] Gregory G.Slabaugh, *Computing Euler Angles from a Rotation Matrix*, Unpublished Report, 1999
[42] *Numerical Receipts in C,* Second Edition, Cambridge University Press 1992.
[43] David Gerhard, *Pitch Extraction and Fundamental Frequency: History and Current Techniques,* Technical Report TR-CS 2003-06, November 2003
[44] P. Mcleod, G. Wyvill, *A SmarterWay to Find Pitch,* in Proceedings of International Computer Music Conference, ICMC, 2005
[45] Anssi Klapuri, *Signal Processing Methods for the Automatic Transcription of Music,* PHD thesis submitted to Tampere University of Technology, 2004.
[46] L.Ikemoto, O.Arikan D.Forsyth, *Knowing when to Put Your Foot Down,* Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, .
[47] P.Glardon, R. Boulic, D. Thalmann, *Robust On-lin Adaptive Footplant Detection and Enforcement for Locomotion,* The Visual Computer: International Journal of Computer Graphics, Volume 22, 2005, pp. 194-209.
[48] Peter Bodik, *AutomaticFootplant Detection Inside FlMoView,* A summer project for Michael Gleicher, June -- August 2000.
[49] L. Kovar, M. Gleicher, F. Pighin, *Motion Graphs,* ACM Transactions on Graphics, 2002, Volume 21, Issue 3, pp. 473 – 482.

[50] J. Lee, J. Chai, P.S.A. Reitsma, *Interactive Control of Avatars Animated with Human Motion Data*, ACM Transactions on Graphics, 2002, pp. 491–500.
[51]Wan-Yen Lo, M. Zwicker, *Bidirectional Search for Interactive Motion Synthesis,*Eurographics, Computer Graphics Forum, 2010, pp. Volume 29, Issue 2, pp. 563-573.
[52] C. Fraley A. E. Raftery, *How many Clusters? Which Clustering Method? Answers via Model-Based Cluster Analysis?,* Computer Journal41, Volume 41, No: 8, 1998, pp. 578-588.
[53]A. Bruderlin, L. Williams, *Motion Signal Processing,* In Proceedings of ACM SIGGRAPH 95, Annual Conference Series, pages 97-104. ACM SIGGRAPH, August1995.
[54] T. Shederberg and E. Greenwood, *A Physical-based Approach to 2D Shape Blending,* Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques of SIGGRAPH, 1992, pp. 25-34.
[55] R. Demori, D. Probst, *Handbook of Pattern Recognition and Image Processing,* Academic Press, 1986, ch. Computer Recognition of Speech.
[56] P. Glardon, R. Boulic, D. Thalmann, *On-line Adapted Transition between Locomotion and Jump,* Proceedings of the Computer Graphics International, 2005, pp.44-50.
[57] P. Glardon, R. Boulic, D. Thalmann, *A Coherent Locomotion Engine Extrapolating beyond Experimental Data,* In: Proceedings of Computer Animation and Social Agent, pp. 73–83, Geneva (2004).
[58] M. Gleicher, *Motion Path Editing,* Proceedings of the 2001 Symposium on Interactive 3D Graphics, pp. 195-202.
[59] S. Andrew, M. Gleicher, *Motion Editing with Paths and Tiles,* BS Honors Thesis, 2003.
[60] A. Safonova, J.K. Hodgins, *Analyzing the Physical Correctness of interpolated Human Motion,* Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2005, pp. 171-180.
[61] J.Harrison, R.Rensink, and M.vandePanne. *Obscuring Length Changes during Animated Motion*. ACM Transactionson Graphics, 23(3): 569–573, 2004.
[62] Gregory M. Nielson*, v-Quaternion Splines for the Smooth Interpolation of Quaternions*, IEEE Transactions on Visualization and Computer Graphics, Vol. 10, No.2, 2004

[63] http://en.wikipedia.org/wiki/Sagittal_plane