DEGREE PROJECT, IN DD221X FOR MASTER DEGREE IN TMAIM , SECOND LEVEL

STOCKHOLM, *SWEDEN 2014*

Periodic Motion Extraction Using Harmonic Regression and Structural Parameterization

EXTRAKTION AV PERIODISKA RÖRELSER GENOM ANVÄNDNING AV HARMONISK REGRESSION OCH STRUKTURELL PARAMETRISERING

MAGNUS RAUNIO

KTH ROYAL INSTITUTE OF TECHNOLOGY

COMPUTER SCIENCE AND COMMUNICATION (CSC)

KTH Computer Science
and Communication

Periodic Motion Extraction Using Harmonic Regression and Structural Parameterization

Extraktion av periodiska rörelser genom användning av harmonisk regression och strukturell parametrisering

MAGNUS RAUNIO
mraunio@kth.se

Degree Project in Computer Science at CSC
Master's programme in Machine Learning
Supervisor/Employer: Florian T. Pokorny and Mikael Vejdemo-Johansson
Examiner: Danica Kragic

October 9, 2014

Periodic Motion Extraction Using Harmonic Regression and Structural Parameterization

Abstract

The work of this thesis focuses on how to construct a smooth and periodic motion from motion capture data of a motion with a periodic structure, such as walk cycles. To identify periodic structures in the motion capture data we rely on previously developed methods from algebraic topology to project each frame of the motion capture data onto a circle. This gives us a description of how the periodic motion looks structurally. To construct a typical periodic motion from this projection we propose a harmonic regression model on the positional joint rotations. To prevent overfitting of the regression model we use the joints velocities and accelerations as constraints, which helps in maintaining a natural flow for the motion. Our method helps in preventing overfitting of the regression model, but the extracted motion can still suffer from temporal artefacts due to difficulties in performing the inverse mapping from structure to time. The method described is implemented as part of a plugin for Blender to provide a tool for performing periodic motion extraction in a 3D animation tool-kit.

Extraktion av periodiska rörelser genom användning av harmonisk regression och strukturell parametrisering

Referat

Arbetet i detta examensarbete fokuserar på hur man kan konstruera en jämn och periodisk rörelse från motion capture data av en rörelse med periodisk struktur, som till exempel en gångcykel. För att identifiera periodiska strukturer i motion capture data använder vi tidigare utvecklade tekniker ifrån algebraisk topologi för att projicera varje bildruta av motion capture data på en cirkel. Detta ger oss en beskrivning av hur den periodiska rörelsen ser ut strukturellt. För att konstruera en typisk periodisk rörelse från denna projicering föreslår vi en harmonisk regressions modell på ledernas rotations positioner. För att förhindra overfitting av regressions modellen använder vi oss av ledernas hastigheter och accelerationer som restriktioner för att bibehålla det naturliga flödet hos rörelsen. Vår modell hjälper att förhindra overfitting av regressions modellen, men den extraherade rörelsen kan fortfarande lida av temporala artefakter på grund av svårigheter i att göra en inverterad mappning från struktur till tid. Metoden som beskrivits är implementerade i form av ett plugin till Blender för att ge tillgång till ett verktyg som kan extrahera periodiska rörelser i ett 3D animations verktyg.

Contents

Introduction

Using motion capture data to create realistic animations has become a standard technique in games and movies. There are also large motion capture databases available online for free [1], and even cheap RGBD cameras can be used to generate motion capture data [16]. It is becoming easier and cheaper to generate, or acquire, motion capture data that can be used by professionals or hobbyists.

However, when one wish to create periodic motions from motion capture data, like a walk cycle, the motion capture data stream will have to be aligned so that the end pose of the walk cycle fluidly transitions into the start pose. This has been commonly done by either manually segmenting and aligning the motion capture data [17], or by automatically estimating the phase and offset of the captured motion so that it can be segmented and aligned [12].

In this paper we instead rely on a method presented in [15] where each pose of the motion capture data is mapped onto a circle $\mathbb{S}^1$, thereby sorting the data so that similar poses follow each other. Techniques for creating a typical periodic motion period from this mapping and how to interpolate between different periodic motions are also presented in [15]. The method for extracting a typical motion would tend to yield jittery output motions though. In this thesis we investigate a different approach for extracting a typical motion from the data once it has been mapped onto a circle, one which should produce a smoother and more natural motion.

Background

2.1 Algebraic Topology

In this section we introduce some fundamental terms in algebraic topology to allow us to easier explain the algorithm for extracting periodic motions. It is provided more as a quick reference to look things up for those of us that are not that familiar with algebraic topology and a complete understanding of these topics are not essential to understand later parts of the thesis.

The most important concepts to take away from this section is that of homology, cohomology and cocycles. For a good reference on algebraic topology readers can reference [4], [7] or [2] which were used as references for the following section.

Simplex. In [4] a k-simplex is defined as "...the convex hull of $k+1$ affinely independent points". For examples of different k-simplices, see figure 2.1. A 0-simplex is a single vertex, a 1-simplex is a edge between two vertices, a 2-simplex is a triangle and so on.

Let u_i be a point in $\mathbb{R}^d$ and let $\sigma = [u_0, ...u_k]$ be a k-simplex. Then a face of the simplex σ is a subset of σ [4]. For example, the (k-1)-faces of the triangle in figure 2.1 would be the edges of the triangle. We notate that a simplex τ is a face of the simplex σ using the notation $\tau < \sigma$.

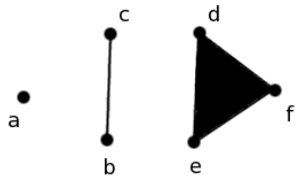


Figure 2.1: a is a 0-simplex, bc is a 1-simplex and def is a 2-simplex.

Simplicial complex. From [4] we have a definition of simplicial complexes which we paraphrase as, "...a finite collection of simplices Y such that $\sigma \in Y$ and $\tau < \sigma$ implies $\tau \in \sigma$, and $\sigma_i, \sigma_j \in X$ implies $\sigma_i \cap \sigma_j$ is either empty or a face of both".

In figure 2.2 we show a simplicial complex consisting of 0- , 1- and 2-simplices. If we removed the simplex bc from the set in figure 2.2 we would still have a valid simplicial complex, but if we instead removed simplex c we would no longer have a valid simplicial complex since we no longer would include all the faces of each simplex in the set

In figure 2.3 we show a set of simplices that are not a valid simplicial complex in the 2D plane due to the intersection between simplices bd and ce.

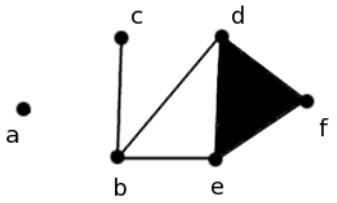


Figure 2.2: A simplicial complex consisting of simplices $\{a, b, c, d, e, f, bc, be, bd, de, ef, fd, def\}$.

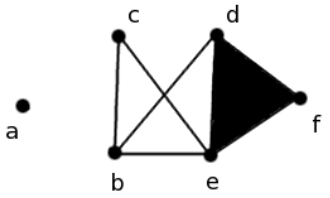


Figure 2.3: This is not a simplicial complex in the 2D plane due to the intersection between bd and ce.

Chains. A k-chain is a linear combination of k-simplices, i.e. $\sum_i a_i\sigma_i$ where σ is a k-simplex and a_i is coefficients [4] selected such that $a_i \in G$ where G is some group, ex. $\mathbb{Z}_2$. A k-chain α is notated as $\alpha \in C_k(Y; G)$, where Y is the simplicial complex that the chain occurs in. See figure 2.4 for an example of a 1-chain.

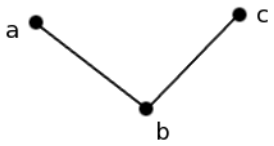


Figure 2.4: An example of a 1-chain, $\alpha = ab + bc$.

Boundary operator. The boundary operator ∂_k for a k-simplex σ is defined as a linear combination of σ:s (k-1)-faces. Let $\sigma = [u_0, ...u_k]$ where $u_i \in \mathbb{R}^d$, then the boundary operator can be defined [4] as:

$$\partial_k\sigma = \sum_{i=0}^{k} (-1)^i [u_0, ...\hat{u_i}, ...u_k] \tag{2.1}$$

Where $\hat{u}$ means that u is excluded. See figure 2.5 for a concrete example on how the boundary operator works.

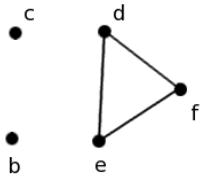


Figure 2.5: The boundaries $b - c$ and $de - ef + fd$ of the simplices in figure 2.1. The 0-simplex a has an empty boundary.

The boundary operator can be applied to chains as well as simplices and the boundary operator ∂_k applied on a k-chain α is defined [4] as:

$$\partial_k\alpha = \partial_k \sum_i a_i\sigma_i = \sum_i a_i\partial_k\sigma_i \tag{2.2}$$

Where a_i and σ_i are as defined in the previous description of chains.

Cycles and boundaries. A cycle is a chain where $\partial_k \alpha = 0$, i.e. its boundary is empty [4] and therefore it is a closed loop. See figure 2.6 for an example of a 1-cycle. A k-boundary is a k-cycle that is obtained from applying the boundary operator on a (k+1)-chain [4].

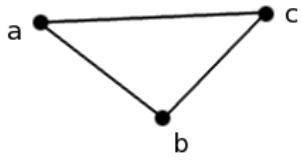


Figure 2.6: An example of a 1-cycle $\alpha = ab + bc + ca$, with boundary $\partial_k \alpha = \{a - b + b - c + c - a = 0\}$.

Homology. Homology is a way of mathematically talking about the structure of a simplicial complex by capturing holes in it. Each hole will be associated with a set of cycles that goes around the hole, these sets are known as homology classes.

The k-th homology classes are specified as the cosets of the quotient group Z_k/B_k, where Z_k indicates all k-cycles and B_k indicates all k-boundaries for a topological space [4], such as a simplical complex. This means that the k-cycles of a k-th homology class will differ only by k-boundaries. Thus by adding any boundary $b \in B_k$ to a k-cycle α from a homology class we will obtain another cycle $\alpha^{'}$ which is a member of the same homology class, i.e. $\alpha + b = \alpha^{'}$ where $\alpha, \alpha^{'} \in H$ and H is a homology class.

The k-th homology classes partitions the k-cycles into equivalence classes which is why we can use a single k-cycle to represent a homology class.

The reason that these cycles will capture holes is that the cycles in a homology class are not the boundaries of any (k+1)-chain, and if they are not a boundary, then there must be a hole there instead. We illustrate this with the following trivial example, the 1-cycle in 2.6 captures a hole, but if we "filled in" the triangle so that we added the simplex abc to the simplicial complex, the 1-cycle would no longer capture a hole because it is the boundary of the 2-chain abc.

Homomorphism. A homomorphism is a structure preserving map, which means that it is an operation that commutes with respect to the group operation [4]. For example, let U and V be groups with the group operation addition, and let $\varphi : U \to V$ be a homomorphism. For $a, b \in U$ we will for φ then have:

$$\varphi(a + b) = \varphi(a) + \varphi(b) \tag{2.3}$$

Next we introduce the dual homomorphism. Let U^* and V^* be the dual spaces of U and V, then the homomorphism $\varphi^* : V^* \to U^*$ is the dual of the homomorphism φ. As per the definition of a dual homomorphism we therefore have the following relation [4] between φ^* and φ:

$$(\varphi^* a^*)b = a^*(\varphi b) \tag{2.4}$$

Where $a^* \in V^*$ and $b \in U$.

Cochains. A k-cochain is a homomorphism φ that takes a k-chain to a coefficient in a group, i.e. $\varphi : C_k \to G$ where C_k is a group of k-chains [4] and G is ex. $\mathbb{Z}_2$. We notate a k-cochain φ as $\varphi \in C^k(Y; G)$ where Y is the a simplicial complex in which it occurs.

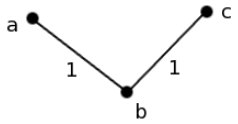


Figure 2.7: An example of a 1-cochain where the 1-chains ab and bc is mapped to 1 while $ab + bc$ is mapped to 0 when working in $\mathbb{Z}_2$.

Coboundary operator. The definition of the coboundary operator is as the dual homomorphism of the boundary operator ∂ [4]. For a more intuitive understanding of what the coboundary is, consider the simplex de in figure 2.1. Its coboundary will be a homomorphism of the simplex def in the same figure and if we had more 2-simplices sharing the face de they would also be part of the coboundary. The coboundary therefore operates in the opposite direction of the boundary operator and we notate the coboundary operator as δ^k where k is the dimension of the simplex we are working with. The coboundary operator δ^{k-1} is the dual homomorphism for ∂_k.

Cocycles and coboundaries. Like for chains and cycles, a k-cocycle is a k-cochain and a k-cocycle is defined as the kernel of the coboundary operator $ker(\delta^k)$ [4]. That is, for φ to be a k-cocycle then for each cycle $c \in C_{k+1}$ the following expression must be true $\delta^k\varphi(c) = 0$. In figure 2.8 we give an example of how to verify that a cochain is a cocycle.

Next, k-coboundaries are defined as the image of the coboundary operator $im(\delta^{k-1})$ [4], which means that k-coboundaries are those k-cochains we get when we apply the coboundary operator to a $(k-1)$-cochain.

Since the coboundary operator δ^{k-1} is defined as the dual homomorphism of ∂_k we will by definition of the dual homomorphism have $(\delta^{k-1}\varphi)\alpha = \varphi(\partial_k\alpha)$ where $\alpha \in C_k$ and $\varphi \in C^{k-1}$.

Cohomology. In [7] cohomology is referred to as "...an algebraic variant of homology, the result of a simple dualization in the definition". The dualization here refers to the dual space of chains, cochains, and the dualization of the boundary operator, the coboundary operator. Due to the relation between homology and cohomology, cohomology captures holes in much the same way as homology. In fact, when the cochains are chosen with field coefficients it is stated in [7] that "...cohomology is the exact dual of homology".

The cohomology classes are computed as elements in the quotient group Z^k/B^k where Z^k are all k-cocycles, and B^k are all k-coboundaries. Cohomology is not as easy to intuitively understand as homology, but it is more expressive by allowing us to assign values to simplices using cochains.

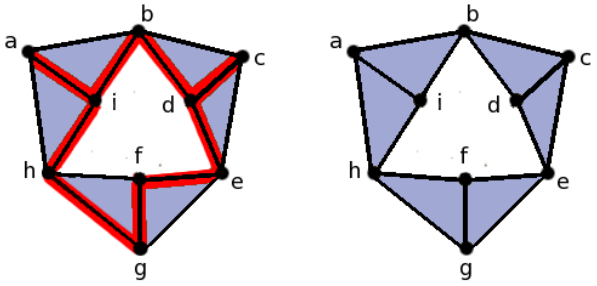


Figure 2.8: In the left figure we have a 1-cochain φ where all 1-simplices mapped to 1 are marked in red/underlined, the unmarked ones are mapped to 0. To the right is coboundary of this cocycle, where each 2-simplex is evaluated to 0 since by definition $\delta^1\varphi(abi) = \varphi\partial_1(abi) = \varphi(ab) - \varphi(bi) + \varphi(ia) = 0 - 1 + 1 = 0$ (try evaluating a different 2-chain, they all result in 0). Thus we have shown that φ is a cocycle since each 2-chain mapped by $\delta^1\varphi$ is 0.

Simplex generators. When computing the homology, or cohomology, classes for some set of points $Y \subset \mathbb{R}^d$, a simplicial complex will have to be constructed which describes Y. There exists a number of techniques for generating a simplicial complex from unordered point data, among them are, for example the Vietoris-Rips complex and the Witness complex [2]. A shared feature between these two complex generators is that they grow the simplicial complex by starting out with the point cloud Y as a set of 0-simplices and then grow the radius around the 0-simplices. Once the 0-simplices are within each others radial distance they are connected to create higher dimensional simplices, as shown in figure 2.9. The radius parameter is usually referred to as ϵ.

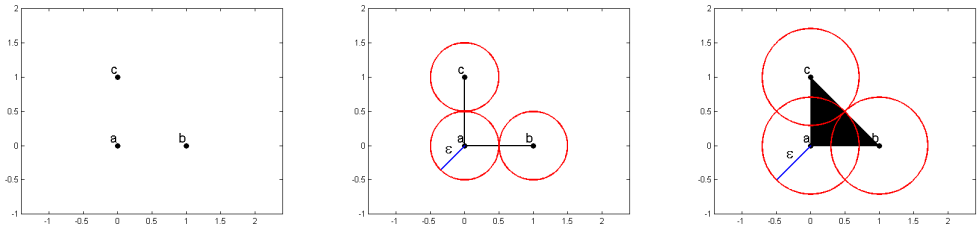


Figure 2.9: An example of the construction of a simplicial complex using Vietoris-Rips filtration. The leftmost image shows the point cloud for which we are constructing the complex. The middle image shows how the simplicial complex $\{a, b, c, ab, ac\}$ is constructed. The right most image shows how the simplicial complex $\{a, b, c, ab, ac, bc, abc\}$ is constructed.

Persistence. In [4] persistent homology is defined as a way of measuring the scale of topological features in a point cloud, where topological features refers to homology classes. The need to measure the scale of topological features comes from that several simplicial complexes can be constructed from a single point cloud, e.g. when we are using the Vietoris-Rips filtration, the simplical complex we constructs depends upon our choice of ϵ.

Persistence deals with this by tracking the births and deaths of homology classes as the simplicial complex is being constructed. This gives for each homology class a ϵ_i at which the class is born and a ϵ_j at which it dies.

Each class will be associated with a interval $[\epsilon_i, \epsilon_j)$ where $\epsilon_i < \epsilon_j$. This can be visualized using a barcode, where the x-axis corresponds to intervals of $[\epsilon_i, \epsilon_j)$, and the y axis corresponds to different classes as seen to the left in figure 2.10. An alternative representation is the one shown to the right in figure 2.10 where we represent each class with a point, and let the x-axis represent the birth ϵ_i, and the y-axis represents the death ϵ_j. The length of the interval $[\epsilon_i, \epsilon_j)$ can then be used to find which classes are the most persistent by selecting the classes with the greatest interval length.

Persistent cohomology can be used to measure the births and deaths of cohomology classes as well, as long as the cohomology is the exact dual of the homology [14].

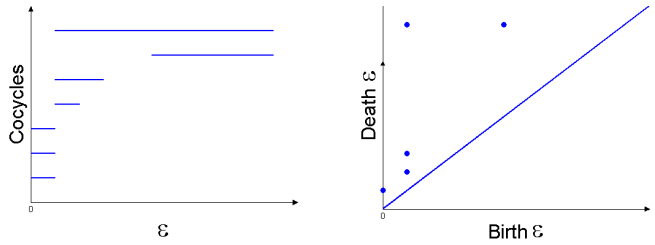
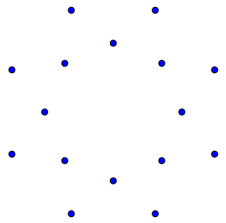


Figure 2.10: The figure shows two different ways to visualising persistence of cocycles. To the left a persistence barcode showing the life length of individual cohomology classes. To the right a persistence diagram shows the same information as in the left image, where each cocycle is represented by a point.

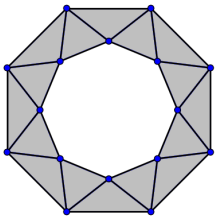
2.2 Previous Work

The work of this thesis is based on the work in [14] and [15]. The work in [14] presents an algorithm for how to produce a circular parameterization of the points in a point cloud. This circular parameterization technique relies on persistent cohomology, and it works by finding persistent cohomology classes which can be used to measure the distance around the hole which the class captures.

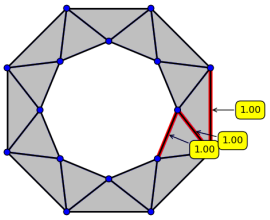
In [15] it is demonstrated how this can be applied for extracting periodic motions from motion capture data, and several issues are reported on difficulties with dealing with motion capture data this way. Among them how to reconstruct the periodic motion so that it still appears natural.



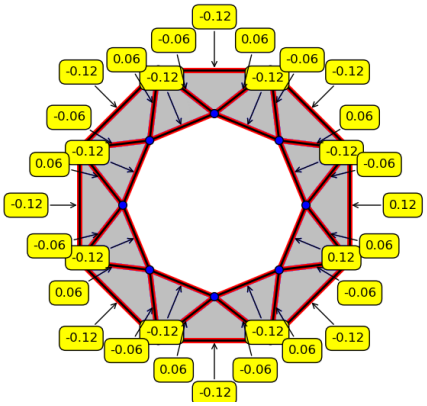
(a) A point cloud called Y...



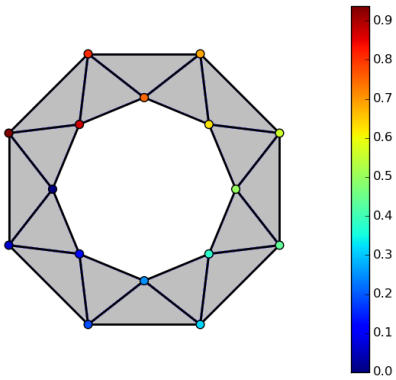
(b) ...that we construct a simplicial complex for...



(c) ...for which a 1-cocycle in $C^1(Y;\mathbb{Z}_p)$ is found. This cocycle is used to construct a 1-cocycle $\alpha \in C^1(Y;\mathbb{Z})$...



(d) ..from which we construct a smooth[2] 1-cocycle $\bar{\alpha} \in C^1(Y;\mathbb{R})$...



(e) ...that gives us a circular parameterization of the vertices on the interval $[0, 1)$.

Figure 2.11: The figure describes the major steps of the pipeline for the circular parameterization method discussed in [14].

[2]Note: The values in this figure are truncated, 0.12 is actually 0.125, and 0.06 is 0.0625.

We will now give a basic explanation of the method used in [14] to perform the circular parameterization, and in figure 2.11 we have outlined the important steps of the algorithm.

First off, for a point cloud Y a simplicial complex is generated, using e.g. the Vitoris-Rips complex, up to some maximum ϵ value. See figure 2.11(a) and figure 2.11(b) for an example of this process.

The next step consists of finding the most persistent cohomology class that occurs as the simplicial complex is generated and selecting a single 1-cocycle to represent this cohomology class. By finding the most persistent class we ensure that we find a significant hole structure in the simplicial complex. In figure 2.11(c) we show such a 1-cocycle which captures the hole in our simplicial complex.

When finding the 1-cocycle we use a cocycle with coefficients in $\mathbb{Z}_p$ where p is a prime number. This 1-cocycle can then be used to construct a 1-cocycle α with coefficients in $\mathbb{Z}$ (For details on this see section 2.4 in [14]).

The reason we construct a 1-cocyle $\alpha \in C^1(Y;\mathbb{Z})$ is that such a 1-cocycle can be used to construct a function that counts each time a lap is completed around the hole captured by the 1-cocyle [14]. A 1-cocycle with coefficients in $\mathbb{Z}_p$ does however not share this property which is why we try to switch from the coefficient group $\mathbb{Z}_p$ to $\mathbb{Z}$.

The algorithm for finding persistent 1-cocycles does however rely on that the cohomology is the exact dual to the homology which means that the algorithm will not work for cocycles with coefficients in $\mathbb{Z}$. Therefore $\mathbb{Z}_p$ coefficients are used to compute the cocycles, and we then try to convert the cocycles from $C^1(Y;\mathbb{Z}_p)$ to $C^1(Y;\mathbb{Z})$. The conversion may fail however, but it is uncommon, and switching the prime number p to a different one, and recomputing the cohomology classes, will generally resolve this.

To just use the 1-cocycle to count each time a lap is completed around the hole is not very useful though. By constructing an additional 1-cocycle $\bar{\alpha}$ with coefficients in $\mathbb{R}$, that lies in the same cohomology class as α, we get a cocycle that instead of counting laps count fractions of a lap, and since both cocycles are in the same cohomology class they capture the same hole.

The 1-cocycle $\bar{\alpha}$ will however need to be smooth in order to get a good parameterization, where the measurement of smoothness for a 1-cocycle $\bar{\alpha}$ is $\sum_i |\bar{\alpha}(\sigma_i)|^2$ where σ_i is a 1-simplex in the simplicial complex. By minimizing this term when constructing $\bar{\alpha}$ we obtain a smooth cocycle, and for our example simplicial complex the smooth 1-cocycle appears as in figure 2.11(d). The reason that minimizing $\sum_i |\bar{\alpha}(\sigma_i)|^2$ will yield us a good parameterization is that it is shown in [14] that this will also minimize the parameterization distance between the faces of σ_i. Since we are dealing with 1-cocycles it means that the faces of σ_i are 0-simplices and the distances, along the circular parameterization, between connected 0-simplices is minimized.

So far we have left out some details on the restrictions for $\bar{\alpha}$ and α, among them that $\bar{\alpha} = \alpha + \delta^0 f$ where f is a 0-cochain with coefficients in $\mathbb{R}$. Because of this relation $\delta^0 f$ actually captures the fractional part of $\bar{\alpha}$, which means that f

can be used to construct our circular function since the fractional part tells us how far around the hole we are (in [14] it is described how to quickly compute f when finding $\bar{\alpha}$). Then the circular function θ can be specified as $\theta = f \mod \mathbb{Z}$. In figure 2.11(e) we show how θ evaluates each 0-simplex to a value in $[0, 1)$, and it gives us a circular function, where we loop around from 1 to 0 once we complete a lap.

In [15] it is demonstrated how this can be used for extracting periodic motions from motion capture data. The motion capture data is represented by a skeleton, like the one in figure 2.12, as well as a time sequence that specifies the rotation of each joint for this skeleton. In order to convert this to a point cloud like in figure 2.11(a) each frame in the time sequence is mapped as a point into $\mathbb{R}^d$ where d is the number of degrees of freedom for the skeleton.

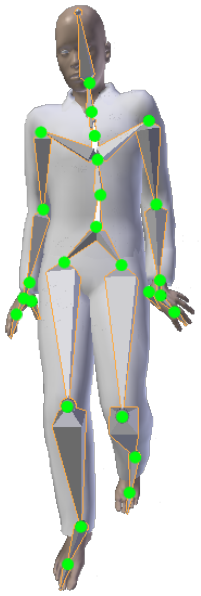


Figure 2.12: The skeleton associated with the motion capture data used in this thesis. The joints have been marked with circles, each with one to three degrees of freedom.

To then extract a periodic motion from this point cloud the circular parameterization approach described earlier is first executed. This gives a parameterization where similar skeleton poses are close to each other according to the parameterization. To this circular parameterization a Gaussian mixture model is fitted, yielding a probability distribution for the structural distribution of poses.

By then sampling a number of circular coordinates from this distribution a new pose is created for each of these sampled values. This is accomplished by performing a k-nearest neighbour lookup for the sampled values. Next, the k-nearest pose neighbours are used to calculate a centroid pose, and the centroid is used as a pose in the output motion. An output motion is then created by ordering the centroids according to their sampled circular coordinate.

While this gave a motion that was structurally similar to the input motion, the reconstructed motion would be quite noisy [15] due to creating the output poses from averaging structurally similar poses. This is what we wish to improve upon in this report, so that we can obtain smoother and more natural looking motions.

Method

3.1 Problem Description

Following the approach in [15] we construct a function $\theta : Y \to \mathbb{S}^1$, where $Y \subset \mathbb{R}^d$, and d denotes the number of degrees of freedom for the skeleton that our motion capture data is associated with. The problem we wish to solve is to construct a function $h : \mathbb{S}^1 \to \mathbb{R}^d$ so that we can map a circular coordinate back onto $\mathbb{R}^d$ so that in $\mathbb{R}^d$ we get a smooth closed curve representing the periodic motion.

The first thing to do is transform the motion capture data into a point cloud Y so that we can find a mapping $\theta : Y \to \mathbb{S}^1$. We do this by constructing each point $y \in Y$ from a frame in the motion capture data where the values along the dimensions of y is the rotational positions of the joints for that frame.

Ideally, this would give us a point cloud Y where the 1-cocycles will represent a complete period of our periodic motion. The problem with this representation though is that if a periodic motion arrives back at the same, or a similar, positional joint configuration before it has actually completed its period we will not find the "correct" 1-cocycle. This could for example occur in a walk cycle where we will pass through the same positional state space twice when the left leg moves before the right leg or vice versa. To deal with this delay embedding can be used, which we describe further in section 3.8.

The rest of this section deals with how we construct the function $h : \mathbb{S}^1 \to \mathbb{R}^d$ to reconstruct our motion from the circular parameterization.

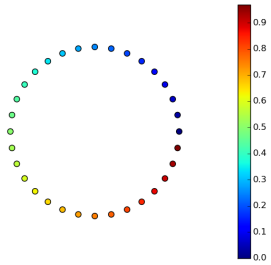
3.2 Harmonic Regression

In order to construct a function $h : \mathbb{S}^1 \to \mathbb{R}^d$ that gives a closed curve in $\mathbb{R}^d$ we use linear regression with a basis of sin and cos functions, known as harmonic regression. In figure 3.1 we show some examples where we have a function $h : \mathbb{S}^1 \to \mathbb{R}^2$ for a point cloud $Y \subset \mathbb{R}^2$ that we have found a circular parameterization $\theta : Y \to \mathbb{S}^1$ for.

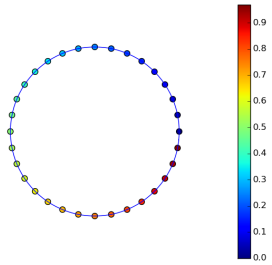
First, we will however discuss what assumptions our linear regression model makes. A linear regression system is typically specified as:

$$\bar{y} = X\beta + e \tag{3.1}$$

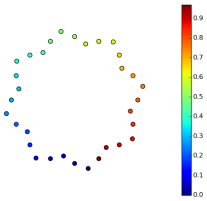
Where $\bar{y}$ is a observation vector whose values are assumed to depend on the observed values in the matrix X, and the dependencies are specified by the coef-



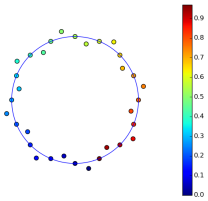
(a) A circular point cloud Y for which we have found a function $\theta : Y \to \mathbb{S}^1$.



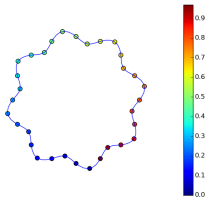
(b) The found function h for $K = 1$



(c) A circular point cloud with an added sin curve Y for which we have found a function $\theta : Y \to \mathbb{S}^1$.



(d) The found function h for $K = 1$



(e) The found function h for $K = 10$

Figure 3.1: The left hand side figures shows a point cloud Y, color coded according to the circular mapping found for Y. The mapping spans the interval $[0, 1)$, and it is colour coded according to this mapping. The right hand side figures shows a function h that we can construct for Y and its circular parameterization using different harmonic regression models.

ficient vector β. What is important here is that the error term e is assumed to be distributed as $\mathcal{N}(0, \sigma^2)$ when using ordinary least squares (OLS) to solve the system. The Gaussian assumption means that we assume the following properties for our error term e:

$$E[e|X] = 0 \tag{3.2}$$

$$\text{Var}[e^T e|X] = \sigma^2 I \tag{3.3}$$

To estimate β in equation (3.1), using OLS [9], one estimates β as:

$$\beta = (X^T X)^{-1} X^T \bar{y} \tag{3.4}$$

We will now introduce the notation we use for our harmonic regression model when we model a periodic motion. We let $x_i \in \mathbb{S}^1$ denote the value onto which y_i is

mapped, where $y_i \in Y$, and $|Y| = M$. Next we will construct a system of regression equations like in equation (3.1) where we for each dimension j of Y formulate the regression equations as:

$$\begin{cases} y_{1j} &= c_j + \sum_{k=1}^{K} a_{jk} \sin(2\pi k x_1) + b_{jk} \cos(2\pi k x_1) \\ y_{2j} &= c_j + \sum_{k=1}^{K} a_{jk} \sin(2\pi k x_2) + b_{jk} \cos(2\pi k x_2) \\ ... & \\ y_{ij} &= c_j + \sum_{k=1}^{K} a_{jk} \sin(2\pi k x_i) + b_{jk} \cos(2\pi k x_i) \\ ... & \\ y_{Mj} &= c_j + \sum_{k=1}^{K} a_{jk} \sin(2\pi k x_M) + b_{jk} \cos(2\pi k x_M) \end{cases} \tag{3.5}$$

By solving this system using the OLS method we can find the coefficients a_{j*}, b_{j*}, c_j and construct a function $h_j(x)$:

$$h_j(x) = c_j + \sum_{k=1}^{K} a_{jk} \sin(2\pi k x) + b_{jk} \cos(2\pi k x) \tag{3.6}$$

Which gives us a periodic model for each degree of freedom of the skeleton, and we use these to construct a model h for the periodic motion as:

$$h(x) = (h_1(x), ...h_d(x)) \tag{3.7}$$

The motivation for this model is that any linear combination of sin and cos functions will result in a periodic function, and by adding a bias term c_j we can have a periodic motion around a certain position. The larger we select K, the more accurately we can model the periodic motion, and when selecting K as infinite any periodic function can be reconstructed. Thus a large K is ideal for allowing us to recreate complex periodic motions, but as can be seen in figure 3.1(d) and 3.1(e) our choice of K can yield quite different functions, and a large K will make the model prone to overfitting.

We do however need a large K value for our model to be able to model complex joint movements. To prevent us from overfitting the model we therefore add additional constraints in the form of the velocity and acceleration of the motion. When the model only depended on the positions of the joints we would solve the following minimization problem using OLS:

$$\arg\min_{h_j} \sum_{i=1}^{M} |h_j(x_i) - y_{ij}|^2 \tag{3.8}$$

By including the velocities and accelerations we will instead solve the following minimization problem:

$$\arg\min_{h_j} \sum_{i=1}^{M} |h_j(x_i) - y_{ij}|^2 + |h_j^{'}(x_i) - y_{ij}^{'}|^2 + |h_j^{''}(x_i) - y_{ij}^{''}|^2 \tag{3.9}$$

Where the velocity for point y_{ij} is notated as $y_{ij}^{'}$ and the acceleration is notated as $y_{ij}^{''}$. The values of $y_{ij}^{'}$ and $y_{ij}^{''}$ are approximated by central finite differences, as described in section 3.7.

3.3 Weighted Harmonic Regression

In section 3.2 we proposed a regression model for reconstructing a periodic motion, but we do have a quite serious problem with this model. We mentioned in the same section that if we solve this system by OLS we assume that all the error terms are $\mathcal{N}(0, \sigma^2)$ distributed, but when we add the velocities and accelerations to this model this assumption is really poor.

A better assumption is that we have three different kind of error term distributions with different variances, one for position, one for velocity and one for acceleration. This kind of problem where the error terms have different variances is known as heteroskedasticity.

When using OLS regression the unbiased estimate of the error terms variance, σ^2, can be estimated [9] as in equation (3.10) where we use the notation from equation (3.1).

$$\sigma^2 = \frac{1}{n-k-1}||X\beta - \bar{y}||^2 \tag{3.10}$$

Where n is the length of $\bar{y}$ and $k+1$ is the width of X. We then model the positional error as $\mathcal{N}(0, \sigma_p^2)$, the velocity error as $\mathcal{N}(0, \sigma_v^2)$ and the acceleration error as $\mathcal{N}(0, \sigma_a^2)$, and estimate the error term variances σ_p, σ_v and σ_a as:

$$\hat{\sigma}_p^2 = \frac{1}{M-2K-1}\sum_i^M |h_j(x_i) - y_{ij}|^2 \tag{3.11}$$

$$\hat{\sigma}_v^2 = \frac{1}{M-2K}\sum_i^M |h_j^{'}(x_i) - y_{ij}^{'}|^2 \tag{3.12}$$

$$\hat{\sigma}_a^2 = \frac{1}{M-2K}\sum_i^M |h_j^{''}(x_i) - y_{ij}^{''}|^2 \tag{3.13}$$

The models for the velocity and acceleration do not have a bias term like the positional model since it is removed when we derivate h which is why we divide by $M-2K$ instead of $M-2K-1$. We will now modify our minimization problem and model it as:

$$\arg\min_{h_j} \sum_{i=1}^{M} |h_j(x_i) - y_{ij}|^2 + \alpha_j |h_j^{'}(x_i) - y_{ij}^{'}|^2 + \gamma_j |h_j^{''}(x_i) - y_{ij}^{''}|^2 \tag{3.14}$$

To remedy our heteroskedasticity problem we will then have to find weights α_j and γ_j for each degree of freedom j so that:

$$\sigma_p^2 = \alpha_j \sigma_v^2 = \gamma_j \sigma_a^2 \tag{3.15}$$

3.4 Weight Estimation

In order to estimate our weights α_j and γ_j we use feasible general least squares regression (FGLS) [8] instead of OLS. For OLS it is assumed the error e has the properties in equation (3.2) and (3.3). For GLS we do not have to assume (3.3) though, instead the error term variance may be a full covariance matrix as in equation (3.16).

$$\text{Var}[e^T e|X] = \Sigma \tag{3.16}$$

Where Σ is a positive definite matrix. The solution for β when using GLS [8] is:

$$\beta = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \bar{y} \tag{3.17}$$

Of course, the covariance matrix Σ is not known which makes GLS tricky to use, and that is the difference between FGLS and GLS. For FGLS the covariance matrix Σ will first have to be estimated. In section 3.3 we suggested that we would have three types of error variances, something known as group heteroskedasticity [8]. This means that we assume Σ has the form:

$$\Sigma = \begin{pmatrix} \sigma_p^2 I & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \sigma_v^2 I & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sigma_a^2 I \end{pmatrix} \tag{3.18}$$

Now we just have to estimate σ_p^2, σ_v^2 and σ_a^2 which will allow us to perform a GLS regression. Since our assumption is that we have group heteroskedasticity this can be done by performing separate OLS regressions [8] for the position, velocity and acceleration. These separate regressions models are then used to estimate our three variances as $\hat{\sigma}_p^2, \hat{\sigma}_v^2, \hat{\sigma}_a^2$ by using equations (3.11) through (3.13).

This gives us an estimate of our diagonal covariance matrix Σ. We could now solve our regression problem using equation (3.17), but we will instead reformulate it as a weighted least squares (WLS) regression, like we have in equation (3.14). This is possible since Σ is a diagonal matrix [8]. Our GLS regression with a diagonal covariance matrix will therefore yield the following minimization problem:

$$\begin{aligned} &\arg\min_{h_j} \sum_{i=1}^{M} \frac{|h_j(x_i) - y_{ij}|^2}{\hat{\sigma}_p^2} + \frac{|h_j^{'}(x_i) - y_{ij}^{'}|^2}{\hat{\sigma}_v^2} + \frac{|h_j^{''}(x_i) - y_{ij}^{''}|^2}{\hat{\sigma}_a^2} \\ =&\arg\min_{h_j} \sum_{i=1}^{M} |h_j(x_i) - y_{ij}|^2 + \frac{\hat{\sigma}_p^2}{\hat{\sigma}_v^2}|h_j^{'}(x_i) - y_{ij}^{'}|^2 + \frac{\hat{\sigma}_p^2}{\hat{\sigma}_a^2}|h_j^{''}(x_i) - y_{ij}^{''}|^2 \end{aligned} \tag{3.19}$$

Which means that our α_j and γ_j are estimated as $\frac{\hat{\sigma}_p^2}{\hat{\sigma}_v^2}$ respectively $\frac{\hat{\sigma}_p^2}{\hat{\sigma}_a^2}$.

The validity of this model depends on that our assumption of group heteroskedasticity is sound though. A much more complicated covariance matrix could be used instead, where the errors of velocities and accelerations are correlated with the positional errors, which seems like a sound assumption since we will approximate the velocities/accelerations with finite differences. We would however need to estimate the correlations well, and if we do this we would no longer be able to reformulate the problem as a weighted least squares problem. This in turn means that we must assure that Σ is positive definite, and use the full GLS regression model instead of the simpler WLS regression model (which we can solve using OLS if we factor in the weights into the absolute values). We would neither be able to estimate the variances/covariances by simply performing our three separate OLS regressions due to the assumption of covariances between the errors. This is why we opt for the model relying on the assumption of group heteroskedasticity.

3.5 Preprocessing

In section 2.2 we said that we could find a smooth circular parameterization $\theta : Y \to \mathbb{S}^1$, which meant that the distance between two 0-simplices in Y, along connecting 1-simplices, is proportional to the distance between the 0-simplices along the parameterization. That we have a "smooth" parameterization is not something we can use optimally though since it is smooth with respect to the distance between the point positions, i.e. skeleton poses that are similar will have a similar parameterization.

This means that our parameterization X will have a distribution over $[0, 1)$ which depends on the structure of Y. For example, a motion which moves at different speeds through different poses will have a distribution with peaks at where motion moves slowly. The structural parameterization therefore only tells us how the poses follow each other, but not at which speed they follow each other. We would like the distribution of X over $[0, 1)$ to not just correspond to structure, but time as well, and if this is not the case the velocities and accelerations can not be used as additional constraints since they are time dependent. This is one of the more serious weaknesses of using structural parameterization for periodic motion extraction since we can obtain a mapping that captures structural periods, but not exactly how these periods are related to time.

We attempt to construct a time dependency for X by relying on the fact that the input motion is uniformly distributed over time, where frames have been captured at a specific frame rate. Therefore each complete period of X should be uniformly distributed over the time interval $[0, 1)$. It is however not always clear where a period begins, and ends, and we may also have incomplete periods which makes it tricky to distribute each period uniformly. Doing so could also ruin the structural mapping we have obtained.

We therefore do not attempt to gain a time dependency by uniformly distributing each period, but instead by uniformly distributing X over the interval $[0, 1)$. In order to obtain a uniform distribution over $[0, 1)$ we sort the x_i values from smallest to largest, and for x_i at the j:th position in this ordering we assign the value:

$$x_i^{new} = \frac{j-1}{M} \tag{3.20}$$

This remaps the x values like in figure 3.2 to figure 3.3.

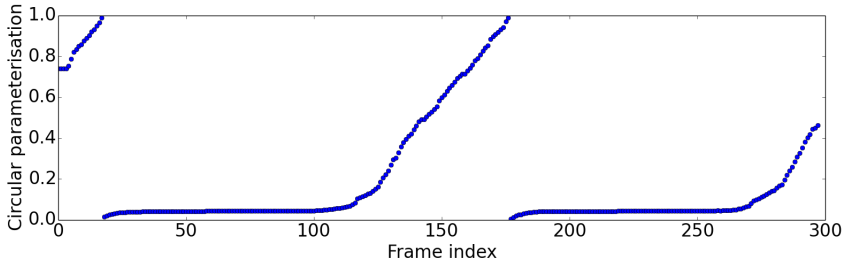


Figure 3.2: The figure shows how the circular parameterization in $[0, 1)$ relates to the frame indices of a single periodic motion with approximately two periods. The motion capture data is motion 08:07.

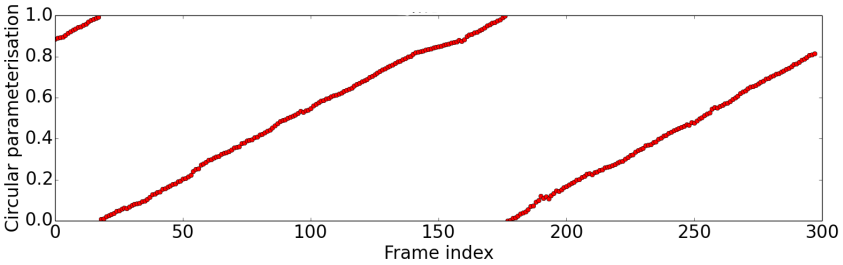


Figure 3.3: The same data as in figure 3.2, but where the points are uniformly distributed over $[0, 1)$.

There are some problems with this approach though which may lead to the following types of errors:

- We may end up pushing points with structural and temporal similarities away from each other, ex. if the input motion consists of the exact same period repeated a number of times after each other we will end up with a zig-zag motion like in figure 3.4 where we have what looks like saddle points where there should be none.
- If we have an overrepresentation of a certain pose it will occupy more time, ex. if have a point cloud consisting of one and a half motion, then the extracted motion will flow through one half of the motion faster than the other half creating a temporal artefact.

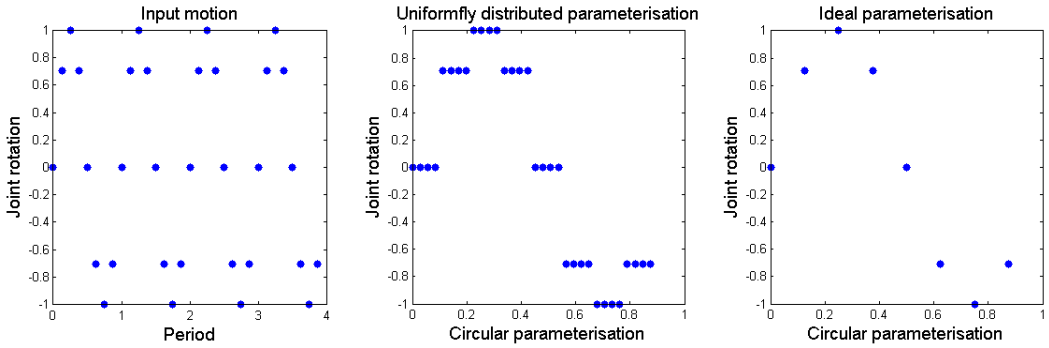


Figure 3.4: To the left we show an example motion for which we find a circular parameterization X. In the middle image we see that we end up find a zig-zag pattern for this motion after having uniformly distributed X. To the right we see the ideal distribution where the X values are overlapping where we have a temporal overlap.

In [15] a Gaussian mixture model approach, described in section 2.2, was used instead. The mixture model was used to sample circular coordinates and construct new poses, but we would like to remap our X values so that they correspond to time better. We could do something similar to this, by fitting a Gaussian mixture model $g(x)$ to X, and remapping each x_i as:

$$x_i^{new} = \frac{\int_0^{x_i} g(x)dx}{\int_0^1 g(x)dx} \tag{3.21}$$

By doing this we would obtain an inverse function from structure to time. The problem with this approach though is that we would not necessarily obtain a close to uniform distribution over $[0, 1)$ for our remapped X values, which we assumed to be the ground truth in the previously described approach. In figure 3.5 we show an example of how this remapping would look for figure 3.2 using the mixture model in figure 3.6.

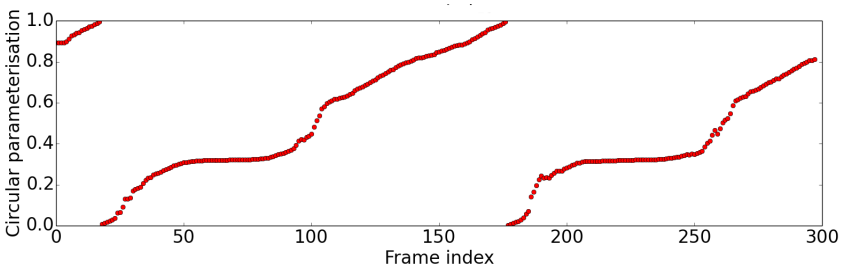


Figure 3.5: The figure shows the same data as in figure 3.2, but where the points are redistributed using the Gaussian mixture model approach.

The Gaussian mixture model may not always spread out the points well because the fitted model does not actually fit the data well. We may end up mapping points that are structurally similar, but not temporally similar to approximately the same

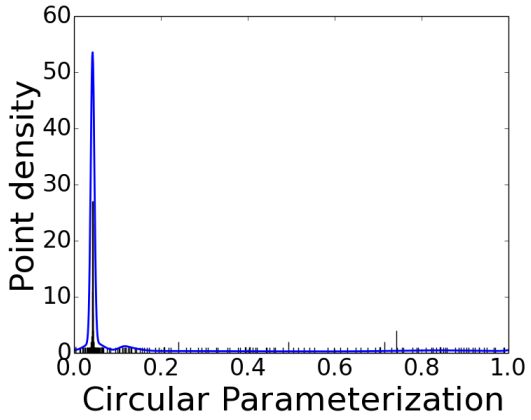


Figure 3.6: The figure shows the Gaussian mixture model for fitting the parameterization values from figure 3.2 together with a histogram for those values.

value. One could argue that this is simply caused by the wrong choice for the number of components for the mixture model or other parameters chosen incorrect when constructing our mixture model, but we have no way of knowing the "correct" parameter choice. The Gaussian mixture model also suffer from the second type of error we have mentioned, where we have an overrepresentation of certain poses making those poses occupy more time.

We choose the first approach discussed over the mixture model approach because our belief is that the errors introduced by uniformly distributing the samples can be better modelled as Gaussian noise than the errors we may get from using the mixture model and therefore be better suited for our regression model. The two approaches both have their weaknesses though, the Gaussian mixture model relies too much on the circular parameterization, and when uniformly distributing the x values we do not rely enough on the parameterization.

3.6 Motion Reconstruction

In order to reconstruct the input motion with the same frame rate as the input motion had we have to estimate the average number of frames for a periodic motion. In figure 3.3 we show the circular parameterization X of each frame for a motion. We wish to estimate the period length N, in frames, for this motion. To do this we use the same method as in [15], which remaps X to $\hat{X}$ with an added offset, like in figure 3.7. By then fitting a line to the points using OLS, like in figure 3.7, the line will have an incline that is the inverse of the frame length N.

The algorithm for performing the remapping from figure 3.3 to figure 3.7 is described in figure 3.8. It works by calculating an offset which tells us how many times, and in which direction, we crossed over the "edge" 0 to 1, i.e. it tracks which period we are currently in.

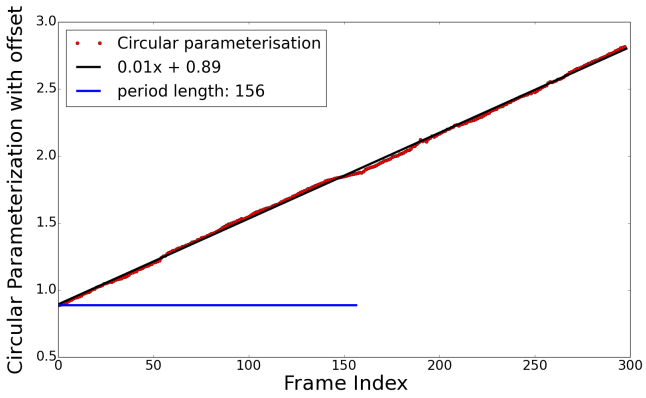


Figure 3.7: In this figure we see the remapped points from figure 3.3 as well as the fitted line and the estimated period length.

```
remap(x)
    x̂_1 = x_1

    for i ∈ {2,..,length(x)}
      offset = ⌊x̂_{i-1} − x_i + 0.5⌋
      x̂_i = x_i + offset

    return x̂
```

Figure 3.8: Pseudocode for performing the transformation from figure 3.3 to figure 3.7.

Now that we have an estimation of the frame length N, let Z be the reconstructed motion sequence and define z_i as:

$$z_i = h(\frac{i-1}{N}) \tag{3.22}$$

To then create one period of periodic motion we calculate all z_i for $1 \leq i \leq N$ where i is an integer. This assumes that we have unwarped the time axis, as described in section 3.5, so that the distribution of X on the interval $[0, 1)$ corresponds to time.

3.7 Derivative Approximation by Finite Differences

In this section we describe briefly how we approximate our velocities $y^{'}$ and accelerations $y^{''}$ by central finite differences [5]. For y_i we calculate $y_i^{'}$ and $y_i^{''}$ as:

$$y_i^{'} = \frac{\frac{y_{i-4}}{280} - \frac{4y_{i-3}}{105} + \frac{y_{i-2}}{5} - \frac{4y_{i-1}}{5} + \frac{4y_{i+1}}{5} - \frac{y_{i+2}}{5} + \frac{4y_{i+3}}{105} - \frac{y_{i+4}}{280}}{t} \tag{3.23}$$

$$y_i^{''} = \frac{-\frac{y_{i-4}}{560} + \frac{8y_{i-3}}{315} - \frac{y_{i-2}}{5} + \frac{8y_{i-1}}{5} - \frac{205y_i}{72} + \frac{8y_{i+1}}{5} - \frac{y_{i+2}}{5} + \frac{8y_{i+3}}{315} - \frac{y_{i+4}}{560}}{t^2} \tag{3.24}$$

Where t is the time step between the frames. This means that we require that we have an equidistantly spaced time grid for our y values, which we have since the motion we are attempting to model has been captured at a specific frame rate.

The problem is to select the correct value for t so that the magnitudes and velocities are scaled correctly when used over the interval $[0, 1)$. In section 3.6 we estimated the period length N of the input motion, and our time step t is in units of periods, which is why we select t as:

$$t = \frac{1}{N} \tag{3.25}$$

By approximating the derivatives this way we assume that N is a good estimate of the period length and also that the input motion has a consistent period length for each motion, ex. if the input motion consists of two periods we assume that the first period moves at the same speed as the second period. If this is not the case the derivatives are not estimated accurately since we would need different time steps for the two periods due to that we measure our time steps in units of completed periods. For example, the second period should have larger time steps than the first period if the second period is moving faster.

Assuming that the motions have approximately the same speed seems like a reasonable assumption though. Two motions that are structurally the same, but with different speeds may not be perceived as the same motion to a human after all. For motions with only slight differences in speed we will consider the additional errors in the velocities/accelerations as additional noise when performing our regression.

3.8 Delay Embedding

When creating our point cloud Y we may end up with self intersections in Y that occurs before we actually have completed the periodic motion due to pose symmetries, as described in section 3.1. Delay embedding [3] can be used to resolve this by encoding the trajectory of the motion, and not just the position. It does this

by constructing a point cloud $\hat{Y}$ from our position data Y, as in equation 3.26. We then use $\hat{Y}$ for constructing the circular parameterization X. Then we use $X_{1:M-m}$ as the circular parameterization of $Y_{1:M-m}$ and use this subset of Y to perform the harmonic regression.

$$\begin{aligned}\hat{y_i} =&(y_{i1}, y_{i2}, .., y_{iD}, \\ &y_{i+1,1}, y_{i+1,2}, .., y_{i+1,D}, \\ &... \\ &y_{i+m,1}, y_{i+m,2}, .., y_{i+m,D})\end{aligned} \tag{3.26}$$

In equation 3.26 the variable m is the embedding factor, which can be increased to encode further trajectory data into our point cloud. This helps us distinguish between when ex. a leg is moving forward or backward. How to choose the delay embedding factor we do not have an answer too, other than trial and error.

3.9 Translation Model

So far we have described how we deal with constructing a periodic motion based upon the joint rotations of the skeleton. However, for a periodic motion it could also be of interest to extract a translation component for the motion so that we have a motion where not just the skeletons joints move correctly, but also so that the skeletons translation speed is correct. For example, the skeleton should translate forward when the leg pushes the skeleton forward.

In order to accomplish this we construct a separate regression model for the translation component, which we call h^T. We construct h^T similar to how we constructed the joint model h, but with an added linear component as in equation (3.27) and equation (3.28).

$$h_j^T(x) = c_j + \sum_{k=1}^{K} [a_{jk} \sin(2\pi kx) + b_{jk} \cos(2\pi kx)] + l_j x \tag{3.27}$$

$$h^T(x) = (h_1^T(x), ...h_{d^T}^T(x)) \tag{3.28}$$

Where d^T is the number of degrees of freedom for the translation components. We choose this model so that the skeletons translation will have a periodic structure just as the joint rotations, but add the linear component since the translation may increase, or decrease, over time and not just rotate around a point like the joints. The X values we use will therefore also have to be modified since $h^T(x)$ is not a periodic function where the interval $[0, 1)$ will correspond to the interval $[n, n + 1)$ for any integer n.

For the translation model our X values will therefore have to be offset with the period they occur at. We therefore construct and use $\hat{X}$ instead of X, as described

in section 3.6. For $\hat{X}$ we no longer have a wrap around at the edge 0 to 1, but instead span over multiple periods.

Other than constructing a separate function h^T, and using $\hat{X}$ instead of X, the method for extracting the translation component is exactly the same as when we extract the joint rotations, as described in the previous sections of this chapter.

Result

In this chapter we evaluate our final weighted model. Ideally, we would like to evaluate how "natural" an extracted motion is, but such a measurement is very ill defined. We instead evaluate our extracted motions by computing the root mean squared error (RMSE) for our regression model as well as compute the error when we try to match the extracted motion sequence against the input motion sequence. We define the root mean squared error as in equation 4.1.

$$RMSE(h) = \sqrt{\frac{\sum_{i=1}^{M} ||h(x_i) - y_i||^2}{M}} \tag{4.1}$$

We also evaluate a weighted root mean squared error for the velocity and acceleration. We define the root mean squared error for the velocity and acceleration as:

$$RMSE(h^{'}) = \sqrt{\frac{\sum_{i=1}^{M} ||\alpha(h^{'}(x_i) - y_i^{'})||^2}{M}} \tag{4.2}$$

$$RMSE(h^{''}) = \sqrt{\frac{\sum_{i=1}^{M} ||\gamma(h^{''}(x_i) - y_i^{''})||^2}{M}} \tag{4.3}$$

Where $\alpha = (\sqrt{\alpha_1}, .. \sqrt{\alpha_d})$ and $\gamma = (\sqrt{\gamma_1}, .. \sqrt{\gamma_d})$. We weight the error for the velocity and acceleration to allow for comparison between the velocity and acceleration error with the positional root mean squared error. If all three root mean squared errors are of similar size then we should have a model that puts equal importance on all three measurements. This is not the same as comparing the standard deviations σ_p, σ_v and σ_a along each degree of freedom or perform a heteroskedasticity test, which is what we would actually like to do to see if we have remedied our heteroskedasticity problems, but it is problematic to present all this information in a compact way which actually tells us something about the motion since we are dealing with so many degrees of freedom. Since we are evaluating the

root mean squared error over all joints at once this gives us a more compact way of presenting similar information, since the standard deviation is evaluated over the residuals along one dimension while we evaluate the RMS error over the residuals along all dimensions.

We present the root mean squared errors for three different types of motions (boxing, walking and running motions) in tables B.1, B.2 and B.3. In these tables we show the root mean squared error for three different models. The weighted position-velocity-acceleration model (M_1), the positional model (M_2) and the unweighed position-velocity-acceleration model (M_3).

We can see in these tables that M_1 tends to yield similar RMS errors on the position as M_2, while M_3 tends to have much larger RMS errors for the position. This would indicate that our weighted model M_1 manages to reconstruct a similar motion to M_2, but where M_1 takes the velocities and acceleration into consideration.

For M_1 we also have similar errors for the position, velocity and acceleration. We will not be as brave to say that we have managed to resolve the inherent heteroskedasticity problems of our model M_1 since we do not perform any tests for heteroskedasticity, but instead say that we have obtained weights for the velocity and acceleration so that they are included in the regression in a sensible way since we put equal importance on the three different measurements now. For M_3 we can see that the errors of the accelerations are much larger than either the velocity and position which makes us overfit to the accelerations and get a motion that does not resemble the input motion as well as either M_1 or M_2.

In figure 4.1 we show a reconstructed motion for the different models. In this figure we see that M_1 gives the most sensible reconstructed motion, which seems to be the case most of the time based upon observations, while M_2 gives an overfitted function, and M_3 give a distorted motion due to overfitting to errors in the velocity and acceleration errors.

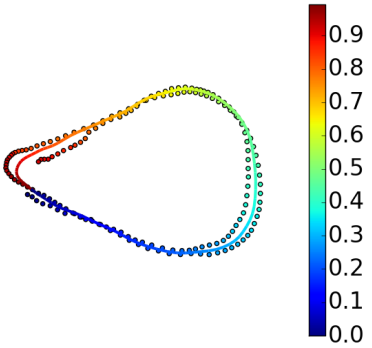
Now we would like to have a measurement of how well the reconstructed motion actually matches the input motion. We try to evaluate this by matching the extracted motion against the input motion as a form of template matching. For the input motion Y we match it against a repeated reconstructed motion Z according to equation (4.4).

$$\arg\min_{n} \sum_{i=0}^{M} ||y_i - z_{i+n}|| \tag{4.4}$$

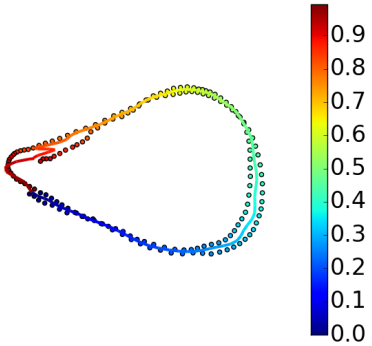
We then calculate the average point error between the matched sequences as in equation (4.5).

$$\sqrt{\frac{\sum_{i=0}^{M} ||y_i - z_{i+n}||^2}{M}} \tag{4.5}$$

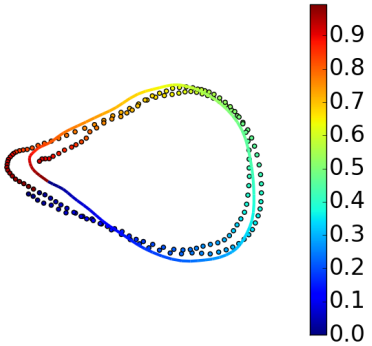
We present the resulting errors in tables A.1, A.2 and A.3. We can observe that M_1 and M_2 both yield approximately the same template matching error and that



(a) Motion extracted using the weighted model M_1. We obtain a smooth motion which resembles the input motion.



(b) Motion extracted using the position model M_2. We obtain a motion which resembles the input motion, but which can be perceived as jittery.



(c) Motion extracted using the unweighted model M_3. We obtain a motion that is smooth and, but that is warped compared to the input motion.

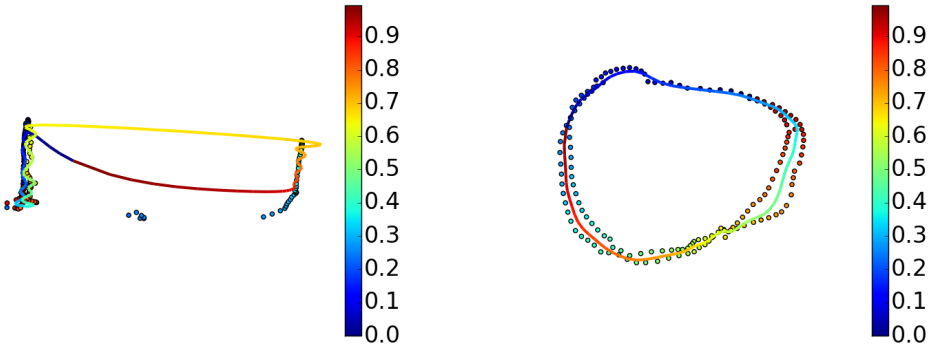
Figure 4.1: PCA projections of the reconstructed motion and the point cloud for motion 02:03 using the three different models.

they match the same indices generally, so M_1 produces a motion similar to M_2, but where M_1 has appropriate derivatives as well.

Since we evaluate the template matching error in much the same way as the positional root mean squared error these should be somewhat comparable. The RMS error will of course be smaller since we have minimized this error when constructing our motion, but if we obtain a motion where the RMS error is small and the template error is also small then we have a motion that we could model accurately and which also appears the same as the input motion.

An example of a motion which does not turn our very well for model M_1 is motion 35:19 which has a template matching error much greater than its positional root mean squared error. The reason this motion does not turn out well is because

of single bone which throws off the circular parameterization. As a comparison, in figure 4.2(a) we show the PCA projection of motion 35:19 using all bones of the skeleton, and in figure 4.2(b) we show the same motion, but where the troublesome bone has been excluded.



(a) PCA projection of motion 35:19.

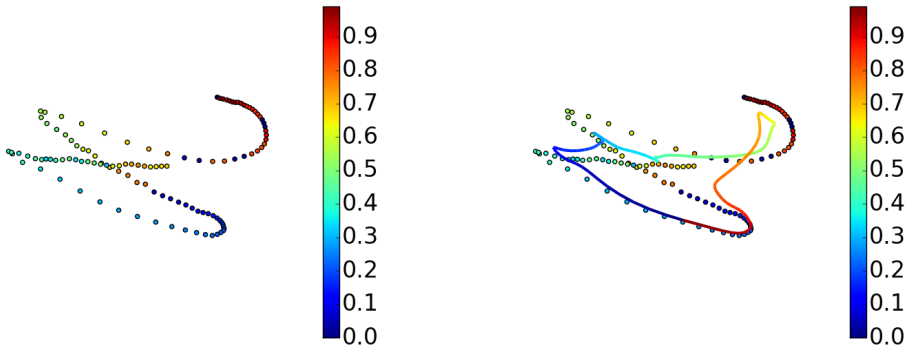
(b) PCA projection of motion 35:19 where the 'rhand' bone has been eliminated.

Figure 4.2: The PCA projection of motion 35:19 is meant to show how a single bone may have a large impact on the circular parameterization. In the left hand side figure we end up performing one and a half lap around the hole shown in the right hand side image, thereby gaining a circular parameterization which is not what we are actually looking for.

The troublesome bone turns out to be a minor bone in the right hand, which suddenly makes a large shift in its rotation along its x-axis, probably caused by errors in the motion capture system. The bone is not really important for the periodic motion though, which is a running motion, but because of the large shift in magnitude the distances between the 0-simplices are largely dependent on the right hand bone, therefore throwing the circular parameterization off target.

This is similar to the trouble we have with the boxing motions, except that instead of a single bone throwing off the parameterization we have a culminative sum of slight differences which gives us a poor parameterization. For the running and walking motions the periods are very similar each time, but for the boxing motions we have a more significant differences between the different periods due to that a lot of joints have minor differences. For example, when the second punch is thrown the pose might have the knees bent more than when the first punch is thrown, or perhaps just the feet are shifted more outwards. Together these differences causes trouble for the circular parameterization, where figure 4.3(a) and figure 4.3(b) illustrates this problem.

Because of these kinds of problems, it could be more useful to use a lower dimensional skeleton when performing the circular parameterization, ex. for a running motion the legs would be enough for finding the periodic structure of the motion or



(a) PCA projection of the circular parameterization for 14:02.

(b) PCA projection of the reconstructed motion using model M_1 for input motion 14:02.

Figure 4.3: PCA projections for motion 14:02, showing how the boxing motions are slightly offset each period making it more difficult to find a good parameterization.

the upper body for a boxing motion. The problem would be to automatically know which bones are actually significant for the motion, using just ex. a PCA projection would not always be helpful as demonstrated by figure 4.3(a).

An alternative approach could be, instead of reducing the dimension, to rescale the dimension values so that bones which are not considered as important for the circular structure have smaller magnitude than the important bones. A reasonable assumption would be that the larger bones are more important than the smaller bones, so by first normalizing the values along each dimension and then scaling each dimension by the length of the bone it represents one could accomplish this. This would help to ensure that we do not let small bones, such as the bones in the hand, be dominant when computing the distances between the 0-simplices, but instead let large bones dominate.

Other than the problem of finding a good circular parameterization we observe the type of errors we have discussed in section 3.5, where the motion flows through certain poses too quickly. We observe this behaviour more among the motion capture data of running motions in comparison to the walking motions, due to that the running motions tend to have fewer periods than the walking motions. The more periods we have, the less noticeable the error will be since the incomplete periods gain less importance. Using the velocities and accelerations as constraints seems to lessen this problem somewhat, but not enough for it to not be noticeable.

Software

In this chapter we give a short description of the software and resources used for this thesis, as well as the software developed which is available in the form of a Blender plugin written in Python.

5.1 Tools

Motion capture data. The motion capture data used in this paper is from the CMU database [1] and is provided in the form of *.asf* and *.amc* file pairs which describes the motion capture data. The *.asf* file describes the skeleton of the captured subject in the form of a hierarchical structure of joints where the root of the hierarchy describes the global rotation and translation for the skeleton. The *.amc* file contains the actual motion capture data and describes a time sequence with joint rotations and global rotation and translation for the root joint. The data is converted from the *.asf*/*.amc* format to *.bvh* format using [10] in order to allow for easy importing into Blender.

Blender. Blender [6] is a opensoure 3D animation suit that we are using for visualizing the motion capture data and the results of periodic motion extraction. Blender provides easy access for integrating user created plugins written in Python, and the method described in this thesis is implemented as a plugin for Blender. The Blender version used is 2.70 running Python 3.3.

Dionysus. Dionysus [11] is a C++ library that provides access to different algebraical topology computations and implements among other things the method described in [14] for mapping a point cloud to a circle. Dionysus also provides Python bindings (using Boost.Python) for the high performance C++ implementation which allows for easy integration with Blender. Since the latest Blender versions uses Python 3, the Dionysus bindings will need to be built with Python 3.

Python libraries. The developed Blender plugin depends on the following python libraries: NumPy, CVXOPT, Matplotlib and scikit-learn.

5.2 Plugin Design

In figure 5.1 we show the design of the Blender plugin. It consists of a front end that handles the interaction with Blender and a back end that performs the motion extraction.

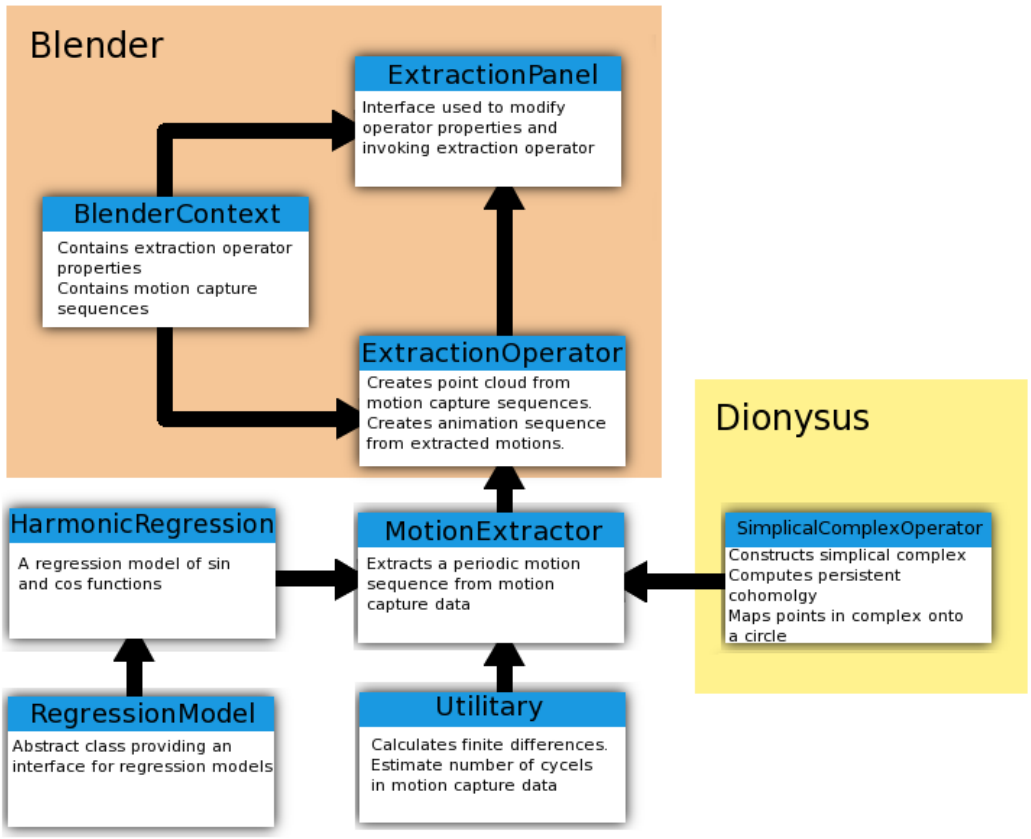


Figure 5.1: The figure shows the components of the software and which ones are connected with Blender and Dionysus.

The Blender part of the software is design according to the MVC (model-view-controller) pattern, where BlenderContext corresponds to the model, ExtractionPanel to the view and ExtractionOperator to the controller. BlenderContext is the model Blender uses internally to track the state of the current session. To this we have attached our own model which tracks the current input options for our ExtractionOperator, and these input options are editable via ExtractionPanel.

ExtractionPanel is the UI part of the plugin, through which input parameters are edited and the ExtractionOperator is invoked. The ExtractionOperator then converts the selected input motions into a point cloud which is sent to MotionExtractor and receives back from MotionExtractor a number of points which it converts back into animations that are stored in BlenderContext.

MotionExtractor depends on Dionysus to perform the circular mapping of the point cloud which it then uses to reconstruct a periodic motion using the HarmonicRegression class. The HarmonicRegression class is a subclass of RegressionModel which provides an interface for regression models.

5.3 Interface

The Blender interface is highly user customizable, consisting of several different *Editors* which are sub windows in the Blender interface. Each *Editor* enables different functions and views. Our plugin operates in the *3D View Editor*, which also allows for switching between different *Modes*. Each *Mode* is meant for working with different aspects of 3D animation and modelling.

Our plugin for extracting periodic motions is available in either *Object Mode* or *Pose Mode* when working in the *3D View Editor*. *Object Mode* is used for working with positioning of 3D objects, such as a collection of bones, known as an armature. *Pose Mode* is on the other hand meant for working with transforming individual bones of an armature. We bring this up since the plugin will behave differently depending on which *Mode* is active.

In *Object Mode* one or several armatures can be selected and the periodic motion extraction operator can be invoked. This will for each selected armature, with an attached animation, extract a periodic motion. In *Pose Mode* individual bones can be selected instead, and periodic motion can be extracted which only uses those specific bones. This can be useful in case one only wish to extract a periodic motion for part of the body, for example the upper body of a human armature.

In figure 5.2 and figure 5.3 we show how the plugin is integrated with the Blender interface, where figure 5.2 is in *Object Mode* while figure 5.3 is in *Pose Mode*. These figures also gives labellings for different parts of the interface, both for the input options for the plugin and other relevant parts of the Blender interface, and what these labellings refers to we explain in the following list.

1. Playback of animations attached to the armatures shown in *3D View Editor*. This option is available in *Timeline Editor*.

2. Lists available animation and displays the currently active animation for the selected armature. Once a periodic motion is extracted it will be available in this list together with all other animations. The extracted motion will be prefixed with the word “Periodic” to distinguish it from the input motion. This option is available in *Dope Sheet Editor*.

3. The currently active armature when in *Object Mode*. It is this skeletons animation which is currently displayed at label two and the skeleton which we will extract a periodic motion for if we invoke the extraction operator. Part of the *3D View Editor*.

4. Invokes the periodic motion extractor operator using the input options listed below.
5. Selects maximum ϵ value to use when constructing the simplicial complex. Lower values decrease computation time, but higher values allows for finding more persistent cocycles that could be better candidates when extracting the periodic motion.
6. Allows for subsampling the input motion. A value of 1 means we sample every frame, a value of 2 every second frame, etc. By subsampling we can decrease the computation time.
7. Allows for limiting the frame range used when performing the motion extraction, in case we are only interested in part of the input motion, or if we wish to decrease computation time. Makes option **8**-**9** available.

Figure 5.2: The figure shows the plugin interface and what it looks like when we mark an armature that which we wish to extract a periodic motion for when *Object Mode* is active.

8. The first frame to use when extracting periodic motions.
9. The maximum number of frames to use from the input motion.
10. The maximum number of cocycles to select for constructing periodic motions. For each cocycle we will perform a separate regression based upon the circular parameterization the cocycle yields, and for each cocycle a periodic motion is output into the animation list. The n cocycles selected are the n most persistent cocycles found.
11. Selects if we wish to compute the translation component for the input motion, in addition to the joint rotations, as described in section 3.9. Makes option **12**-**14** available.
12. Check box for enabling computation of the translation along the x-axis, while the drop down menu allows for switching between regression model equation (3.27) and (3.6).
13. Same as option **12** but along the y-axis.

14. Same as option **12** but along the z-axis.
15. Advanced options check box. Enables advanced input options **16-24**.
16. Selects the delay embedding factor to be used, as described in section 3.8.
17. Selects the prime number p to use when performing the circular parameterization. Only important if we fail to lift the cocycle from Z_p to Z.
18. The K value to use for our joint regression model, as described in section 3.2.
19. The K value to use for our translation regression model.
20. Uncheck in case we do not wish to use the velocity as a constraint when extracting a periodic motion.
21. Uncheck in case we do not wish to use the acceleration a constraint when extracting a periodic motion.

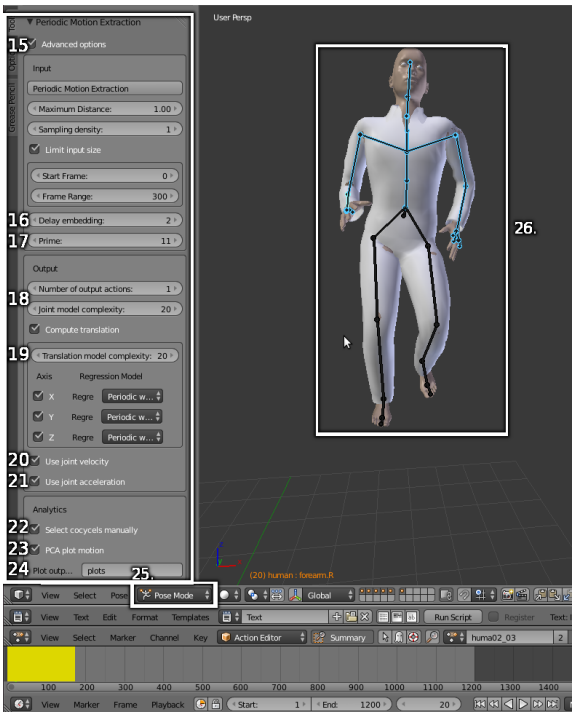


Figure 5.3: The figure shows the plugin interface when the advanced options have been enabled and how the view may look when *Pose Mode* is active. By selecting the upper body a periodic motion for just the upper body of the skeleton can be extracted.

22. If checked, a window will open once the simplicial complex has been constructed and the persistent cocycles found. The window will display a persistence diagram representing the cocycles. In this diagram cocycles can be selected manually and will override the automatic selection of the most persistent cocycles.
23. Check if one wish to output 2D plots of a PCA projected representation of the input motion, as well as the output motion.
24. Specifies the output directory for the 2D PCA plots.
25. Switches between *Object Mode* and *Pose Mode*.
26. In *Pose Mode* one can select individual bones to use when extracting the periodic motion. Only the selected bones will be animated. The selected bones that are used when extracting a periodic motion are highlighted in blue.

Discussion

In this report we have attempted to use the first and second order derivatives to prevent overfitting of our harmonic regression model. We will now discuss some alternative methods to our approach for modelling a periodic motion.

For our harmonic regression model we use the velocities and accelerations to prevent overfitting of the regression model, but there are also other ways of doing this, such as ridge regression [13]. It works by assuming that the coefficients β, excluding the bias term, also follows a normal distribution around 0. By including this assumption into the minimization problem large weights would be penalized, which would prevent the model from growing too complex. Instead of minimizing equation (3.8) one would then minimize:

$$\arg\min_{\beta} ||X\beta - \bar{y}||^2 + \lambda||\beta_{-1}||^2 \tag{6.1}$$

Where β_{-1} are the coefficients with the bias term excluded, and λ is a manually selected weight. To select λ cross validation would have to be used and iteratively find where λ causes the model to do overfitting instead of underfitting. To use the velocities and accelerations as constraints on the weights suits our problem better since we only have to solve a constant number of regression problems in order to find suitable weights and construct a model that is not overfitted. Since we are dealing with d number of degrees of freedom it is preferable that each model can be found fairly quickly in order to allow for better user interaction.

For our current model we have also assumed that we have group heteroskedasticity, but we have not dealt with any heteroskedasticity problems that may occur internally within ex. the velocities. This does not seem to be a serious problem for our extracted motions, but a possible way to deal with this could be to first follow our approach to obtain weights to remedy the group heteroskedasticity and use that as a basis to further improve the model. Using iterative reweighed least squares (IRLS) [13] we could try to find individual weights for each points position/velocity/acceleration to attempt to deal with any possible remaining heteroskedasticity problems. Our assumption of group heteroskedasticity seems to be valid though, so we do not include IRLS in our solution since it would further add to the computation without necessarily much gain.

A more interesting aspect to take into account is that for our current model we have assumed that our y values have a periodic structure around some mean

value. This may not always be the case for joints that can rotate 360°, since they may just rotate with an increasing value and still complete multiple periods. For a human skeleton there is only the shoulder and the root joint, which tells us the overall rotation of the skeleton, that can do this though, so it is not such a severe problem. But it does prevent us from modelling full body rotations, such as a human performing a pirouette. To resolve this one could use use polar coordinates to represent the rotation of those joints. For joint j in the point cloud Y we would compute polar coordinates (y^u_{*j}, y^v_{*j}), where we use $*$ to indicate all values, as:

$$y^u_{*j} = \cos(y_{*j}) \tag{6.2}$$

$$y^v_{*j} = \sin(y_{*j}) \tag{6.3}$$

We would then specify the the joint model as $h_j(x) = atan2(h^v_j(x), h^u_j(x))$, where h^u_j and h^v_j are periodic functions fitted to y^u_{*j} respectively y^v_{*j}. The problem is to find suitable functions for the polar coordinates. We could use separate harmonic regression models for them, but that would not be entirely sound. Between h^u_j and h^v_j we have the relation $h^u_j(x)^2 + h^v_j(x)^2 = 1$ due to that we are modelling polar coordinates. We would therefore, instead of minimizing (3.8), wish to minimize the constrained model:

$$\arg\min_{h^u_j, h^v_j} \{\sum_{i=1}^{M} |h^u_j(x_i) - y^u_{ij}|^2 + |h^v_j(x_i) - y^v_{ij}|^2 \mid h^u_j(x)^2 + h^v_j(x)^2 = 1\} \tag{6.4}$$

To this we could add the velocity and acceleration as well, as we do in equation (3.9). If one wish to use polar coordinates the joints will of course have to be converted into polar coordinates before constructing the circular parameterization, since the circular parameterization also relies on that the y values have a periodic structure.

Conclusions

The method we have presented for generating a periodic motion from a structurally parameterized point cloud shows some merit. We can generate a motion that represents the input motion, and by using the first and second order derivatives, as additional constraints for our regression model, we can also obtain a model that is not prone to overfitting and therefore generates a smoother and more natural motion. Our velocity and acceleration approximations depend upon that we estimate the length of a period correctly though or we will have an incorrect time step when approximating the derivatives by finite differences.

Of the three types of motions we tried our method on we find that for running and walking motions the method works well, but for the boxing motions we try the method on we can not obtain any good structural parameterizations. The boxing motions tend to not be as structurally similar as the running/walking motions each period in the high dimensional space, due to a few bones which do not exhibit periodic behaviour and those bones can come to dominate the structural parameterisation. For this reason we suggest that only a few major bones are used to construct the structural parameterization.

One problem with our approach is that there may exist temporal artefacts. The motion may flow too fast through certain poses due to the fact that we can not recover a perfect inverse function from structural to temporal space. This means that poses that are more common, due to motion data with incomplete periods, will be overrepresented when reconstructing a periodic motion. The easiest way to resolve this would be to prune the data to only contain complete periods or use motion capture data with many periods so the temporal artefacts become minimal.

The elegant way to resolve this would be to construct a better inverse function from structure to time, which takes both the structural parameterization and time into consideration. The two methods we have proposed in section 3.5 rely instead on the structural parameterization and pose density. How to construct such a function is not clear though.

Another interesting problem to still solve is how to deal with joints that can perform 360° rotations. We suggested in section 6 to use polar coordinates and to solve the constrained minimization problem (6.4) to deal with this, but we can not suggest a method for solving this minimization problem.

The final product of our work is implemented as a Python package to be used as a plugin for Blender, and provides access to a tool for extracting periodic motions.

Bibliography

[1] CMU Graphics Lab Motion Capture Database. http://mocap.cs.cmu.edu/.

[2] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[3] Frederic Chazal, Daniel Chen, Leonidas Guibas, Xiaoye Jiang, and Christian Sommer. Data-driven trajectory smoothing. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2011.

[4] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[5] Bengt Fornberg. Generation of finite difference formulas on arbitrarly spaced grids. *Mathematics of Computation*, 1988.

[6] Blender Foundation. Blender. http://www.blender.org/, 2014.

[7] Allen Hatcher. *Algebraic Topology*. Allen Hatcher : paper or electronic for noncommercial use may be freely withhout explicit permission from the author, 2000.

[8] Chung-Ming Kuan. *Statistics: Concepts and Methods*. http://homepage.ntu.edu.tw/~ckuan/pdf/et01/et_Ch4.pdf, second edition, 2004.

[9] Harald Lang. Topics on applied mathematical statistics, 2012.

[10] Fengjun Lv. Amc2Bvh. http://vipbase.net/amc2bvh/, 2006.

[11] Dmitriy Morozov. Dionysus. http://www.mrzv.org/software/dionysus/, 2012.

[12] Dirk Ormoneit, Michael J. Black, Trevor Hastie, and Hedvig Kjellström. Representing cyclic human motion using functional analysis. *Image and Vision Computing*, 23:12641276, 2005.

[13] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. 2012.

[14] Vin Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete Computational Geometry*, 45(4):737–759, 2011.

[15] Mikael Vejdemo-Johansson, Florian T. Pokorny, Primoz Skraba, and Danica Kragic. Cohomology learning of periodic motion. *Submitted to AAECC*, 2013.

[16] Xiaolin Wei, Peizhao Zhang, and Jinxiang Chai. Accurate realtime full-body motion capture using a single depth camera. Technical report, Texas AM University, 2012.

[17] Yaser Yacoob and Michael J. Black. Parameterized modeling and recognition of activities. Technical report, Computer Vision Laboratory, University of Maryland, 2002.

Template Matching Errors

- M_1 is the weighted model suggested in (3.14).
- M_2 is the positional model suggested in (3.8).
- M_3 is the unweighted model suggested in (3.9).

Table A.1: Boxing cycle errors. The table shows the average errors in radians for each frame when matching the extracted motion against the input motion for three different models M_1,M_2,M_3, and the matched index is also listed. All the motions listed in this table are of motion capture data of a human who is boxing. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 5.

Motion ID	Boxing average point error (radians)			Period count
	M_1	M_2	M_3	
	Error, Index	Error, Index	Error, Index	
13:17	1.75 176	1.90 175	8.30 226	1.00
14:02	0.95 41	1.02 42	3.42 43	1.00
15:13	0.51 173	0.53 174	2.86 172	1.52
17:10	0.62 23	0.74 23	5.58 25	1.00
79:08	0.64 53	0.71 54	6.39 54	1.00
Mean	0.89 -	0.98 -	5.31 -	
Median	0.64 -	0.74 -	5.58 -	

Table A.2: Walk cycle errors. The table shows the average errors in radians for each frame when matching the extracted motion against the input motion for three different models M_1,M_2,M_3, and the matched index is also listed. All the motions listed in this table are of motion capture data of a walking human. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 2.

Motion ID	Walking average point error (radians)			Period count
	M_1	M_2	M_3	
	Error, Index	Error, Index	Error, Index	
02:01	1.15 197	1.13 226	8.92 287	1.00
02:02	0.38 90	0.39 90	0.78 90	2.49
05:01	0.99 255	0.99 252	4.01 249	2.27
06:01	0.34 42	0.35 42	9.96 76	3.33
07:01	0.37 126	0.37 126	0.95 125	2.40
07:02	0.41 15	0.41 16	0.75 16	2.49
07:03	0.30 66	0.30 66	0.70 65	2.83
07:04	0.37 87	0.38 87	1.35 87	2.58
07:05	0.36 130	0.37 130	0.83 130	2.73
07:06	0.28 87	0.28 87	1.18 87	3.00
07:07	0.38 28	0.38 28	0.88 28	2.67
07:08	0.39 88	0.40 88	0.79 89	2.39
07:09	1.44 174	1.39 150	16.60 176	1.00
07:10	0.40 72	0.41 72	0.87 73	2.40
07:11	0.47 25	0.48 25	0.84 25	2.43
07:12	0.44 50	0.45 50	1.67 51	2.46
08:01	0.90 0	1.04 1	12.33 24	2.28
08:02	0.41 17	0.42 17	1.32 15	2.47
08:03	0.32 118	0.33 118	0.71 119	2.84
08:04	0.28 29	0.29 29	11.71 44	3.04
08:05	0.46 88	0.50 88	12.20 78	2.16
08:06	0.44 15	0.46 15	0.80 15	2.66
08:07	0.41 138	0.42 138	2.03 138	1.91
08:08	0.67 29	0.97 29	2.50 28	2.53
08:09	0.53 109	0.56 109	11.66 19	2.60
08:10	0.38 27	0.38 27	0.73 26	2.33
08:11	0.26 125	0.26 125	0.56 125	2.07
10:04	0.89 122	0.90 123	4.29 74	1.98
12:01	0.32 36	0.32 36	0.46 37	3.36
12:02	0.34 44	0.36 44	0.61 44	4.23
12:03	0.31 11	0.31 11	0.48 12	3.53
Mean	0.50 -	0.52 -	3.66 -	
Median	0.39 -	0.40 -	0.95 -	

Table A.3: The table shows the average errors in radians for each frame when matching the extracted motion against the input motion for three different models M_1,M_2,M_3, and the matched index is also listed. All the motions listed in this table are of motion capture data of a running human. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 3.

Motion ID	Running average point error (radians) M_1 Error, Index	M_2 Error, Index	M_3 Error, Index	Period count
02:03	0.32 2	0.33 2	0.57 2	1.89
09:01	0.50 9	0.52 9	1.03 8	1.66
09:02	0.57 28	0.58 28	1.16 28	1.40
09:03	0.54 32	0.55 32	1.43 31	1.33
09:04	0.57 20	0.58 20	1.32 20	1.53
09:05	0.52 77	0.53 77	1.30 77	1.57
09:06	0.54 19	0.56 19	1.11 18	1.56
09:07	0.53 48	0.55 48	1.36 48	1.50
09:08	0.52 74	0.54 74	1.48 74	1.38
09:09	0.51 22	0.52 22	0.93 22	1.67
09:10	0.58 42	0.59 42	1.78 42	1.34
09:11	0.45 11	0.46 11	1.01 11	1.77
16:08	1.10 130	1.13 139	6.31 203	1.00
16:35	0.42 17	0.43 17	1.42 19	1.67
16:36	0.24 1	0.25 1	0.42 1	1.97
16:45	0.53 15	0.54 15	1.51 15	1.61
16:46	0.56 10	0.57 11	1.15 10	1.68
16:55	1.69 28	1.68 26	3.30 31	1.64
16:56	0.64 76	0.70 76	1.29 76	2.12
16:57	0.84 236	0.89 238	7.79 16	1.00
35:17	0.27 50	0.28 50	0.88 50	1.83
35:18	0.26 62	0.27 62	0.95 62	1.87
35:19	1.93 151	1.98 151	9.92 101	1.00
35:20	0.32 5	0.33 5	1.06 6	1.84
35:21	0.33 12	0.35 12	0.96 11	1.83
35:22	0.35 6	0.39 6	1.24 7	1.89
35:23	0.37 6	0.42 6	1.00 6	1.83
35:24	0.54 23	0.56 23	1.45 24	1.68
35:25	0.31 67	0.33 67	0.81 67	1.90
35:26	1.42 0	1.46 0	25.77 29	1.00
Mean	0.61 -	0.63 -	2.72 -	
Median	0.52 -	0.54 -	1.26 -	

Root Mean Squared Errors

- M_1 is the weighted model suggested in (3.14).
- M_2 is the positional model suggested in (3.8).
- M_3 is the unweighted model suggested in (3.9).

Table B.1: Root mean squared errors for motion capture data of a boxing human. The table shows the root mean squared error for position, velocity and acceleration for three different models M_1,M_2,M_3. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 5.

Motion ID	Boxing root mean squared error (radians)		
	M_1	M_2	M_3
	h, h', h''	h, h', h''	h, h', h''
13:17	0.85 1.05 0.75	0.69 - -	8.79 127.48 16326.26
14:02	0.38 0.40 0.33	0.28 - -	2.94 29.47 1108.90
15:13	0.39 0.37 0.36	0.36 - -	3.07 47.34 5931.75
17:10	0.38 0.41 0.25	0.15 - -	5.54 43.93 511.18
79:08	0.60 0.62 0.44	0.35 - -	6.43 43.02 1480.19
Mean	0.52 0.57 0.42	0.37 - -	5.35 58.25 5071.66
Median	0.39 0.41 0.36	0.35 - -	5.54 43.93 1480.19

Table B.2: Root mean squared errors for motion capture data of a walking human. The table shows the root mean squared error for position, velocity and acceleration for three different models M_1,M_2,M_3. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 2.

Motion ID	Walking root mean squared error (radians)		
	M_1	M_2	M_3
	h, h', h''	h, h', h''	h, h', h''
02:01	0.71 0.76 0.68	0.66 - -	8.62 75.16 10948.75
02:02	0.29 0.28 0.27	0.28 - -	0.68 9.30 1825.07
05:01	0.38 0.38 0.35	0.35 - -	3.77 55.00 13379.91
06:01	0.29 0.29 0.29	0.28 - -	10.31 77.30 8219.03
07:01	0.21 0.21 0.20	0.20 - -	0.87 14.45 3086.85
07:02	0.24 0.24 0.24	0.24 - -	0.60 15.61 4235.04
07:03	0.23 0.22 0.22	0.22 - -	0.67 17.68 5133.89
07:04	0.26 0.25 0.25	0.25 - -	1.29 20.09 5794.19
07:05	0.31 0.30 0.30	0.30 - -	0.79 21.45 8318.72
07:06	0.27 0.27 0.27	0.27 - -	1.18 15.31 3228.85
07:07	0.27 0.27 0.26	0.26 - -	0.80 15.98 4292.46
07:08	0.24 0.24 0.23	0.24 - -	0.68 14.62 3929.15
07:09	0.53 0.61 0.51	0.50 - -	16.26 141.68 15523.74
07:10	0.30 0.29 0.29	0.29 - -	0.76 14.23 2998.49
07:11	0.27 0.26 0.26	0.26 - -	0.72 13.19 2945.50
07:12	0.33 0.32 0.31	0.32 - -	1.61 16.30 2535.73
08:01	0.65 0.60 0.49	0.49 - -	12.96 117.49 11700.07
08:02	0.23 0.23 0.23	0.23 - -	1.21 17.20 3793.05
08:03	0.26 0.26 0.26	0.26 - -	0.69 17.70 3593.04
08:04	0.28 0.28 0.27	0.27 - -	11.86 93.57 12333.07
08:05	0.43 0.40 0.39	0.40 - -	12.69 104.01 10549.11
08:06	0.30 0.30 0.29	0.29 - -	0.68 14.28 2767.69
08:07	0.35 0.34 0.34	0.34 - -	2.09 27.49 7389.38
08:08	0.63 0.43 0.31	0.31 - -	2.54 25.55 3848.47
08:09	0.44 0.42 0.41	0.41 - -	11.83 84.15 7649.07
08:10	0.17 0.17 0.17	0.17 - -	0.61 11.53 2743.16
08:11	0.22 0.21 0.21	0.21 - -	0.52 18.30 5869.92
10:04	0.44 0.44 0.37	0.37 - -	3.92 48.47 9884.15
12:01	0.20 0.19 0.19	0.19 - -	0.35 8.09 1941.92
12:02	0.23 0.22 0.22	0.22 - -	0.55 10.32 2759.23
12:03	0.14 0.14 0.14	0.14 - -	0.36 8.45 2036.36
Mean	0.33 0.32 0.30	0.30 - -	3.63 36.90 5975.91
Median	0.28 0.28 0.27	0.27 - -	0.87 17.68 4235.04

Table B.3: Root mean squared errors for motion capture data of a running human. The table shows the root mean squared error for position, velocity and acceleration for three different models M_1,M_2,M_3. The results were obtained using $K = 20$, a delay embedding factor of 2 and a maximum ϵ value of 3.

Motion ID	Running root mean squared error (radians)		
	M_1	M_2	M_3
	h, h', h''	h, h', h''	h, h', h''
02:03	0.24 0.22 0.22	0.22 - -	0.53 7.11 1056.13
09:01	0.23 0.22 0.21	0.21 - -	0.86 10.45 1609.51
09:02	0.18 0.18 0.16	0.16 - -	0.93 10.49 1690.53
09:03	0.16 0.16 0.15	0.14 - -	1.30 12.36 1229.33
09:04	0.20 0.20 0.18	0.18 - -	1.12 9.33 1159.75
09:05	0.21 0.20 0.19	0.19 - -	1.12 12.89 2174.89
09:06	0.20 0.20 0.18	0.18 - -	0.89 10.68 1536.68
09:07	0.20 0.20 0.18	0.18 - -	1.18 12.70 1737.03
09:08	0.18 0.18 0.17	0.17 - -	1.34 15.48 1969.14
09:09	0.21 0.21 0.19	0.20 - -	0.78 12.59 2743.59
09:10	0.17 0.17 0.16	0.16 - -	1.68 17.04 2300.80
09:11	0.23 0.23 0.22	0.22 - -	0.89 9.65 1450.28
16:08	0.39 0.44 0.33	0.32 - -	6.39 60.69 7742.40
16:35	0.22 0.21 0.20	0.20 - -	1.34 12.78 1998.31
16:36	0.24 0.22 0.22	0.23 - -	0.42 7.01 1054.93
16:45	0.28 0.26 0.24	0.24 - -	1.40 11.58 1091.23
16:46	0.33 0.32 0.31	0.31 - -	0.94 13.25 2463.22
16:55	0.81 0.77 0.73	0.69 - -	2.61 29.65 3147.93
16:56	0.58 0.53 0.52	0.52 - -	1.24 12.69 1443.62
16:57	0.54 0.64 0.35	0.31 - -	7.77 76.36 10639.76
35:17	0.20 0.19 0.19	0.18 - -	0.84 10.48 1435.39
35:18	0.19 0.19 0.18	0.18 - -	0.92 17.27 2619.36
35:19	0.66 0.78 0.46	0.43 - -	12.24 113.79 18083.95
35:20	0.20 0.20 0.19	0.19 - -	1.05 10.70 1545.42
35:21	0.25 0.23 0.22	0.22 - -	0.97 11.32 1737.95
35:22	0.29 0.26 0.25	0.25 - -	1.24 12.24 1529.77
35:23	0.33 0.29 0.29	0.28 - -	0.95 14.40 1626.25
35:24	0.29 0.28 0.27	0.27 - -	1.39 15.60 2131.89
35:25	0.28 0.25 0.25	0.25 - -	0.82 11.32 1256.01
35:26	0.57 0.60 0.47	0.44 - -	27.94 199.86 12757.30
Mean	0.30 0.30 0.26	0.26 - -	2.77 26.06 3165.41
Median	0.23 0.22 0.22	0.22 - -	1.12 12.64 1713.78

Frame Ranges

For the boxing motions we only used a subset of the frames while we used all frames for the running and walking motions. The frames used for the boxing motions are listed in table C.1.

Table C.1: For some motions we only use a subset of frames. For those motions we list the frames used here.

Motion ID	Frame Start	Frame End
13:17	200	450
14:02	130	250
15:13	140	420
17:10	180	250
79:08	140	220

Online Video Resources

In this chapter we have collected a number of online links for videos we have rendered of motion 02:03, 06:01 and 14:02. For each motion we have three videos, one displaying a looped version of the motion capture data, one displaying the result from using model M_1 and one for using model M_2.

Table D.1: The table contains links to renderings of three different motions. The videos helps illustrate the difference between model M_1 and M_2 in addition to allow for comparing the output motion to the input motion capture data. The motion capture data was captured at 120 fps and is playedback at 30 fps and the extracted motions are generated at 120 fps and played back at 30 fps as well. The motions are playedback at reduced speed to help highlight the difference between the models as well as errors in the extracted motions.

Motion	Link
02:03 raw data	https://www.youtube.com/watch?v=7_Gu9DrOyKo
02:03 M_1	https://www.youtube.com/watch?v=-67BqSJODIk
02:03 M_2	https://www.youtube.com/watch?v=kU_c7t1cCVY
06:01 raw data	https://www.youtube.com/watch?v=b0EIf1XCRgo
06:01 M_1	https://www.youtube.com/watch?v=_66NMZg8TFM
06:01 M_2	https://www.youtube.com/watch?v=3tLLzjc7T_A
14:02 raw data	https://www.youtube.com/watch?v=Ry9dO09xEpk
14:02 M_1	https://www.youtube.com/watch?v=-WQNnBqsg2U
14:02 M_2	https://www.youtube.com/watch?v=pzRZSmWuWak