Beyond Mocap
Animating Soccer Players Based on Positional Tracking Data

Daniel João Bandim Faustino

Master Thesis
ICA-3443140
Utrecht University
Department of Information and Computing Sciences

Supervisors:
Project Supervisor: dr. ir. Arjan Egges
Second Supervisor: dr. Zerrin Yumak
Daily Supervisor: ing. Dittmar Tukker

August 29, 2016

Abstract

Due to current advances made in Virtual Reality solutions, we can now simulate real world professional soccer matches in such a way that anyone can relive a match as if they were on the pitch. Such simulation can be built through the use of motion capture clips and positional tracking data captured during a match to replicate the player's motions on the virtual field. However, the automation of motion capture usage is limited due to the lack of pose information.

We propose a framework, Beyond Mocap, which takes these two data sources and, together with an annotation system, automatically generates the desired motion. Beyond Mocap works by taking user provided descriptions of the moves performed by the soccer player and matching them with an annotated motion dataset. To ensure the virtual character follows the same path and with minimal foot-skating, we present a novel approach of travelled distance matching by taking advantage of the fact that locomotion is cyclical and, as such, it can be made longer or shorter by adding or removing locomotion cycles. A simple method of motion dataset duplication by mirroring is also provided, enabling a wider range of available motion capture.

Beyond Mocap outputs motions that are more realistic-looking, have a higher degree of naturalness and are smoother than the motions produced by the current solution, while at the same time following the path defined by the positional tracking data and with minimal foot-skate.

Table of Contents

1. Introduction

Beyond Sports is a virtual reality (VR) soccer simulation, developed by Triple, that uses match data[1] to simulate real world professional soccer matches. Using a VR headset such as the *Oculus Rift*, coaches, players and even fans can relive a match through their own eyes, or through the eyes of any player on the field.

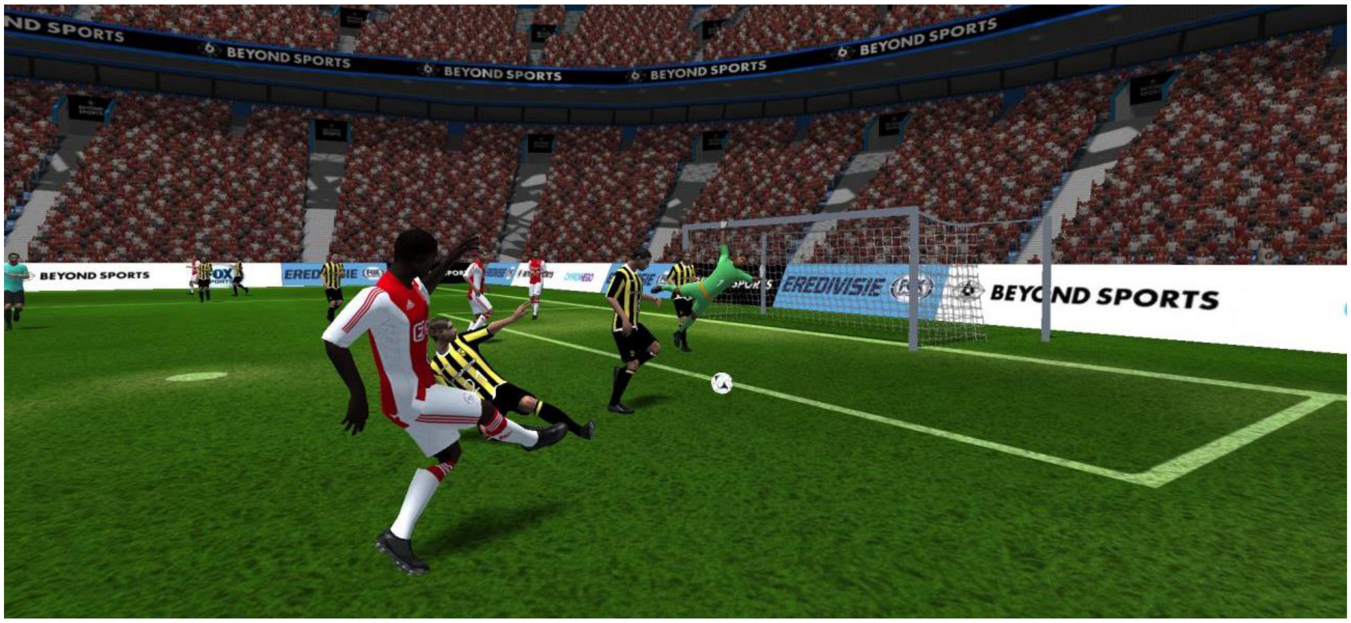


Figure 1: Screenshot taken from Beyond Sports.

This real match data, however, does not provide any information about the pose of each player throughout the match. Due to this limitation, the virtual soccer players in Beyond Sports only have basic movement, such as walking and running.

The current solution in Beyond Sports suffers from the issues described below**:**

- The animations of the virtual soccer players are not realistic-looking due to the absence of non-locomotion animations such as kicks or headers. And when these are present, there is heavy foot-skating and glaring continuity breaks in transitions between locomotion and non-locomotion animations.
- The solution currently used to insert non-locomotion animations is prohibitively time-consuming due to the amount of manual work required.

As a starting point, the team behind Beyond Sports had previously captured dozens of motion clips to be used for the improvement of the animation system. The goal is then to improve the realism of the simulation by developing an automated approach that uses motion capture (mocap) together with real match data to synthesize motions for soccer players. It should

[1] Positional tracking data captured during a soccer match. It consists of the locations of all players and the ball, sampled at 10Hz.

require minimal input from the user, be faster than current solutions, and match as closely as possible to the moments from the soccer match it is replicating, as well as field positions and moves performed by the players.

The focus is on offline data-driven motion synthesis approaches since the positional tracking data is completely provided in advance. By taking an offline approach we can simplify the solution and remove the need for real-time computation; and by using mocap data (data-driven approach) we can have a short waiting time from the moment the match data has been loaded to visualizing the match. Another advantage of using an offline global search method has to do with the ability to account for global planning during synthesis, which is necessary for several soccer specific motions such as a free kick. In this example, the player might have a long setup phase by first taking a few steps back to gather momentum before the kick. Finally, a data-driven approach was chosen over a more realistic physics-based approach because it would have been too computationally intensive [Geijtenbeek and Pronost 2012] for the desired low turnaround times.

This project was conducted as part of an internship at *Triple* situated in Alkmaar, The Netherlands.

1.1. Research Questions

By taking into account the problems described previously, we pose the following questions:

Q1: *To what extent can we synthesize an animation for a virtual soccer player through the use of motion capture and positional tracking data which is more realistic-looking than the current solution?*

Q2: *To what extent can we automate the process so that it takes less time to obtain an animation for a virtual soccer player than the current solution?*

The main contributions to research in the field will therefore be on the automatic generation of realistic-looking animations, in an offline manner, which match any constraints in time and space throughout the duration of a soccer match, with minimal user input.

1.2. Report Structure

In the next chapter, related work in the field is presented and split into sub chapters for synthesis (Chapter 2.1) and editing techniques (Chapter 2.2). This is followed by the presentation of the approach chosen for motion synthesis and an overview of the steps involved in Chapter 3. Since several steps in Beyond Mocap are performed before and after the main synthesis phase, the approach description is further divided into pre-processing (Chapter 3.1), synthesis (Chapter 3.2) and a final post-processing phase (Chapter 3.3). Chapter 4 covers the implementation highlights and intermediate files' specifications as well as the external tools and scripts necessary for the functioning of the Beyond Mocap tool. A small pilot study was performed to evaluate the synthesis approach and both its results and analysis are presented in Chapter 5. This report ends with a conclusion in Chapter 6.

2. Related Work

In a virtual simulation with human movement, it is important the human characters move with smooth and realistic-looking animations. Synthesizing such realist-looking animation becomes harder to do when applied to interactive situations such as video games, since the character needs to react to the AI, player-input, surrounding environments and interaction with other characters quickly and accurately.

In the following sub-chapters, we will cover the various methods through which human movement can be synthesized (Chapter 2.1), giving more detail to data-driven approaches; and the modifications that can be performed on existing motions in order to adapt it to pre-defined constraints (Chapter 2.2). A final sub-chapter introduces definitions that will be used throughout this report (Chapter 2.3).

2.1. Motion Synthesis

There are various ways of obtaining a realistic looking motion segment: procedurally simulated, through the application of mathematical formulas; in a physically-based simulation, with the application of the laws of physics; and data-driven, by using clips of motion captured animation.

Procedural methods can include a large number of parameters to synthesize motion, but are typically supplanted by either physically-based or data-driven techniques. These latter two approaches can synthesize motion with a realism the former is not able to match. This is the case because of two main reasons: it is hard to make a synthesized motion comparable in detail to mocap clips using only mathematical formulas; and to maintain physical naturalness, this has to be explicitly specified in the model for all possible parameter instances [Van Welbergen et al. 2010]. A physics-based approach has the advantage of possibly allowing the highest degree of realism but at a high computational cost. On the other hand, a data-driven approach provides a realistic enough motion for most applications with a low requirement on computational power.

2.1.1. Data-Driven Approaches

Research in the area of motion synthesis with a data-driven approach, also known as example-based, is mainly focused on the data structures which allow motion generation (Chapter 2.1.1.1) and how and when to perform transitions between each motion clip (Chapter 2.1.1.2).

The main disadvantage with data-driven approaches is the high cost of generating enough motion variations that can cover the whole range of motions a character can have, as synthesis is dependent on what has been previously captured. The techniques used to address this issue, and introduce modifications to example motion clips, are discussed in Chapter 2.2.

Finally, as a mocap database grows with new example motion clips added to its library, it becomes increasingly impractical to manually search the whole database for a specific clip. Motion retrieval techniques can mitigate this by complementing the various data-driven motion synthesis approaches. These techniques help to automatically and accurately search and analyze the mocap database and retrieve example motions which match the desired specification. Motion retrieval is outside of the scope of this project, and thus can be considered for future optimization as the motion database is relatively small (under 100 motions) and can be manually annotated. The study performed in [Pejsa and Pandzic 2010, chap.4.1] provides more information in the area of motion retrieval.

2.1.1.1. Data Structure

Currently, the most widely used graph structures are *Motion Graphs*, proposed in [Kovar, Gleicher, et al. 2002]. In this paper, the authors automatically construct a flat directed graph structure consisting of original motion clips and their transitions, with the final motion synthesized through local search. Other similar approaches using graphs appeared in the same year: in [Arikan and Forsyth 2002] a motion database was also used to build a hierarchical motion graph and randomized search to generate the desired motion; and in both [Lee et al. 2002; Li et al. 2002] modified versions of motion graphs are presented, using statistical models and motion textures respectively.

Following the definition in [Kovar, Gleicher, et al. 2002], a motion graph is a directed graph comprised of *edges* and *nodes*. An *edge* is a motion segment: either a portion of one of the example clips from the mocap database or a transition between two such clips, usually where poses are similar. A *node* is a choice point for connection: each outgoing edge is potentially the successor to any incoming edge. Synthesizing a motion requires us to perform a *graph walk* (placing edge after edge).

To build a motion graph, as described in [Pejsa and Pandzic 2010], we first need to compare every motion clip, frame-by-frame, using a *frame distance metric* (see Chapter 2.3.2). The output of which is a 2D grid of frame distances for each pair (Figure 2). The grid is then searched for local minima: if a minimum is below a user-specified threshold, then the corresponding frame pair is a transition point. On each transition point, two edges are created: transitions of first motion to second motion and vice versa. Finally, the graph is pruned to ensure that it is well-connected, eliminating poorly connected nodes (dead ends and sinks). The algorithm from [Tarjan 1972] can be used to eliminate any node that is not part of the largest strongly connected components.

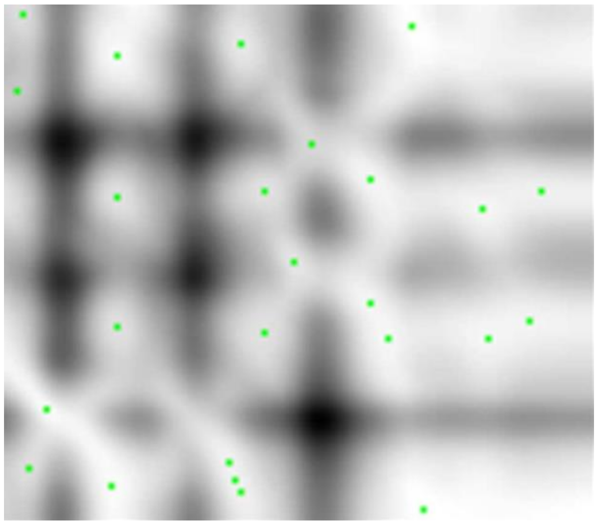


Figure 2: 2D grid of frame distances for a pair of motions. The green points are local minima and, therefore, transition points [Kovar, Gleicher, et al. 2002].

Using a whole database of motion clips can result in an unnecessarily large graph with duplicate transitions. Manually choosing which clips to include would be too work-intensive and could result in a graph that is too small or poorly connected. [Zhao et al. 2009] propose a semi-automatic method for constructing a minimum-size graph with enough motions to synthesize acceptable motion sequences. This can be improved with the technique presented in [Zhao and Safonova 2008] which increases connectivity by including interpolated poses (blends of original motion clips) in the graph. Another approach to improving connectivity was described in [Ren et al. 2010] where instead of using interpolation, *optimization-based graphs* were built. These graphs allow for a higher number of complex and realistic-looking transitions when compared to a standard motion graph.

More recently, a different representation of motion data, *motion fields*, has been presented in [Lee et al. 2010]. A motion field is a "mapping which associates each possible configuration of a character with a set of motions describing how the character is able to move from their current state". This approach has the advantage of allowing quick responses to user input and is thus indicated for online synthesis. However, since it requires a large mocap dataset for proper results, it is impractical when dealing with a large space of motions. This issue can be addressed by combining motion graphs and motion fields [Xing et al. 2014]. Motion fields provide the agility to react to user input and the environment, and the structure provided by motion graphs guarantee the fast connectivity and smooth transitions between motions.

2.1.1.2. Search Method

Since motion graphs are inherently unstructured, the resulting motion sequence will be dependent on the type of graph walk that is performed. Two main methods exist: *local search* and *global search*. Local search methods generate the sequence incrementally and are better suited for online synthesis but with the disadvantage of not generating globally optimal motion sequences, with the possibility of failing to achieve the desired objective. This can be addressed by expanding the search horizon with some sort of *motion planning*. Global search methods generate the whole sequence at once and are better suited for offline synthesis.

Local Search Methods

In [Kovar, Gleicher, et al. 2002] the branch and bound algorithm is used for synthesis. The graph is explored up to a specific horizon and generates multiple graph walks simultaneously. This allows for motion sequences where the character moves along a desired path or reaches a desired target. By modelling motion as a first-order Markov process and assigning higher probabilities to better-quality transitions, the authors of [Lee et al. 2002] are able to expand the motion graph with an additional layer. [Li et al. 2002] propose another local search method where recurring patterns in example motions are identified. These are then represented as linear dynamic systems which capture dynamic properties of the patterns, also called textons. A motion sequence is synthesized by defining the start and end textons.

Global Search Methods

The leading approaches in global search methods are those by Arikan et. al. in [Arikan and Forsyth 2002] and [Arikan et al. 2003]. In the first paper, motion graphs are represented differently than those defined by [Kovar, Gleicher, et al. 2002]: nodes represent example motions and edges are the transitions between those motions. Motion is synthesized by employing a randomized search. In the second paper, search is done directly on previously annotated motion clips. Dynamic programming is later applied to refine the initial coarse motion.

Improving Local Search Methods

Local search methods' limitations in obtaining an optimal motion sequence can be addressed in several ways. Since motion graphs are unstructured, a possible approach would be to make sure they are structured before the search is performed, as described in [Gleicher et al. 2003]. Another approach would be to precompute the search results [Srinivasan et al. 2005]. Finally, with *motion planning* techniques it is possible to obtain optimal motion sequences that meet complex high-level goals, through the use of either probabilistic roadmaps, precomputed search trees or reinforcement learning [Lee and Lee 2004; Lee et al. 2010]. Motion planning methods consider the entire movement space and generate motion sequences that are close to being globally optimal, meaning they will potentially achieve the desired goal in the best, fastest way possible [Pejsa and Pandzic 2010; Van Welbergen et al. 2010]. In an example taken from [Lo and Zwicker 2008] of a character having to grab an object (Figure 3), a local search would not plan into the future and fail to select a motion that would have the character meet the objective (leftmost figure). With motion planning, the objective of grabbing an object is met with a detour (center figure) when only example motion clips are used. The optimal solution is obtained when using both motion planning and synthesized motions (rightmost figure).

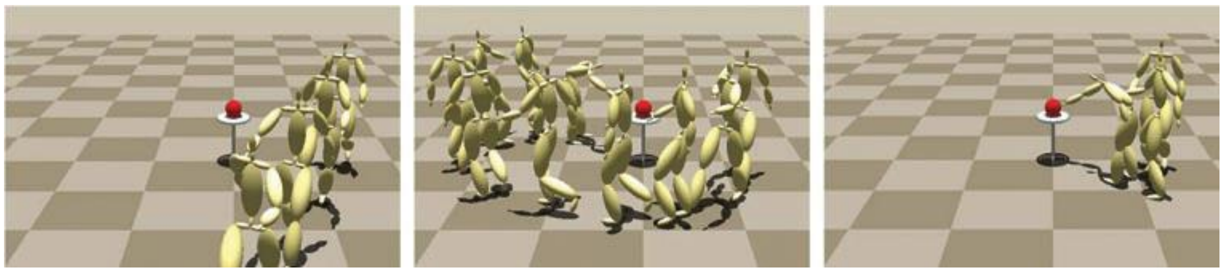


Figure 3: Comparison of motion synthesis methods. Left: local search; Center: motion planning with example motion clips only. Right: motion planning with synthesized motion.

2.1.2. Other Approaches

Besides motion synthesis that is physically-based or obtained through procedural simulations, which are not covered here (see [Van Welbergen et al. 2010] for an overview on the subject), there are also *hybrid approaches*. They combine the advantages of both data-driven (naturalness) and physically-based techniques (physically correct motions and interaction with the environment).

An approach presented in [Shapiro et al. 2003] switches between both techniques. The character is animated through data-driven techniques and, when there is an interaction with the environment, the physical virtual character simulation takes over. A similar approach is used by NaturalMotion's animation system and also used in [Mandel 2004], where the transition from data-driven to physically-based is done whenever the character enters a falling motion.

Physically-based character simulation can also be minimal and used only as transition between a motion clip used before physical impact and reaction motion clip, as presented in [Zordan et al. 2005] and [Zordan et al. 2007].

2.2. Motion Editing

We would need a large mocap database to account for all possible motions (such as kicking a soccer ball at different heights) in order to have enough motion clips that could be applied to all real-world situations. Since this is not feasible, due to the associated cost and how time-intensive it would be to capture the whole range of motions a soccer player can have, there needs to be some sort of editing performed on motion clips. Two types of motion editing can be performed: either by creating new motions by modifying a single motion clip, *motion adaptation* (Chapter 2.2.1); or by creating new motions by blending between multiple motion clips, *motion blending* (Chapter 2.2.2).

To perform motion editing, animation parameters need to be provided. *Parameterization* is the process of selecting animation parameter values (like blend weights, stiffness gains, Principal Component Values and so on) to be able to enforce certain constraints. An example of such constraints is requiring the foot to be at a certain location at a desired time (also known as eliminating foot-skate). Such parameters can also be more abstract, such as emotion or physical state of the virtual character [Van Welbergen et al. 2010].

Techniques like the ones used in motion adaptation approaches do not need extra work for parameterization: the animation parameters are directly used as control parameters. As an example, [Callennec and Boulic 2004] provide a PFIK+F framework that handles multiple (geometric) constraints and resolves possible conflicts using prioritization.

However, this is not the case with motion blending techniques, creating the need for a mapping of control parameters to animation parameters. A parameterized motion space is defined by a set of example motion clips with known parameter values. During animation, blend weights are computed from parameter values associated with example motion clips. Since these clips are irregularly distributed in the motion space, scattered data interpolation needs to be used [Pejsa and Pandzic 2010] (Chapter 2.2.2).

2.2.1. Motion Adaptation (Enforcing Pose Constraints)

Originally, techniques from the field of signal processing were used for motion adaptation since a motion can be defined as a function of time. The authors of [Bruderlin and Williams 1995; Witkin and Popovic 1995] use a technique called *displacement mapping,* locally modifying the signal while preserving its continuity and global shape, but with constraints satisfied only at specific key frames. These constraints can, however, be enforced on every frame where desired, followed by a smoothing of the synthesized motion through a filter using B-splines [Lee and Shin 1999]. This type of motion adaptation is designated as "Per Frame Inverse Kinematics + Filtering" (PFIK+F). In the same line, [Choi and Ko 2000] use an IK solver to enforce constraints at each frame. An overview of constraint-based motion editing methods up to the year of 2001 can be found in [Gleicher 2001]. Analytical and numerical IK solver algorithms for limbs are discussed and a combination of them proposed in [Tolani et al. 2000].

Different constraints can be in conflict, however. If one of the constraints is not achievable, or some constraints are not simultaneously achievable, a priority strategy will sort them and satisfy the most important constraints first. This is the approach presented in [Callennec and Boulic 2004], where the original motion clip can be deformed using prioritized constraints, allowing an arbitrary number of priority-layers and constraints to be used, and control for the position of the center of mass is included so the adapted motion is physically plausible. In this framework, each frame is individually deformed using a PFIK + F algorithm.

In [O'Brien et al. 2011], an approach using a vertex-based representation of the character's skeleton, and mass distribution at each frame, also maintain the appearance of physical plausibility. Constraints are, therefore, based on points (rather than rigid bodies) to formulate a space-time optimization problem that solves for edited character motion. Multiple frames are coupled together to enforce certain features of motion such as smooth acceleration and dynamical correctness.

As with motion synthesis, there are also physically-based approaches to motion adaptation, which are especially desirable for highly dynamic motions. The technique described in [Sok et al. 2010] uses trajectory optimization based on normalized dynamics to allow the user to control the momentum and force, while at the same time maintaining the landing position and physical plausibility of the original motion clip.

2.2.2. Motion Blending (Transitions and Interpolation)

Motion blending allows the creation of new motions by performing a weighted combination of two or more similar base motion clips. The resulting motion is called a *blend*. Blends can be distinguished by the number of motion clips being blended. An *interpolation* is a blend of two or more motions, but when the blend is performed between only two clips, we get a special interpolation case, also called *transition*.

2.2.2.1. Transitions

When blending, the example motions should be aligned in time and space. In transitions where blend weight change monotonically from 1 to 0, these blending constraints, usually on end-effectors, are easily dealt with by enforcing the constraint on the motion with greater blend weight and ignoring it on the other motion, as presented in [Kovar, Gleicher, et al. 2002] and [Ashraf and Wong 2001]. Inverse kinematics can also be used to calculate proper positions for the end-effectors [Kovar, Schreiner, et al. 2002]. However, in parametric motions they can change in an unpredictable manner. As an example, in a walking motion one of the feet must always be planted on the ground, but that is not the case with running motions, and such conflict can arise when blending between running and walking. High-level parameters such as speed can also be mapped to blend weights to control the blending of the base motions [Pejsa and Pandzic 2010]. The most complete method is presented by [Kovar and Gleicher 2003] where *registration curves* are used to automatically determine the time, space and constraint correspondences between a family of example motions.

To transition between discrete motions, pairs of similar frames must be identified in the example motion clips, followed by the creation of transitions centered around these frames. However, parametric transitions are more complicated since the number of possible transition points is infinite, thus requiring a different transition technique [Pejsa and Pandzic 2010].

Transitions between discrete motions

Transitions of length L are performed by blending between motions starting $(L-1)/2$ frames before and ending $(L-1)/2$ frames after the transition point. The second motion is first rotated in the 2D plane to align it with the first motion. Root positions P are blended using linear interpolation (lerp) and joints' orientations q using spherical linear interpolation (slerp) [Pejsa and Pandzic 2010]. For every transition frame p:

$$P_{root,p} = \alpha(p)P_{root,i+p} + (1-\alpha(p))P_{root,j-L+1+p} \tag{1}$$

$$q_p = slerp(q_{i+p}, q_{j-L+1+p}, \alpha(p)) \tag{2}$$

Where i and j are start and end frame indexes, p is relative frame index in the transition window ($0 \leq p \leq L$) and $\alpha(p)$ specifies the blend weights:

$$\alpha(p) = 2\left(\frac{p+1}{L}\right)^3 - 3\left(\frac{p+1}{L}\right)^2 + 1 \tag{3}$$

This simple transition scheme is suitable for short transition intervals ([Kovar, Gleicher, et al. 2002] use $L \approx 0.33s$). Longer or variable transition lengths are proposed in different studies: [Rose et al. 1996] and [Zordan et al. 2005] take *character dynamics* into account when generating transitions; [Wang and Bodenheimer 2008] use new methods to calculate *optimal blend weights and durations* and [Kovar and Gleicher 2003] describe *registration curves* for correct blending of N motions and generation of high-quality transitions.

In several other approaches [Arikan and Forsyth 2002], [Gleicher et al. 2003], [Ikemoto et al. 2007], [Callennec and Boulic 2004], [Oshita 2008], [Ménardais et al. 2004] and [Lee et al. 2002], where the transition period L is $1s \leq L \leq 2$, *displacement mapping* is used. This technique can be used to preserve fine details even over long transition periods.

Transitions between parametric motions

Three schemes have been proposed for parametric transitions: *stitching motions together* using techniques of Snap-Together Motions (STM) [Gleicher et al. 2003], where a motion graph is semi-automatically constructed. This idea is expanded in [Shin and Oh 2006] with fat graphs but with the disadvantages of having less structure in parametric motion. In [Heck and Gleicher 2007], the authors address this issue by using the techniques presented in [Kovar and Gleicher 2004] and [Kovar and Gleicher 2003] to build parametric motions and then organize them into a *parametric motion graph*. Nodes of the graph correspond to parametric motions, while edges represent transitions between them. Its disadvantages are that an edge between two parametric motions can be generated only when every sample motion of the source motion space can transition to at least one subspace of the target motion space; and transitions are only possible from the end of the source motion which can be a problem when parametric motions are long.

In contrast to the two online techniques mentioned above, a more powerful technique that can only be used in offline motion synthesis is described in [Safonova and Hodgins 2007], where an A* search of interpolated motion graphs is performed.

2.2.2.2. Interpolation

Several interpolation methods exist, such as *Scattered Data Interpolation with RBFs* which was originally presented in [Rose et al. 1998]. Further improvements were made but it remained very limited and produced poor-looking blends for extrapolation (when input parameter values are far from sample values). To resolve these issues, *k-Nearest-Neighbors Interpolation on Densely Sampled Motion Spaces* was proposed in [Kovar and Gleicher 2004]. They used the method of match webs to retrieve similar motion segments to a query motion, followed by the creation of *registration curves* [Kovar and Gleicher 2003] for these segments. An analysis and comparison of these two and two other motion blending techniques for interpolation (Barycentric interpolation and Inverse Blending optimization) are presented in [Feng et al. 2012].

2.3. Definitions

2.3.1. Motion Specification

A character is animated via its skeleton: hierarchical structure composed of bones connected at joints. Each joint inherits a 3D transformation (translation and rotation) from the parent joint. Motion can therefore be defined as a continuous function by interpolating between frames:

$$m(f) = \left(p_r(f), q_1(f), \ldots, q_n(f)\right) \tag{4}$$

Where f is the frame index; p_r is the position of the root joint at frame f and defined as a 3D vector $[x, y, z]$; and q_i is the joint i orientation at frame f and defined as a quaternion [Pejsa and Pandzic 2010]. The same is applicable when defining motion as a function of time:

$$m(t) = \big(p(t), q_1(t), \dots, q_n(t)\big) \tag{5}$$

2.3.2. Frame Distance Metric

In order to compare two motions for similarity, a frame distance metric was defined in [Kovar, Gleicher, et al. 2002]. It computes the difference between character poses at two frames and takes the following into account:

- *Joint velocities and accelerations at both frames*. Given frames a and b the metric compares windows of frames of fixed length, centered on frames a and b.
- *Joint influences*. Since not all joints affect the motion's appearance equally, different joints have different weights w.
- *Position and orientation in the 2D plane*. Because there's no difference between two walking motions rotated by 180 degrees in the 2D plane, a second motion can be aligned in the 2D floor plane to the first motion using a 2D transformation T_{θ,x_0,z_0}.

$$D(a, b) = min_{\theta,x_0,z_0} \sum_i w_i \left\| p_i(a) - T_{\theta,x_0,z_0} p_i(b) \right\|^2 \tag{6}$$

Where p_i is the world position of joint i. This equation can be solved analytically using equations given in [Kovar, Gleicher, et al. 2002].

2.3.3. Motion Blending

$B(t)$ is a frame of a blend constructed from N base motions $(m_1, m_2, \dots, m_N)$ and an N-dimensional weight function $w(t) = [w_1(t), w_2(t), \dots, w_N(t)]$. The weight determines the influence of the corresponding base motion clip on the final motion. For example, if $w_k(t_0) = 1$ and $w_i(t_0) = 0$ for every $i \neq k$, then $B(t_0) = m_k(f_{t_0})$. In a transition, the sum of the weights of each motion is 1 at all times: $w_1 + w_2 = 1$, $w_1(t)$ smoothly changes from 1 to 0 and $w_2(t)$ smoothly changes from 0 to 1. In an interpolation, $w_i(t) = const$.

Lerp is used for root positions. They are blended by computing the weighted average of root positions or velocities in base motion clips:

$$p_{blend} = w_1 p_{r1} + w_2 p_{r2} + \dots + w_N p_{rN} \tag{7}$$

Using quaternions for joint orientations, blending between two motions is performed with slerp:

$$q_{blend} = slerp(q_1, q_2, w) = \frac{sin\big((1-w)\theta\big)}{sin\,\theta} q_1 + \frac{sin(w\theta)}{sin\,\theta} q_2 \tag{8}$$

where $\theta = \cos^{-1}(\,q_1 \cdot q_2)$.

But when blending between N ($N > 2$) motions, the most widely accepted definition of the weighted average $\mathrm{q_{blend}}$ of N quaternion orientations q_i according to [Pejsa and Pandzic 2010] is given in [Buss and Fillmore 2001] as:

$$\sum_i w_i \log(q_{blend}^{-1} q_i) = (0,0,0) \tag{9}$$

where log is the logarithm map operator which maps a unit quaternion to its corresponding 3-DOF rotation vector. $\log(q_{blend}^{-1} q_i)$ represents a displacement rotation vector between orientations represented by q_i and q_{blend}. A blending technique is proposed in [Park et al. 2002] where all orientations q_i are transformed into displacement vectors with respect to a reference orientation q_*:

$$v_i = \log(q_*^{-1} q_i) \tag{10}$$

where v_i are the displacement vectors. q_* is chosen so that it is as close as possible to all blended quaternions q_i and may be computed using a least-squares method. Once displacement vectors have been determined, they are blended with lerp:

$$v_{blend} = w_1 v_1 + w_2 v_2 + \cdots + w_N v_N \tag{11}$$

The blended quaternion is then derived by computing the exponential map of the displacement vector v_{blend} back into quaternion space and applying it to the reference quaternion:

$$q_{blend} = q_* \exp(v_{blend}) \tag{12}$$

Using registration curves

In [Kovar and Gleicher 2003] registration curves are used to perform blending. A registration curve is composed of a timewarp curve $S(u)$, alignment curve $A(u)$ and constraint match information $C(u)$. To create a blend frame $B(t_i)$:

1. Determine the current position $S(u_i)$ on the timewarp curve for current time t_i.
2. Position and orient the frames at $S(u_i)$ using the 2D transformations at $A_j(u_i)$.
3. Blend the timewarped and aligned frames using equations (7), (10), (11) and (12).
4. Query $C(u)$ to determine and enforce active constraints. Weighted averaging is used to determine when the constraint is active.

2.4. Conclusion

In the case of Beyond Sports, several dozen motion captures are provided in advance, making a good case for a data-driven approach over one that is physically-based. Therefore, we can say the first step starts with deciding between an online or offline synthesis based on the constraints at hand. If the synthesized motion is going to be used in an interactive environment where user input guides the synthesis and where the end-goal is unknown, an online approach is the indicated starting point. Otherwise, as is the case with the motions to be synthesized for Beyond Sports, an offline approach is preferred since it is a simpler method of synthesis with fewer constraints in terms of computation time.

Global search methods are recommended for offline synthesis since they generate the whole sequence at once. In case the motion should be synthesized in an online fashion, however, local search methods are better suited, together with some form of motion planning if an optimal solution is desired.

Furthermore, as shown in Chapter 2.2, variation in the motion capture database is not critical since there are numerous techniques focused on increasing the dataset for cases where the available mocap is limited. These techniques can either generate completely new variations of a single motion or combine one or more source motions to synthesize something in between. The latter techniques are also useful when a smooth transition is required from one motion to another.

3. Motion Synthesis Approach

In this chapter the synthesis approach will be presented. The core of the approach described in this chapter works with a motion capture database. In order for this database to be used for the synthesis process, our approach analyzes the motions contained therein, segments their content and uses the obtained general motion information to splice them together into new motions.

Beyond Mocap is composed of three main phases: the pre-processing phase where the annotation vocabulary is defined and the mocap database annotated and mirrored (Chapter 3.1); the main synthesis phase where the actual synthesis takes place (Chapter 3.2); and a final post-processing phase which cleans up any discontinuities left in the motion and finalizes the continuous position constraint matching (Chapter 3.3).

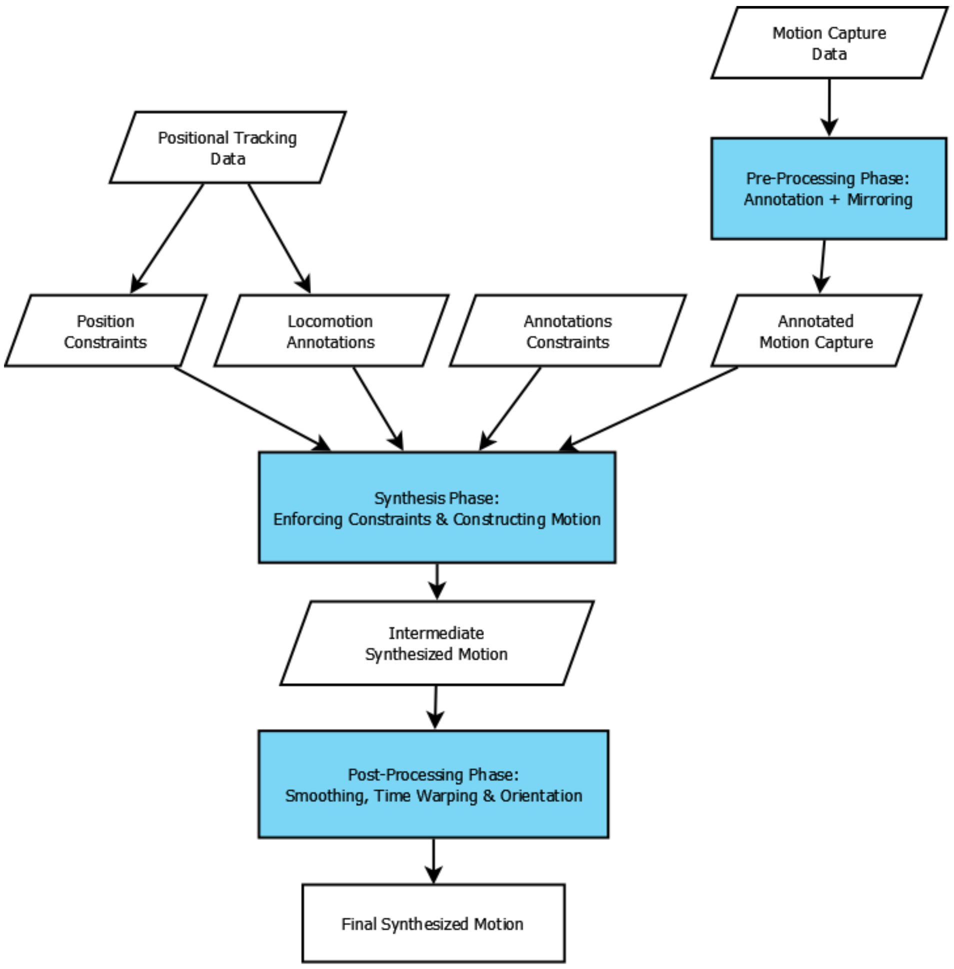

Motion Capture Data
Positional Tracking Data
Pre-Processing Phase: Annotation + Mirroring
Position Constraints
Locomotion Annotations
Annotations Constraints
Annotated Motion Capture
Synthesis Phase: Enforcing Constraints & Constructing Motion
Intermediate Synthesized Motion
Post-Processing Phase: Smoothing, Time Warping & Orientation
Final Synthesized Motion

Figure 4: Beyond Mocap Framework.

Since the positional tracking data provides only translation information for the soccer player, lacking any height or limb position data, it needs to be augmented with some sort of pose or motion information. A simple and yet robust approach is to adopt an annotation system as described in [Arikan et al. 2003] as a starting point. With such a system, as long as the vocabulary is clearly defined beforehand (Appendix A), it is possible to specify what movements the soccer player should perform at any given time, while allowing for long term planning when using global search method (Chapter 2.1.1).

An annotation example would be to state the soccer player is running for the first 8 seconds, then dribbles with the ball for 2 seconds, kicks it to score a goal and finally celebrates the goal for 5 seconds. Due to it being a global search method, the algorithm will ensure the setup phase before the kick as well as the post-kick phase (regaining balance for example) will be as long as necessary, providing a better result than a local search method.

Like the framework of [Arikan et al. 2003], the proposed approach will synthesize motions by selecting frames from the mocap database which, when concatenated, match the annotations **(annotation constraints)** and are continuous by ensuring the absence of noticeable cuts between the concatenated frames **(continuity constraint)** while enforcing user selected frames to occur at defined moments **(frame constraints)** and at the desired spatial position **(position constraints)**.

Due to the fact the mocap database will not include every single possibility of annotation composition, the synthesis itself is designed as an optimization problem, to be solved by application of dynamic programming with the objective to choose the best frames that respect all constraints listed above.

The method described here improves on the reference paper's framework to allow for continuous constraints to the positions as opposed to discrete constraints, with the advantage of forcing a motion to follow a specific path as described by input data. Although the positions obtained from the positional tracking data are sampled at 10Hz, they are referred to as continuous constraints since the position of the character in the synthesized motion should match the reference positions at all times. Beyond Mocap also improves on [Arikan et al. 2003] work by automatically providing locomotion annotations for the motion to be synthesized based on the positional tracking data; and by ensuring exact matches on input annotations, as opposed to "close enough", enabling the synthesized motion to replicate as closely as possible the movements of the real soccer player.

3.1. Pre-Processing

In the pre-processing phase, the mocap database and the motion to be synthesized are fully annotated (Chapter 3.1.1). We also present a simple technique that duplicates the motion dataset by mirroring the original animations (Chapter 3.1.2).

3.1.1. Annotating

Annotations will serve to describe the motions in the database as well as to specify what movements the soccer player should perform in the motion to be synthesized. Therefore, two sorts of annotations need to be created: per motion in the mocap database (**mocap annotations**) and per soccer player (**desired annotations**). The desired annotations, **annotation**

constraints, are what will be fed into the synthesis algorithm in order to choose the best matching motions (annotation file specification in is presented in Chapter 4.2.2).

Such annotations are derived from an annotation vocabulary. The vocabulary (Appendix A) is designed by taking into account the motions present in the database and the soccer context. As an example, if the player in the real match runs for the first 8 seconds, dribbles with his right foot for 2 seconds and then shoots the ball at a low height with the top of his right foot, then the motion to be synthesized should have its first 8 seconds annotated with `running`, the following 2 seconds with `right dribble` and finally with `top kick low right`.

The motion dataset was manually annotated due to its small size: about 75 motions with an average duration of 5-7 seconds each. It is possible, however, to employ machine learning to lessen the amount of manual work needed for annotation of larger databases as presented in [Arikan et al. 2003], where the authors use Support Vector Machine classifiers. Once the full mocap database is annotated, it does not need to be annotated again for further synthesis.

Beyond Mocap departs from the approach taken by [Arikan et al. 2003] regarding the annotation of locomotion when specifying the **desired annotations** for the motion to synthesize. In the reference paper, the user has to manually state when the character is running or walking for example, but for the most time throughout a soccer match, the player will be walking, jogging, running or in another form of locomotion, with only a small amount of its time taken by other types of motion. Since we can extract movement speed from the positional tracking data, locomotion annotation is automated, greatly reducing the time necessary to annotate a player timeline. Taking Beyond Sports' mocap database into account, **locomotion** is any cyclical motion which translates the character by an almost constant distance: walking, jogging, running, sprinting and dribbling.

Automatic locomotion annotation works by first specifying which mocap to use for which locomotion. The average speed of all locomotion mocap is first computed by calculating the total distance travelled divided by the motion's total duration in seconds. We then define thresholds between different type of locomotion as:

$$threshold_{A,B} = s_A + \frac{(s_A - s_B)}{\omega}$$

Where A and B are locomotion mocap, $s_B > s_A$ are the average speed of each locomotion and ω a constant defining where the threshold should be placed relative to both mocap speeds. We set $\omega = 4$ in order to limit the usage of slower locomotion mocaps on higher movement speed. These thresholds are then used together with the movement speed extracted from the positional tracking data in order to perform automatic locomotion annotation:

```
1.  FOR EACH frame $f$ IN total number of frames of motion to synthesize $m$
2.    COMPUTE $s_m(f) = {\|\vec{p}(f-1) - \vec{p}(f)\|}/{\Delta t(f)}$
3.  FOR EACH $threshold_{a,b}$ IN all computed thresholds from $threshold_{0,slowest}$ to $threshold_{i,fastest}$
4.    IF $s_m(f) < threshold_{a,b}$
5.      $A(f) = A_a(f)$
6.      $f = f + 1$
```

In cases where no threshold is found where $s_m(f) < threshold_{a,b}$, then $A(f) = A_b(f)$. In the pseudo-code above, $s_m(f)$ is the speed at frame f of the motion to be synthesized, $\vec{p}$ the position of the soccer player taken from the positional tracking data, $\Delta t(f)$ the time duration of a frame in the positional tracking data, $A(f)$ the annotation at frame f and with $s_B > s_A$.

Since we want to use only the specified locomotion animations to prevent different speed definitions for the same annotation (such as running at different average speeds), all other mocap must not be annotated with locomotion annotations. It would be interesting in a future study to research if we would still obtain good, or even improved results by removing this requirement.

3.1.2. Mirroring

By mirroring the existing motions on skeleton's sagittal plane, we are able to duplicate the available mocap dataset without the need for extra expensive motion capture sessions. In the case of Beyond Sports, since the mocap vendor provided motions with only one dominant side, such as kicks with only the right foot for example, we can mirror them to have animations for players with any dominant side.

When analyzing the skeleton provided by the vendor (Figure 5), the following was noticed:

A) For all joints, the X axis in the local coordinate system, $\hat{\boldsymbol{x}}_{\boldsymbol{l}}$ pictured as a red unit vector in Figure 5, points to the child joint:

$$\hat{x}_l = normalized(p_{i+1} - p_i)$$

Therefore, the **mirrored world rotation** should position the local (mirrored) X axis unit vector, $\hat{\boldsymbol{x}}_{\boldsymbol{l}}'$, such that its origin is located at **current world position**, p_i, and directed towards the **mirrored world position** of the child joint, $p_{i+1}{}'$:

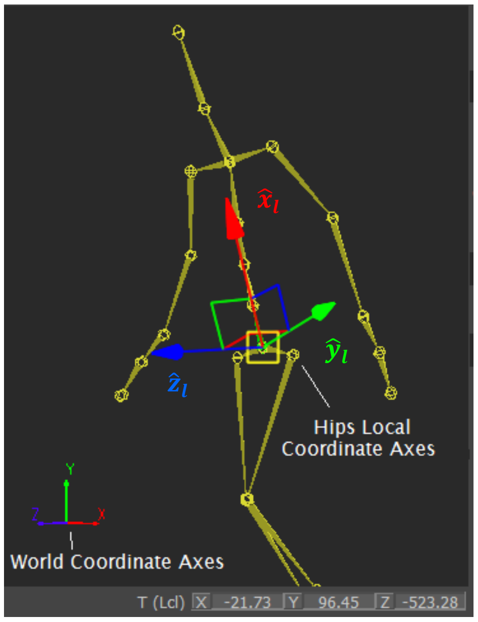

$\hat{x}_l$
$\hat{y}_l$
$\hat{z}_l$
Hips Local Coordinate Axes
Y
Z
X
World Coordinate Axes
T (Lcl) X -21.73 Y 96.45 Z -523.28

Figure 5: Skeleton with the root's (Hips) local coordinate axes.

$$\hat{x}_l' = normalized(p_{i+1}' - p_i)$$

B) The Y axis in the local coordinate system, $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ pictured as a green unit vector in Figure 5**,** is in the opposite direction of the facing direction of the root joint, $\vec{d}_r$: $\hat{y}_l = -\vec{d}_r$. For all the other joints, it is the Z axis, $\hat{\boldsymbol{z}}_{\boldsymbol{l}}$ and pictured in blue, which is either in the opposite or in the same direction as the joint's facing direction: $\hat{z}_l = -\vec{d}_i$ or $\hat{z}_l = \vec{d}_i$ depending on the joint $i \neq r$.

Therefore, the **mirrored world rotation** should position the local (mirrored) Y (or Z) axis unit vector, $\hat{\boldsymbol{y}}_{\boldsymbol{l}}'$ (or $\hat{\boldsymbol{z}}_{\boldsymbol{l}}'$), such that its origin is located at the same position but directed towards the mirrored direction of the current Y (or Z) axis in the ZY plane (negating X coordinate):

$$\hat{y}_l' = T_M \hat{y}_l \quad \text{or} \quad \hat{z}_l' = T_M \hat{z}_l$$

Where T_M is a 3D transformation that performs mirroring on the plane M defined as $\hat{x}_w \cdot (p - p_r) = 0$. $\hat{x}_w$ is the X axis in the world coordinate system and p_r the root's position.

Using the motion specification defined in Equation (4) we can further define WR and LR as the set of all world and local rotations, respectively, and WP as the set of all world positions. The following pseudo-code performs the mirroring of the original motion:

```
1.    FOR EACH frame $f$ IN total number of frames of motion $m$
2.      GET $\vec{d}_i(f)$ FOR EACH joint $i$
3.      GET $WP_m(f) = \{p_0^w(f), p_1^w(f), \dots, p_n^w(f)\}$
4.      COMPUTE $WP_{m'}(f) = T_M WP_m(f) = \{p_0^{w'}(f), p_1^{w'}(f), \dots, p_n^{w'}(f)\}$
5.      COMPUTE $WR_{m'}(f) = \{q_0^{w'}(f), q_1^{w'}(f), \dots, q_n^{w'}(f)\}$
6.        WHERE $q_i^{w'}(f)$ is such that FOR EACH joint $i$
7.          $\hat{x}_l'(f) = q_i^{w'}{}_{\theta_y,\theta_z}(f)\hat{x} = normalized(p_{i+1}'(f) - p_i(f))$
8.          $\hat{y}_l'(f) = q_i^{w'}{}_{\theta_x,\theta_y,\theta_z}(f)\hat{y} = T_M \hat{y}_l(f)$ where $\hat{y}_l(f) = -\vec{d}_i(f)$ for $i = root$
9.          $\hat{z}_l'(f) = q_i^{w'}{}_{\theta_x,\theta_y,\theta_z}(f)\hat{z} = T_M \hat{z}_l(f)$ where $\hat{z}_l(f) = -\vec{d}_i(f)$ or $\hat{z}_l(f) = \vec{d}_i(f)$ for $i \neq root$.
10.     COMPUTE $LR_{m'}(f) = \{q_0^{l'}(f), q_1^{l'}(f), \dots, q_n^{l'}(f)\}$
11.       WHERE $q_i^{l'}(f) = \left(q_{i-1}^{w}{}'(f)\right)^{-1} q_i^{w'}(f)$ where $q_i^{l'}(f) = q_i^{w'}(f)$ for $i = root$
12.     COMPUTE $m(f) = \left(p_r^{w'}(f), q_0^{l'}(f), q_1^{l'}(f) \dots, q_n^{l'}(f)\right)$
```

Where m is the original motion, q_i^l the local rotation of joint i, p_i^w the world position of joint i, n the total number of joints in skeleton and the superscript $'$ defines it is related to the mirrored motion. In step 7, $\mathrm{q_i^{w'}}_{\theta_y,\theta_z}(\mathrm{f})$ is used to denote that only two rotations are needed to get $\hat{x}_l'(f)$ and in steps 8 and 9 the other rotation is computed to get either $\hat{y}_l'(f)$ or $\hat{z}_l'(f)$.

3.2. Synthesis

The synthesized motion will be a sequence of concatenated frames that match user provided annotations **(annotation constraints)** and are continuous **(continuity constraint)** while ensuring user selected frames occur at desired moments **(frame constraints)** and at the desired spatial position **(position constraints)**.

Taking the same definition as in [Arikan et al. 2003], if frames in the database are represented as $f_1, \dots, f_T$, the synthesized motion will be $f_{\sigma_i}, \dots, f_{\sigma_n}$ where $\sigma_i \in [1 \dots T]$ is frame number i. T is the total number of frames in the database and n the number of frames of the motion to synthesize. The function to minimize which will give the desired motion is:

$$\min_{\sigma_i \dots \sigma_n} \left[\alpha \sum_{i=1}^{n} D\left(i, A\left(f_{\sigma_i}\right)\right) + (1-\alpha) \sum_{i=1}^{n-1} C(f_{\sigma_i}, f_{\sigma_{i+1}}) \right] \tag{13}$$

D and C evaluate how well the motion to synthesize matches the desired annotations and how far each frame is from each other, respectively. The α parameter is used to give more weight to motions that are more continuous ($\alpha \to 0$) or motions that match the annotation better ($\alpha \to 1$). Functions D and C introduce annotation and continuity constraints to the synthesis and are delved into in Chapters 3.2.1 and 3.2.2. Due to the way these two functions are defined in our approach, a value of 0.2 was experimentally reached and set for α. This value gives a good result because the motion to synthesize will be mostly composed of locomotion with punctual sections of non-locomotion. These non-locomotion sections are defined as frame constraints and, therefore, will always occur in the output motion, decreasing the importance of a good annotation match for such cases.

The function presented in equation (13) only takes local continuity and annotation matching into account. In order to obtain a global minimum, it can be formulated as a dynamic programming problem, as presented in [Arikan et al. 2003]:

$$J\left(i, f_j\right) = \min_{\sigma} \left[\alpha D\left(i, A\left(f_j\right)\right) + (1-\alpha) C\left(f_\sigma, f_j\right) + J(i-1, f_\sigma) \right] \tag{14}$$

$$J\left(1, f_j\right) = D(1, A\left(f_j\right)) \tag{15}$$

Position constraints are not included in the equations above and will be addressed at a later stage in the Beyond Mocap framework (Chapter 3.2.4). A possible improvement on our approach would integrate position constraint evaluation with annotation matching and continuity evaluation, which could potentially make it more memory and time efficient.

If the synthesis was performed by single frame analysis, as described by the equations (14) and (15), it would be too computationally costly in regards to both memory and time cost: $O(n \times T^2)$. As described in the reference paper, a much faster approach that provides an approximation to an optimum solution is using **frame blocks** taken from the mocap database, and further refining the synthesized motion by using progressively shorter frame blocks.

The initial synthesized motion created out of big blocks of frames captures the most significant characteristics desired by the user. By using smaller blocks, we improve the smoothness and annotation matching while making the algorithm run faster due to the smaller search space provided by the rough initial motion. The dynamic programming (DP) problem will be run with **frame blocks** instead because for most of the time, a soccer player does not perform a motion for just a couple of frames, be it either running, walking, kicking the ball or falling over a tackle. It usually lasts for a couple of seconds. Therefore, since the mocap was performed at 30 frames per second, and the shortest mocap in the database is about 36 frames long, we start by performing DP on blocks of 32 frames. This is further refined in following runs to 16 and 8 frame blocks.

With all motions in the database split in blocks of frames, many of these will be similar to each other. As an example, with a single running motion it is possible to get as many similar frame blocks as there are running cycles. To prevent redundant computations, and following the technique provided in the reference paper, clustering is performed on all similar blocks of the same length into representative frame blocks represented by a cluster. This will significantly speed up the synthesis since, as noted by [Arikan et al. 2003], the computational cost of DP is quadratic in the number of available frame blocks.

For a motion n frames long and with a starting frame block length of 32, there will be $n/32$ time slots. Each slot will need to be filled with a 32 frame block taken from the database and in accordance to equations (14) and (15). Due to the computational cost of DP being $O(n \times T^2)$, we aim to have the minimal possible number of clusters so that the solution is close to the global optimum that we would have obtained if we had used all frame blocks. The reference paper proposes 100 clusters to use in the clustering of all frame blocks in the database, and the same amount is also used in Beyond Mocap. Since it was empirically shown as providing enough variety between each cluster while still enabling the DP to be fast enough. However, contrary to the reference paper, we do not limit the number of frame blocks to 100 for each cluster. Since our mocap database is small, we take all blocks that are contained in the respective cluster, meaning that once we reach slots of 8 frames long, which is the stopping point of the algorithm, we have reached the best result for the optimization function without needing to restart the search.

In order to perform clustering, the frame blocks need to have a corresponding coordinate system so k-means clustering can be applied:

- The coordinate system of a frame is the frame's **feature vector**. The feature vector $F(f)$ for frame f contains the joint position, velocity and acceleration for all joints relative to the skeleton root for that frame, meaning the root only contains speed and velocity information with the position being the world center. This ensures each feature vector contains temporal information about each joint. Therefore, the first two frames of every motion are discarded since the first frame does not contain velocity and acceleration and the second frame does not have acceleration information.

- Given the above definition, the coordinate system of a frame block is the corresponding **augmented feature vector**. An augmented feature vector is simply all feature vectors concatenated into a big vector. If the frame block is 32 frames long, then the augmented feature vector is the concatenation of 32 feature vectors.

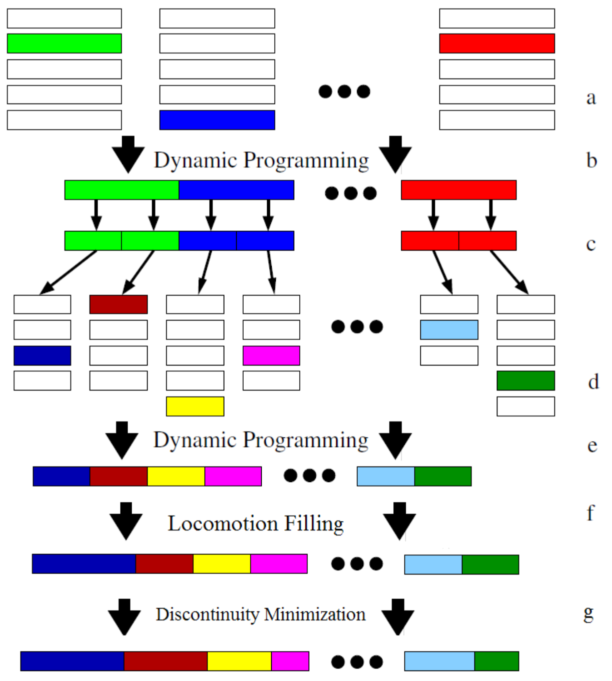


Figure 6: Representation of Beyond Mocap, adapted from [Arikan et al. 2003], depicting the main synthesis phase.

As presented in [Arikan et al. 2003], once all augmented feature vectors are computed for all blocks, k-means clustering is performed to find 100 clusters of frame blocks of the desired length (Figure 6-a, where each cluster is represented by a column and each rectangle is a frame block). These will be used to provide candidate frame blocks for each slot of the motion to be synthesized. After DP is performed on 32 frame blocks, we will have a resulting motion capturing the desired general characteristics (Figure 6-b). In the next step, this sequence of 32 frame blocks is refined by doing another DP run on 16 frame blocks. By doing so, continuity and annotation matching are improved without changing the main structure of the synthesized motion. The candidate blocks for each slot will also be taken from a representative cluster but the cluster selection is performed differently: the same amount of clusters (100) will be used for k-means, and the cluster where the 16 frame block of the parent solution belongs to will be used to obtain the candidate blocks for this second dynamic programming run (Figure 6-c,d,e, where the initial rough motion is broken into 16 frame blocks, which are then used to find what cluster to use for obtaining the candidate frame blocks for each slot, and another run of DP is performed). The search stops at 8 frame blocks since further refinement would not provide significant gains in terms of continuity or annotation matching while making the algorithm much slower.

A note regarding the first time DP is performed: at the start of the algorithm, we first need to select which clusters provide better continuity and better annotation matching. Therefore, DP is actually performed twice: once to select the cluster for each time slot, and a second with the candidate frame blocks from each cluster. This is in contrast to the following iterations where we do not need to analyze clusters in this way since we take the cluster where the 16 (or 8) frame block of the parent solution belongs to. The optimization formula used is the same as if we were working with frame blocks, but instead of getting a sequence of frame blocks, the result will be a sequence of clusters.

After obtaining a sequence of 8 frame blocks, two more steps are taken before the motion is constructed: locomotion filling to respect position constraints (Figure 6-f) and discontinuity minimization (Figure 6-g). These will be addressed in chapters 3.2.4 and 3.2.5, respectively. In the following chapters we will also delve deeper into how **continuity constraints** (3.2.1)**, annotation constraints** (3.2.2) and **frame constraints** (3.2.3) are enforced, and end with how the concatenation of the resultant sequence of frame blocks is performed (3.2.6).

3.2.1. Continuity Constraint

In order to ensure a certain degree of continuity and smoothness in the synthesized motion, a function specifying the level of continuity when placing frame f_j after frame f_i will need to be defined. This is the C function in equations (14) and (15). It is calculated as the frame distance between f_j and f_{i+1} (frame following f_i). To perform such calculations, feature vectors are computed for all frames in the mocap database. Frame distance, $C(f_i, f_j)$, is therefore calculated as:

$$C(f_i, f_j) = \left\| F(f_{i+1}) - F(f_j) \right\| \tag{16}$$

However, computing these frame distances with such a large number of features (279 in the case of the skeleton provided by the vendor for Beyond Sports: 31 joints × 9) would be time prohibitive. The number of features is therefore reduced using Principal Component Analysis as presented in [Arikan et al. 2003]. In our approach, we reduce the number of features to a number which satisfies the cumulative energy of the principal components being more than 98%. For the mocap database used in this project, 27 features were enough to describe all frames, which implies at least a tenfold reduction in the frame distance calculation time as well as a high reduction in memory usage. The value of 98% was empirically chosen due to the fact it still gives results very similar to a calculation with the original number of features and is quick enough to be practical.

Since the algorithm will be working with **frame blocks ($\boldsymbol{fb}$)** instead of individual frames, frame distance will also be calculated as frame block distance instead and will use **augmented feature vectors ($\boldsymbol{AF}$)** for its calculation:

$$C(fb_i, fb_j) = \left\| AF_{end}(fb_i) - AF_{start}(fb_j) \right\| \tag{17}$$

AF_{start} and AF_{end} are the augmented feature vectors representing the initial and the end motion of the respective frame block. In our approach, AF_{start} is obtained from the feature vectors of a moving window of predefined size with its center on the *first frame* of the respective frame block. AF_{end} is obtained from the feature vectors of a moving window of the same size with its center on the *last frame* of the respective frame block (Figure 7-a). Beyond Mocap uses $slot\_length/2$ for the window size, meaning that for the initial 32 frame blocks, both AF_{start} and AF_{end} are computed from the feature vectors of 16 frames, starting 8 frames before and ending 8 frames after the first frame (or last frame).

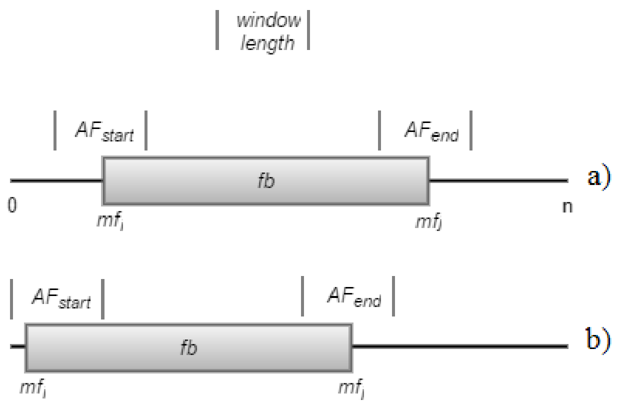


Figure 7: Computing AF's for a frame block (a) and computing AF's for a frame block where the window of AF$_{start}$ cannot be centered on the first frame (b). The horizontal line represents the full length of the original motion from where *fb* is extracted.

The augmented feature vectors are defined as moving windows because of cases where the first frame (or last frame) are positioned in the original motion in such a way that it is not possible to obtain the amount of preceding (or following) frames necessary for the AF computation. If the first frame of a frame block, mf_i, is at the beginning of the original motion where the frame block was taken from, $mf_i < window\_size/2$, the window center of AF_{start} will need to be translated. It will then start at the first usable frame of the original motion and end $window\_size$ frames after (Figure 7-b). The equivalent approach is also taken for AF_{end} but with regards to the last frame.

When working with clusters instead of frame blocks, equation (17) is still valid but the calculation of AF_{start} and AF_{end} is much simpler. They are, respectively, the first $window\_size$ frames and the last $window\_size$ frames of the representative frame block of the respective cluster.

3.2.2. Annotation Constraint

The other component of the optimization problem expressed in equations (14) and (15) is the level of similarity in annotations between the motion in the mocap database and the desired annotation for the motion to be synthesized. This can be represented by a cost function to be minimized for a best annotation match: $D(i, A(f))$, where lower values indicate a better annotation match.

Following [Arikan et al. 2003], with an annotation vocabulary of size l, an annotation vector for frame f is defined as $A(f)[1 \dots l] \in \{1, -1\}$ where the k'th element, with $1 \leq k \leq l$, has the value "**1**" if frame f has the k'th annotation and "**-1**" if it doesn't. For example, in a vocabulary consisting only of **walking, running** and **kick**, and where the first item in the annotation vector represents the presence of walking, the second item the presence of running and the final element the presence of kick, we could have annotation vectors as follows:

- For a walking only frame, f_A: $A(f_A) = [\ \ 1 \quad -1 \quad -1]$
- For a running kick frame, f_B: $A(f_B) = [-1 \quad 1 \quad 1]$
- For a running only frame, f_B: $A(f_B) = [-1 \quad 1 \quad -1]$

As such, $A(f_B)[1] = 1$ for a 0-based vector:

When annotating the player timeline, which are **desired annotations**, the same rule applies. If the annotation is desired, the corresponding element in the annotation vector will have the value 1, and if it should not be present, the element will have the value -1.

In the same way **augmented feature vectors** are all feature vectors of a block of frames concatenated into one big vector, an **augmented annotation vector (AA)** is similarly defined as all annotation vectors of a block of frames concatenated into one big vector.

The annotation similarity cost function, adapted from [Arikan et al. 2003], is then:

$$D\big(i, AA_m(fb)\big) = -\frac{\sum_{j=1}^{j=l \times slot\_length} AA_d(fb_i)[j] \times AA_m(fb)[j]}{r} \tag{18}$$

for all $AA_d(fb_i)[k]$ and $AA_m(fb)[k]$ that are present in the corresponding frame (having value 1).

$AA_m(fb)$ represents the augmented annotation vector for frame block fb in the mocap database; $AA_d(fb_i)$ represents the augmented annotation vector for frame block fb_i of the motion to be synthesized (desired annotations); $slot\_length$ is the number of frames present in a frame block (and, therefore, 32 at the start of the algorithm) and, finally, r is the total number of annotations that are present in either frame. Taking the annotation examples shown above, if we were comparing frames A and C, the value of r would be 2.

In case D is being computed between AA_d and the AA of a cluster, AA_c, we calculate the similarity between AA_d and each AA of all frame blocks contained in the cluster, averaged by the total number of frame blocks:

$$D\big(i, AA_c(c)\big) = \frac{\sum_{j=1}^{b} D\left(i, AA_m(fb_j)\right)}{b} \tag{19}$$

Where $AA_c(c)$ represents the augmented annotation vector for cluster c and b the total number of frame blocks present in cluster c.

There are two differences in our annotation similarity cost function when comparing to the equivalent function defined in [Arikan et al. 2003]: there is no "*don't care*" annotation (value of 0), and only annotations that are present in either frame are compared. Annotations marked as "*don't care*" are not supported because we are concerned with replicating a real life moment, making it undesirable to have variations that are not exactly what is intended in the synthesized motion: they need to be exact. If the desired annotation states the player is kicking the ball with the top of the left foot at a lower height, then a motion annotated with a kick with the top of the left foot at a medium height instead is not acceptable as an annotation match. And only annotations that are present in either frame are compared, due to the vocabulary being quite large. This can lead to a running motion matching well with a jogging motion when the whole vector is used for comparison, only for the fact that all the other annotations are defined as not present, and therefore as matching with each other.

Since the entire player timeline will be annotated, contrary to the scenario presented by [Arikan et al. 2003], a potentially better approach mentioned by the authors would have been to simply calculate the squared difference between annotation vectors as a representation of annotation similarity.

There is one exception to how the annotation of a frame should be done as defined at the start of this sub-chapter. The motion to synthesize will contain several *setup* or *entry* phases, which are the **transition motions** just before a **key moment**, as well as *post key moment* or *exit* phases. In a match situation where the player executes a free kick, the motions corresponding to the player taking a few steps back before running forward to kick the ball (entry phase) and to the player regaining balance after the kick (exit phase) would be the **transition motions** around the **key moment** of the player kicking the ball. When the player is in a transition motion, the annotation vector should have all elements with value '0' regardless of the characteristics of such motion, forcing the synthesis algorithm to only consider continuity constraints when choosing which frame block to choose for the corresponding slot. This is intended to ensure there is a smooth and continuous transition between the motion before and after the kick, and that it takes as long as necessary, preventing half-entry or half-exit phases which would not be natural looking.

3.2.3. Frame Constraint

Frame constraints in Beyond Mocap are used for **key moments** and are defined by a specific annotation format presented in Chapter 4.2.2. With regards to the synthesis algorithm, frame constraints restrict the search space in the dynamic programming problem for the slot in consideration, to the individual frame block that contains such a frame. We can then go directly to the cluster and select the relevant frame block. Frame constraints make the motion synthesis much faster because the computational cost of DP is quadratic in the number of frame block candidates, and in these cases, there is only one candidate: the frame block containing the desired frame.

Since the algorithm works with frame blocks and not individual frames, there could be multiple frame blocks which contain the desired frame. In order to deal with this, a multiplier, FC_m, is applied to the result of $D\big(i, AA_m(fb)\big)$ (equation (18)) when comparing blocks where the desired augmented annotation vector contains the frame constraint. FC_m is a function of the distance to the desired frame number location and value of the annotation similarity between the two individual frames.

$$FC_m = \beta \left| a_{AA_m} - b_{AA_d} \right| \times D(f_a, f_b) \tag{20}$$

where $\beta = slot\_length.$ By using FC_m we ensure the chosen frame block from the mocap database has the frame constraint occurring at exactly the same time as desired.

3.2.4. Position Constraint

At this point, the DP formulated by equations (14) and (15) has output a sequence of 8 frame blocks:

$$S = \{fb_0, fb_1, \dots, fb_B\} \tag{21}$$

Where fb_i is frame block i and $B = n/slot\_length$ is the total number of blocks in the sequence.

The solution proposed by [Arikan et al. 2003] describes how to respect position constraints, but only for discrete cases and is mostly focused on having a motion start and end at specific locations but not following a specific path. Due to this reason, a new approach is presented in this sub-chapter that ensures the synthesized motion follows the path described by the positional tracking data.

Locomotion is cyclical and therefore it should be possible to add or remove cycles to a locomotion animation if we desire to add to, or remove from, the virtual character's distance travelled. Another observation is that foot-skating is introduced when the skeleton's root is moved in space at a different speed than the original. But if the motion is only time warped by speeding up or slowing down its playback, no foot-skating is introduced.

Taking these observations into account and if we ensure the distance travelled by the virtual character is the same as in the positional tracking data, we can then match the speed at which it is played so that the distance travelled at each frame is also the same, with no foot-skating added. With this approach we enforce the position constraints. In this chapter we will cover the algorithm that synthesizes locomotion sequences in order to have the character travel a desired distance. In Chapters 3.3.2 and 3.3.3 we will cover the second part of enforcing position constraints: time warping specific sections of the synthesized motion and setting movement direction.

The algorithm described here works by adding or removing locomotion cycles per locomotion sequence, to ensure the distance being travelled in the synthesized motion matches the distance travelled by the soccer player obtained from the positional tracking data. In order to maintain continuity, the entry and exit frame blocks, fb_{entry} and fb_{exit}, respectively, are kept the same and the frame blocks in between are replaced. Let S_L denote a sequence of frame blocks from a locomotion sequence, dt_{fb_i} the distance travelled by the soccer player in the frames contained in fb_i and PTD refers to the positional tracking data:

```
1.    INITIALIZE $S' = S$
2.    GET ALL $S_L = \{fb_{entry\_block}, fb_i, \ldots, fb_{exit\_block}\}$ where $entry < i < exit$ AND $AA(fb_i) = locomotion\ only$
3.    FOR EACH locomotion sequence $S_{L_i}$ IN all locomotion sequences found in step 1
4.      COMPUTE $PTD_{L_i} = \{f_j, \ldots, f_{j+n}\}$ where $A(f_j) = locomotion\ only$
5.      COMPUTE $dt_{PTD_{L_i}} = \sum_{k=o}^{j+n} dt_{f_k}$ where $f_k$ is taken from PTD
6.      COMPUTE $dt_{S_{L_i}} = \sum_{k=entry\_block}^{exit\_block} dt_{fb_k}$
7.      COMPUTE $dt_{diff} = \left| dt_{PTD_{L_i}} - \left( dt_{S_{L_i}} + \sum_{k=0}^{entry\_block_{S_{L_i}}-1} dt_{fb'_k} \right) \right|$
8.      IF $dt_{diff} > \delta$ THEN
9.        COMPUTE $dt_{desired} = dt_{S_{L_i}} + dt_{diff}$
10.       COMPUTE $S'_{L_i}$ such as $dt_{S'_{L_i}} = dt_{desired}$
11.       REPLACE $S_{L_i}$ with $S'_{L_i}$
```

Where δ is an acceptable travelled distance mismatch and fb' a frame block from S'_L. As shown in the above pseudo-code, since this approach is only possible with locomotion animation, the desired distance the character should travel has to take into account the distance mismatch in non-locomotion animation. Due to the usage of a small mocap database, the non-locomotion animations in the majority of cases will not traverse the same distance as provided by the real match data. As an example, we could be using a kick animation with a dt of 1 meters when, according to the positional tracking data, the player travelled 1.5 meters, losing 0.5 meters in the synthesized sequence. As such, $dt_{desired}$ is calculated as the sum of the distance travelled by the current synthesized locomotion sequence ($dt_{S_{L_i}}$) plus the distance we have to add, or remove, (dt_{diff}). dt_{diff} is calculated as the difference between the distance travelled by the player provided by the PTD until the end of the current locomotion sequence ($dt_{PTD_{L_i}}$) and the sum of $dt_{S_{L_i}}$ with the distance travelled in the replacement synthesized locomotion sequences until the beginning of the current sequence ($\sum_{k=0}^{i-1} dt_{S'_{L_k}}$). If instead we took $dt_{desired} = dt_{PTD_{L_i}}$ we would only be taking into account the distance lost (or gained) during locomotion only, ignoring the distance mismatch during non-locomotion animations.

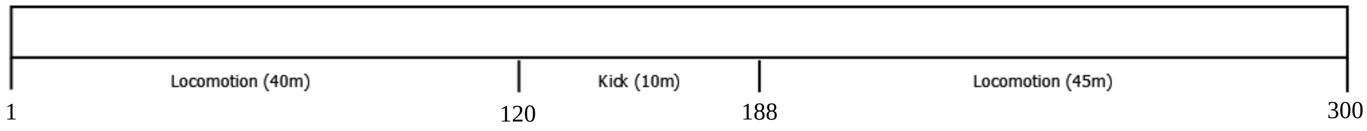


Figure 8: Soccer player motion to be synthesized split into locomotion and non-locomotion sections with their respective distance travelled and boundary frame numbers.

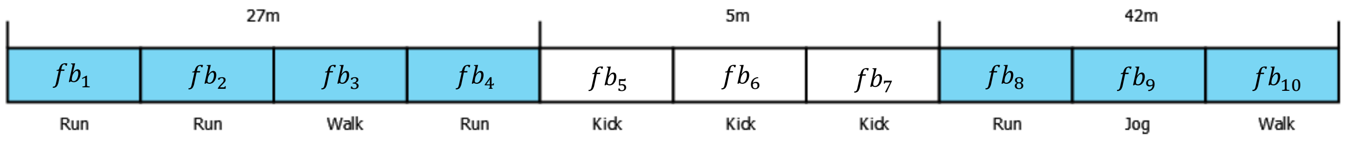


Figure 9: Synthesized sequence at the start of position constraint enforcement. We list the corresponding annotation below each frame block and the distance travelled for each locomotion and non-locomotion sections.

In order to illustrate this approach, let us take an example of a motion where the soccer player runs and walks for about 40 meters, then kicks the ball while in motion traversing 10 meters throughout, and finally runs, jogs and walks totaling 45 more meters (Figure 8). With a synthesized sequence S shown in Figure 9, the steps shown below are performed by the algorithm described in the previous pseudo-code. For example purposes only, we arbitrarily set δ to 1m:

Iteration 1:

$S_{L_1} = \{fb_1, fb_2, fb_3, fb_4\}$

(4) $PTD_{L_1} = \{f_0, \dots, f_{120}\}$

(5) $dt_{PTD_{L_1}} = \sum_{k=o}^{120} dt_{f_k} = 40m$

(6) $dt_{S_{L_1}} = \sum_{k=1}^{4} dt_{fb_k} = 27m$

(7) $dt_{diff_1} = \left| dt_{PTD_{L_1}} - \left(dt_{S_{L_1}} + \sum_{k=0}^{1-1} dt_{fb'_k} \right) \right| = |40 - (27 + 0)| = 13m$

(8) Since $dt_{diff_1} > 1m$:

(9) $dt_{desired} = dt_{S_{L_1}} + dt_{diff_1} = 27 + 13 = 40m$

(10 & 11) S'_{L_1} is synthesized, consisting of 6 frame blocks with a total dt of 39m, and replaces S_{L_1} in the final sequence:

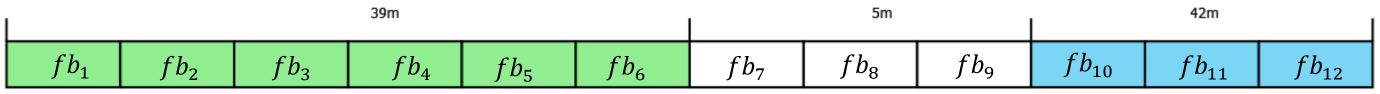


Figure 10: Final synthesized sequence after Iteration 1 where S_{L_1} has been replaced by S'_{L_1}.

Iteration 2:

$S_{L_2} = \{fb_{10}, fb_{11}, fb_{12}\}$

(4) $PTD_{L_2} = \{f_{188}, \dots, f_{300}\}$

(5) $dt_{PTD_{L_2}} = \sum_{k=o}^{300} dt_{f_k} = 95m$

(6) $dt_{S_{L_2}} = \sum_{k=10}^{12} dt_{fb_k} = 42m$

(7) $dt_{diff_2} = \left| dt_{PTD_{L_2}} - \left(dt_{S_{L_2}} + \sum_{k=0}^{10-1} dt_{fb'_k} \right) \right| = |95 - (42 + 44)| = 9m$

(8) Since $dt_{diff_2} > 1m$:

(9) $dt_{desired} = dt_{S_{L_2}} + dt_{diff_2} = 42 + 9 = 51m$

(10 & 11) S'_{L_2} is synthesized, consisting of 6 frame blocks with a total dt of 51, and replaces S_{L_2} in the final sequence:

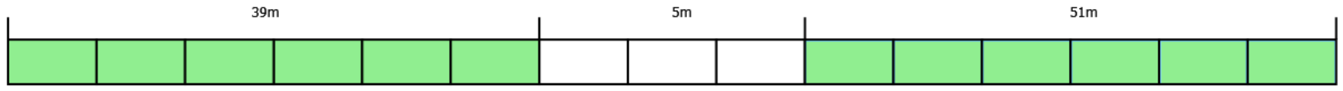


Figure 11: Final synthesized where the distance travelled by the virtual character is the same as in the positional tracking data. Locomotion frame blocks are represented in green.

Following the above pseudo-code, the objective is the minimization of $dt_{desired} - dt_{S'_{L_i}}$. Thus, we can formulate it as a DP problem in a similar way as previously done in equations (14) and (15):

$$J_L\,(i, fb_l) = \min_{\sigma}\left[D\big(i, AA_m(fb_l)\big) + C(fb_\sigma, fb_l) + J(i-1, fb_\sigma)\right] \tag{22}$$

$$J_L\,(2, fb_l) = D\big(i, AA_m(fb_l)\big) + C\big(fb_{entry\_block}, fb_l\big) \tag{23}$$

Where D and C evaluate how well the motion to synthesize matches the desired annotations and how far each frame is from each other, respectively, and $AA_m(fb_l)$ the augmented annotation vector for frame block fb_l. In the equations above, the pool of motions the frame blocks fb_l andfb_σ are extracted from consist of only locomotion mocap. S'_L, from the previous pseudo-code, is therefore computed as:

```
1.   FOR EACH locomotion frame block $fb_l$
2.     WHILE $\Delta dt < dt_{desired}$ with $\Delta dt = \left(\sum_{j=1}^{i} dt_{CS'_L(j)}\right) + dt_{fb_{exit\_block}}$
3.       $CS'_L = \{fb_0, fb_i, \dots, fb_B\}$ found through $J_L\,(i, fb_l)$ where $fb_B = fb_{exit\_block}$ and $fb_0 = fb_{entry\_block}$
4.   FOR EACH candidate sequence $CS'_L$ obtained from previous cycle
5.     $S'_L = \min_{CS'_L}\left[\left|dt_{desired} - \sum_{j=1}^{n} dt_{CS'_L(j)}\right|\right] = \min_{CS'_L}\left[\left|dt_{desired} - dt_{CS'_L}\right|\right]$
```

From equations (22) and (23), $D\big(i, AA_m(fb_l)\big)$ will need to be computed. However, the positional tracking data cannot be simply split into the same number of blocks of frames for annotation matching calculation. $AA_d(fb_i)$ is then the **desired augmented annotation vector** between frames in the positional tracking data corresponding to the total distance travelled of $\sum_{j=0}^{i-1} dt_{fb_j}$ and $\sum_{j=0}^{i} dt_{fb_j}$.

As mentioned at the start of this sub-chapter, this is the first part of enforcing position constraints. The obtained final sequence S'_L with the correct distance travelled does not have orientation information. This is introduced at a later stage and described in Chapter 3.3.3.

An interesting future research question is whether the synthesis would enforce all constraints in a shorter amount of time by integrating the technique presented in this sub-chapter in the initial motion synthesis algorithm.

3.2.5. Discontinuity Minimization

Since the synthesized motion is constructed out of a sequence of frame blocks, there will always be a certain degree of discontinuity at the boundaries between each pair of blocks coming from different motions. Therefore, as proposed by [Arikan et al. 2003], a local minimization step is introduced to decrease the discontinuity between exit and entry frames of the frame blocks sharing a boundary, while at the same time ensuring the overall frame count remains the same.

Taking the definition of sequence of frame blocks given by equation (21), and with frame block length of $slot\_length$, at each boundary point, between $S(i) = fb_i$ and $S(i+1) = fb_{i+1}$, the motion switches from frame $f_{exit} = fb_i(slot\_length)$ taken from mocap m_1 to frame $f_{entry} = fb_{i+1}(0)$ taken from mocap m_2.

The objective at this step is to decrease the distance $C(f_{exit}, f_{entry}) = \|F(f_{exit+1}) - F(f_{entry})\|$. As such, this function is minimized with respect to f_{exit} and f_{entry} using gradient descent as presented in [Arikan et al. 2003], shifting the entry and exit frames forward or backwards. However, although the majority of frame shifts stay within 1-2 frames, as stated in the reference paper, some boundaries produce bigger frame shifts. This could be due to the small size of our mocap database, but ensuring a constant motion length is crucial to our approach since we must match the distance travelled by the character in the synthesized motion to that of the positional tracking data. Even 1 or 2 frame shifts for each boundary, for a large number of boundaries, could end up introducing a distance error of more than one meter as well as a mismatch on when a **key moment** should occur.

In Chapter 3.3.2 we present a simple approach to minimize these key moment mismatch errors. Perhaps by including this step in the optimization problem formulated for motion synthesis we could improve the discontinuity minimization described here, the motion synthesis speed and minimize travelled distance error as well as key moment frame mismatch. Another possible, and interesting, approach that would minimize the distance error introduced by this step include using blending techniques within a window of frames centered around f_{exit} and f_{entry}, in case increasing the available mocap dataset is not feasible.

3.2.6. Constructing Motion

In order to construct the final synthesized motion, we concatenate all frame blocks from the final sequence, S, by taking the frames from the corresponding motions the blocks were extracted from. This chapter describes how each frame is concatenated and how the character's movement direction is computed for motions where the character is not in an upright position.

When taking frame f_{i+1} from motion m_a to concatenate with the preceding frame f_i from motion m_b it can happen the skeleton is not moving in the same direction, especially for motions $a \neq b$. With this in mind, and in order to have a smooth concatenation with no sudden change in direction, it is necessary to align the movement direction of the skeleton at the beginning of fb_{i+1} with the movement direction of the skeleton at end of the preceding frame block, fb_i, before concatenation is performed.

A simple alignment in the horizontal movement direction is not sufficient however. For the most part, the character will be in an upright position but in situations where that is not the case we also need to ensure the direction of the torso remains unchanged between both frames. Otherwise, it could lead to situations where the skeleton would change the position of its torso from one side to the other between frames f_i and f_{i+1}. Our approach starts by first aligning the skeleton's horizontal movement direction followed by an extra check on the spine's position relative to the root. Both of these directions can be easily obtained through the observation mentioned in Chapter 3.1.2 (A and B).

After fb_{i+1} is aligned with fb_i a simple concatenation of all frames referenced by fb_{i+1}, to the current resulting motion that has been constructed until fb_i, can be performed. The starting position of the skeleton at frame f_{i+1} is obtained by simply calculating the displacement of the skeleton's root in the last two frames and applying it to the skeleton's position at f_i:

$$p_r(f_{i+1}) = p_r(f_i) + \Delta p_r(f_i) \tag{24}$$

where $\Delta p_r(f_i) = p_r(f_i) - p_r(f_{i-1})$.

3.3. Post-Processing

After the synthesized motion is built out of the frame blocks selected in the synthesis phase, our approach enters its final post-processing phase. Although the sequence of frame blocks, which the motion is built from, goes through several passes to ensure a certain degree of continuity between each other, including the discontinuity minimization described in Chapter 3.2.5, there will still be some small discontinuities at the boundaries between them. These small discontinuities are hard to prevent from occurring without resorting to blending, since we only ensure C^0 continuity during concatenation, and thus there will always be a small velocity change due to the limited number of available mocap. In the following Chapter, 3.3.1, we discuss how we smooth out these discontinuities by spreading it over several frames.

The other steps in the post-processing phase are concerned with finalizing the enforcement of the position constraints (Chapter 3.3.2), and referenced in Chapter 3.2.4, and with orienting the character so that it moves in the desired direction (Chapter 3.3.3).

3.3.1. Discontinuity Smoothing

The small discontinuities between consecutive frames extracted from different original motions, that are still present in the synthesized motion, are now smoothed as suggested by the reference paper [Arikan et al. 2003], presented in [Arikan and Forsyth 2002, chap.4.2.3], but adapted to our approach.

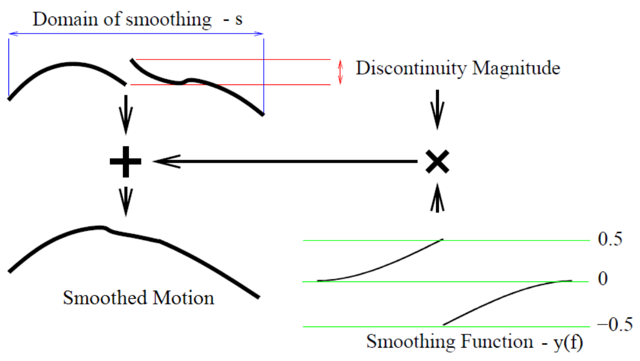


Figure 12: Discontinuity smoothing adapted from Figure 4 in [Arikan and Forsyth 2002]. Top left signal represents the original non-smoothed motion.

Smoothing works by spreading the magnitude of the discontinuity over a window of frames around the boundary where the discontinuity is present (Figure 12). Since the synthesis phase outputs a motion with an optimal continuity, controlled by the α parameter, these discontinuities should be small enough to render any resulting visual artifacts such as foot-skating imperceptible to the user. In practice, the magnitude of the discontinuity is multiplied by the smoothing function presented in [Arikan and Forsyth 2002] and reproduced below:

$$y(f) = \begin{cases} 0 & f < d - s \\ \frac{1}{2}\left(\frac{f-d+s}{s}\right)^2 & d - s \le f < d \\ -\frac{1}{2}\left(\frac{f-d+s}{s}\right)^2 + 2\left(\frac{f-d+s}{s}\right) - 2 & d \le f \le d + s \\ 0 & f > d + s \end{cases} \qquad (25)$$

This equation represents how much of the original displacement should be added to frame f; and added back to the original motion. $d = f_i$ is the frame that follows the discontinuity present between frames f_{i-1} and f_i. Our approach differs from that of [Arikan and Forsyth 2002] in the value given to s: it is the size of the smoothing window around the discontinuity frame and, in our case, set to half of the length of the smallest sequence of frames, $fseq$, without discontinuities. Such a sequence is selected between the frames preceding and following the discontinuity frame:

```
1.     FOR EACH discontinuity point $i$
2.       SET $d_i$ as the discontinuity frame number
3.       GET $d_{i-1}$ as the previous discontinuity frame number
4.       GET $d_{i+1}$ as the following discontinuity frame number
5.       $n_{fseq_{i-1}} = d_i - d_{i-1}$ and $n_{fseq_{i+1}} = d_{i+1} - d_i$
6.       IF $n_{fseq_{i-1}} < n_{fseq_{i+1}}$
7.         $s = n_{fseq_{i-1}}/2$
8.         ELSE    $s = n_{fseq_{i+1}}/2$
```

Where $\mathrm{n_{fseq}}$ is the number of frames in sequence of frames fseq. Otherwise, if the window was set to 30 frames as suggested in the reference paper, the accumulation of successive displacement distribution would cause bigger discontinuities than those we are aiming to smoothen.

If $f_{i-1} = f_j^a$, where f_j^a is frame number j taken from motion m_a, and $f_i = f_k^b$, then the discontinuities are calculated as joint rotation difference and their magnitude is taken between frames f_{j+1}^a and f_k^b with the exception of the root joint. The root rotation is taken from the pair of frames in the discontinuity of the current synthesized motion (between f_{i-1} and f_i) since some orientation discontinuities might still exist after the frame block alignment performed before concatenation. The discontinuity magnitude at frame d for joint i can be given by the following:

$$magnitude(d,i)\begin{cases} q_i\left(f_k^b\right) - q_i\left(f_{j+1}^a\right) & , i \neq r \\ q_r(f_{i-1}) - q_r(f_i) & , i = r \end{cases} \tag{26}$$

3.3.2. Time Warping

Although the virtual character is now travelling the same distance in the synthesized motion as the real soccer player (as provided by the positional tracking data), this distance is not traversed at the same **speed**, causing its position to not match at all times with the desired coordinates, and is only done in a straight line since no **orientation** data has yet been analyzed. Only after speed and orientation are adapted to match the real match data, will the virtual soccer player respect the provided position constraints. To accomplish this, we start by first time warping the locomotion sections followed by orienting the character at each frame. The latter is presented in the following chapter (3.3.3).

During locomotion, small differences of less than a meter between the position of the character in the synthesized motion and the position obtained from the positional tracking data can be ignored as long as this difference is zero for the **key moments** (defined by frame constraints, Chapter 3.2.3). This is done by time warping the synthesized motion, through simple lerp for root position and slerp for joint rotations, in order to have the virtual character move at the same speed as the speed computed from the match data.

However, this is not sufficient for the majority of the synthesized motions due to the distance lost (or gained) when performing non-locomotion animations as mentioned in Chapter 3.2.4. The virtual character will need to travel this distance delta after a non-locomotion animation. If, for example, the character performs a kick, travelling 1 meter in the process, followed by a run of 4 meters, and in the real match data it should have travelled 2 meters during the kick motion, followed by a run of 3 meters, then at the end of the kick there will be a position error of 1 meter. This is the distance, d_{error}, that will need to be caught up by the virtual character when running the following 4 meters so that at the end of the total 5 meters of kick & run, he will be in the same coordinates as those provided in the positional tracking data.

In our approach, this is solved by having the character move at a faster speed when catching up lost distance (or slower speed when it has travelled more than desired) until the coordinates match with the positional tracking data. A logistic function, $d(f)$, is used to spread d_{error} over a user defined window of locomotion frames, l, by providing how much distance it should be added (or removed) to the speed computation per frame, in order to make it look like the virtual character is progressively moving faster to catch up and slowing down to the actual desired speed once it has caught up with the lost (or gained) distance:

$$d(f) = \frac{d_{error}}{1 + e^{v}} \tag{27}$$

Where:

$$v = -\frac{C}{l/2}(f - f_0 - l/2) \tag{28}$$

And where $C = 4.59512$ found empirically and f_0 is the initial frame of the current locomotion section.

This approach is very limited in nature to only motions with sparse frame constraints and special motions with large time intervals between them. Otherwise the speeding up (or slowing down) of the synthesized motion is too obvious to look natural.

In this step we also fix any **key moment** frame mismatch introduced by the discontinuity minimization described in Chapter 3.2.5 by ensuring that, after time warping, the key moments occur at the frame indicated by the corresponding frame constraints.

3.3.3. Movement Orientation

To finalize position constraint enforcement, all that is left to do is to change the virtual character's movement from a straight line to the actual path provided by the positional tracking data. Since the data used throughout this project did not include character movement orientation, $\vec{d}(f)$, orientation was computed at a frame level based on the position delta obtained from the current position in the synthesized motion, *m*, and the next position based on travelled distance from the positional tracking data:

$$\vec{d}(f) = normalized\left(p_m(f) - p_{PTD}\left(f_{dt_{m(f+1)}}\right)\right) \tag{29}$$

Where $f_{dt_{m(f+1)}}$ is the frame in the positional tracking data corresponding to the distance travelled by the character in the synthesized motion until frame $f + 1$. However, if the provided data is not properly filtered this can cause unrealistic rotations performed by the virtual character. Improvement suggestions are presented in Chapter 6.2.

4. Implementation

Beyond Mocap is only part of the full suite of tools necessary to automate the usage of real match data and motion capture in Beyond Sports. A number of tools and scripts need to be integrated to allow for close interaction with each other:

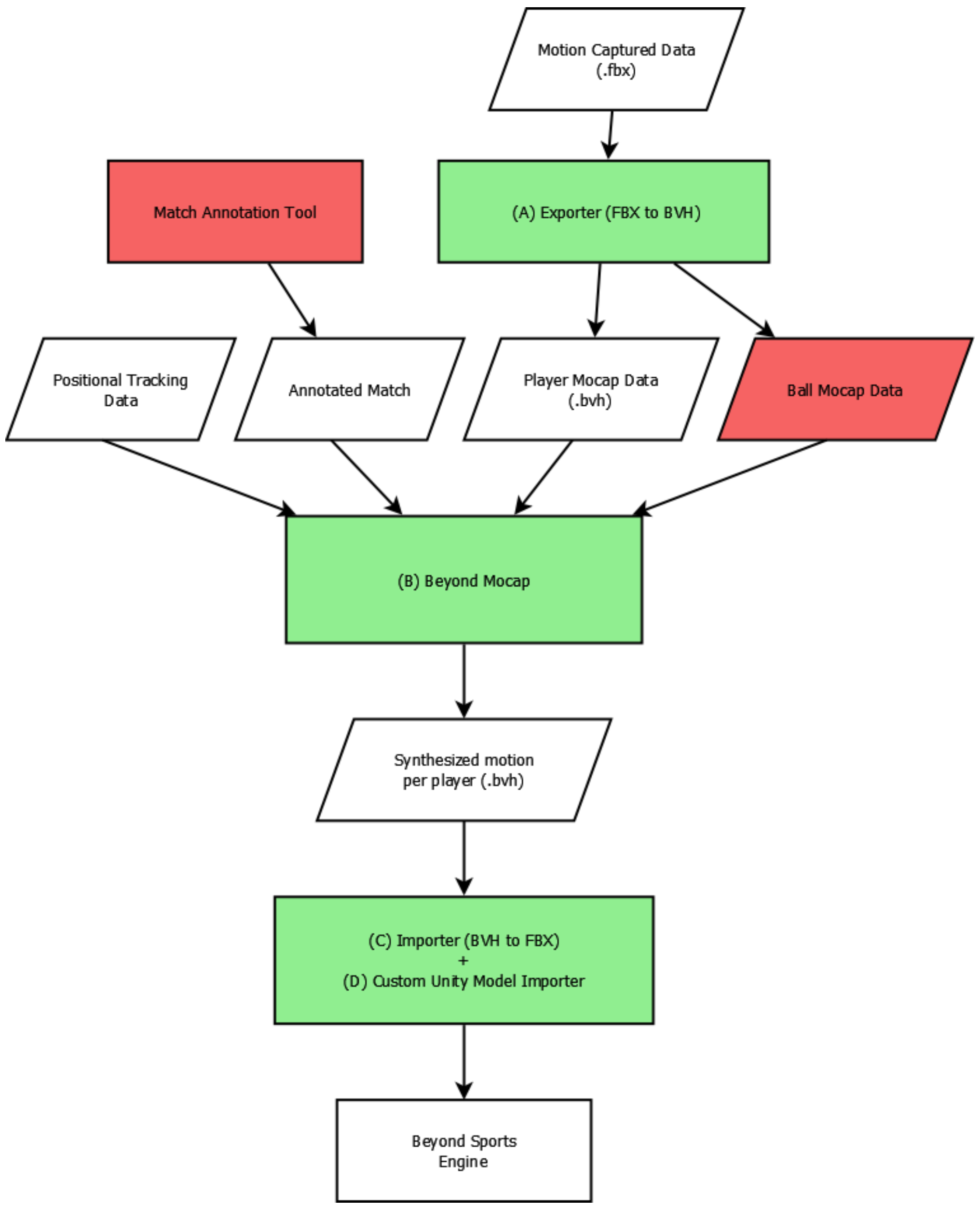

Motion Captured Data (.fbx)
Match Annotation Tool
(A) Exporter (FBX to BVH)
Positional Tracking Data
Annotated Match
Player Mocap Data (.bvh)
Ball Mocap Data
(B) Beyond Mocap
Synthesized motion per player (.bvh)
(C) Importer (BVH to FBX) + (D) Custom Unity Model Importer
Beyond Sports Engine

Figure 13: Flowchart diagram showing how Beyond Mocap is integrated with other external tools and scripts in order to be used with Beyond Sports. Ball Mocap Data extraction and a Match Annotation Tool (red background) are possible improvement avenues to be taken in a future version of the framework.

The motion capture data provided by the vendor is in the Filmbox (FBX) format and, in order to be used by Beyond Mocap, is converted to a more easily editable motion file format (Figure 13-A, Chapter 4.1). Since our motion synthesis approach works by having player timelines annotated, a match annotation tool would simplify and speed up the annotation compared to simply manually typing the corresponding annotations per player (Figure 13-Match Annotation Tool in red background). Finally, in order to load the output synthesized motion files, these have to be converted back to the FBX format due to the game engine used in the development of Beyond Sports not accepting the intermediary motion file format used during synthesis (Figure 13-C and D). The external tools created and used to perform these conversions, and the decisions taken in the implementation direction, are presented and discussed in Chapter 4.1.

In Chapter 4.2 we present the specifications of the intermediary motion file format, of the annotation file format and the positional tracking data used by Beyond Mocap, together with associated issues originating from their use and how these were solved. Finally, since optimization in runtime is a priority in the implementation of Beyond Mocap, we show in Chapter 4.3 how this is further minimized by simple serialization.

4.1. Supporting Tools

Initially, the intention was to have Beyond Mocap and associated scripts implemented using Unity[2], since this is the game engine being used in the creation of Beyond Sports. Therefore, an initial feasibility study was performed, focused on how Unity could assist in the editing and synthesis of character animation as well as its playback in the simulation.

The study centered on the API provided by Unity for its own animation system, Mecanim, and the possibility of allowing us to simply load the FBX motion files and create new motions, or edit existing ones. However, most of the available calls that would allow deeper motion editing are hidden, with only some experimental interfaces available. Yet these do not provide enough flexibility in joint data manipulation for example. Since Unity cannot be used for such operations, and the FBX file format is proprietary and not straightforward to edit, an extra tool which would read from and write to motion files, as well as a different file format, became necessary. Biovision hierarchical data (BVH) was chosen as the intermediary motion file format. In Chapter 4.2.1 we cover it in more detail.

Another limitation encountered in Unity is its lack of functionality to import BVH motion files. Due to this reason, conversions between FBX and BVH have to be performed. Autodesk MotionBuilder[3] was chosen amongst other packages due to its handling of FBX files, easy conversion between these two formats, and ability to run scripts in Python to extend its capabilities and automate the conversion.

[2] https://unity3d.com/. Version used for feasibility study and development of motion file import scripts: Unity v5.3.4f1.

[3] http://www.autodesk.com/products/motionbuilder/overview. Version used: Autodesk MotionBuilder 2016.

Due to the factors just described, it was decided to proceed with a standalone tool (Figure 13-B – Beyond Mocap) that takes as input the match and mocap data and outputs the resulting synthesized animation for each player. Unity is simply used to load and playback the synthesized animation. This has the added benefit that only minimal changes are required on the Beyond Sports engine in order to support the new animations.

To simplify the import of the resulting synthesized animations into Unity, we simply changed the standard import system to automatically extract the animation from the FBX file and retarget it to the model being used, which features the same skeleton hierarchy (Figure 13-D).

Besides character animation and the 3D models for the soccer player and ball, the FBX files provided by the vendor also include ball trajectory whenever the animation has player-ball interaction. It would be interesting to, in a future version of the tool, extract this data and use it to better integrate the synthesized motions in the simulation. Another possible automation improvement would include the creation of a Match Annotation Tool to simplify player timelines annotations and, at a later stage, automatically suggest annotations based on the player motions.

4.2. Input & Output Files

4.2.1. BVH Motion File

As reasoned above, BVH[4] is used as the intermediary motion file format for synthesis and editing (general structure presented in Appendix B). The BVH file manipulation tool used by Beyond Mocap is based on a Python library developed internally at Utrecht University[5]. In its implementation, the skeleton is defined through a hierarchy of BVHNode objects, the root of which is the skeleton's root and the only joint to contain translation data besides rotation; and the end nodes are the end effectors plus top of the head. Each BVHNode has an offset: for the root this represents the starting position in world coordinates and for all the other joints it represents the corresponding bone length.

Regarding the vendor's avatar specification, it has an extra, stationary, root joint and thus the root of the skeleton matches this extra joint, with the hips specified as child. Due to this, before conversion from BVH to FBX, an extra root is added back again following the same name convention, to allow for an easy import and retargeting in Unity.

Finally, the order of the rotation channels in the BVH motion file follow a non-standard order: first Z, followed by X and finally Y rotations. Therefore, the rotation matrix needs to be computed through the concatenation of Y, X and Z rotation matrixes as follows:

[4] http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html.

[5] http://www.uu.nl

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{bmatrix},\ R_y = \begin{bmatrix} \cos(\theta_y) & 0 & -\sin(\theta_y) \\ 0 & 1 & 0 \\ \sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix},\ R_z = \begin{bmatrix} \cos(\theta_z) & \sin(\theta_z) & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (30)$$

Therefore:

$$R_{BVH} = R_y R_x R_z$$

$$R_{BVH} = \begin{bmatrix} \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z) & \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z) & -\cos(\theta_x)\sin(\theta_y) \\ -\cos(\theta_x)\sin(\theta_z) & \cos(\theta_x)\cos(\theta_z) & \sin(\theta_x) \\ \sin(\theta_y)\cos(\theta_z) + \cos(\theta_y)\sin(\theta_x)\sin(\theta_z) & \sin(\theta_y)\sin(\theta_z) - \cos(\theta_y)\cos(\theta_z)\sin(\theta_x) & \cos(\theta_y)\cos(\theta_x) \end{bmatrix} \quad (31)$$

Where R_x denotes rotation around the x axis. When saving back to disk into a BVH file, the Euler angles from the R_{BVH} rotation matrix defined above can be extracted through simple trigonometry.

4.2.2. Annotation File

The annotation file is a major component of Beyond Mocap. Below an example file, with time stamps specified in seconds. Lines are indicated for illustration purposes only:

```
[Line 1] 0 8.5
[Line 2] 8.5 9.8 right dribble
[Line 3] 9.8 top kick low right
[Line 4] 9.8 15
[Line 5] 15 20 celebration arms inside
```

Time stamps are used to state when certain movements should appear in the synthesized motion. When specifying **frame constraints**, which constrain the motion to be synthesized to a specific frame of the mocap database or indicate a key moment in the mocap file (such as the exact moment a ball is kicked), a single timestamp must be used indicating the moment in time it should occur (Line 3 in the example above):

```
[Frame Constraint Timestamp] [Annotation #1] … [Annotation #N]
```

For all other situations (locomotion, before or after key moments) both a start and end timestamps must be specified:

```
[Start Timestamp] [End Timestamp] [Annotation #1] … [Annotation #N]
```

Timestamps can be indicated in either frame numbers or seconds. Whenever the motion is a transition motion, thus not locomotion nor a frame constraint such as setup for a free kick, annotations must not be set in the corresponding line of the annotation file (Lines 1 and 4 in in the example above).

4.2.3. Positional Tracking Data

In this project, the positional tracking data consists of only the X and Y coordinates of the soccer player (no height data), specified in *meters* and sampled at 10Hz. Besides parsing the coordinates and player names, the tool which handles this data also **calculates the player's orientation** through its position difference between consecutive data points**; performs automatic locomotion annotation** (as mentioned in Chapter 3.1.2) based on the current soccer player speed, obtained through translation delta**;** and **time scales the input data into any desired framerate**. Since the motion capture data is given at a frequency of 30 frames per second and the input data at 10Hz, we can simply interpolate and obtain positional coordinates at a frequency of 30Hz in order to match it with the mocap data

We expect a much better looking synthesized motion by, in a future version of the tool, adding filtering on the position coordinates in order to obtain a smoother sequence of orientations; using past data and with it infer the most likely orientation, or maybe even obtaining the real orientation from a better source of tracking data.

4.3. Optimization

Through the reduction of the number of features by application of PCA (Chapter 3.2.1) we already lower computation time during synthesis. But more can be done to remove redundant calculations through serialization.

The Accord.NET Framework[6] implementation of Principal Component Analysis is used to reduce the size of the feature vector of a motion frame. The resulting principal components and transformation matrix are then saved on local disk. Since the original data will not change unless new mocap is added to the database, we can continue using the same transformation on further executions of Beyond Mocap.

Given that the mocap in the database is read-only, we also serialize the computed feature vectors as well as the resulting feature vectors after dimensionality reduction through PCA.

As for clustering through k-means, we use ALGLIB's[7] implementation. Clustering is performed for all possible block sizes (thus for 32, 16 and 8 frame blocks) before synthesis begins and the results saved to disk. The clusters themselves, with their associated feature vectors and frame blocks, are also serialized.

By only performing PCA and k-means computations once, and saving the results to disk, we only have to load them the next time the application is started, decreasing the time it takes to synthesize a motion. The gains obtained here are presented in

[6] http://accord-framework.net/

[7] http://www.alglib.net/

Chapter 5.2. It should also be possible, and would be interesting to evaluate the benefits of pre-computing and serializing continuity between frame blocks.

5. Evaluation and Results

To determine if the new animation system is successful in achieving the goals defined at the start of the research, a qualitative evaluation was performed alongside a quantitative analysis. For the qualitative evaluation, a pilot study was performed as described in Chapter 5.1, and for the quantitative evaluation several aspects of the synthesized motion were measured and are presented in Chapter 5.2.

5.1. Motion Naturalness

From Q1 defined in Chapter 1.1, one of the objectives is to research how to synthesize an animation that is more natural and smooth-looking than the current solution. Since naturalness is more of a subjective evaluation, a limited pilot study should be the best approach to analyze how, and if, Beyond Mocap improves on this aspect.

The main hypothesis that can be extracted from the mentioned research question is as follows:

[H1]

The motion output by Beyond Mocap is more natural and smooth looking
than the motion output by the current system.

Since foot-skating is a visible factor that can influence how natural and realistic a motion looks, this hypothesis can be complemented by the following sub-hypothesis:

[H1.1]

The foot-skating removal algorithm (Chapter 3.2.4)
contributes favorably to the naturalness of the motion.

In the next sub-chapter the pilot study is detailed and in 5.1.2 the results are presented.

5.1.1. Pilot Study Setup

In order to measure the level of naturalness and smoothness as seen by the end-user, a total of 5 soccer scenarios of about 20 seconds long were chosen, each having 2 or 3 frame constraints. These scenarios are excerpts taken from real soccer matches:

Scenario 1) Gerrard Scenario: The player starts with a slow walk, picks up some speed and starts running, performs a short dribble followed by a kick and finally runs a while longer finishing his run with a goal celebration.

Scenario 2) Messi Scenario: The player walks and jogs around for a little while. He then receives the ball with his foot at a low height, performs a long dribble close to the penalty area followed by a pass and finally runs towards the goal in order to perform a scoring header. The scenario ends with the player celebrating the goal.

Scenario 3) Vilhena Scenario: After running and jogging around for a while, the player performs a slide to intercept the ball but does not end in possession. A team mate does, however, allowing the player to run towards the goal and kick at a low height with the inside of the foot, scoring a goal. The scenario ends with the player celebrating the goal.

Scenario 4) Lennon Scenario: In this scenario, the player receives a ball lobbed at him with his chest bouncing over to his feet. Even though he does not maintain possession of the ball after these moves, he is still able to later score a goal with the top of his foot at a low height.

Scenario 5) Hein Scenario: The player in this scenario tries to score a goal by kicking the ball at the line of the penalty area. The ball bounces off the top bar and he finishes his try by converting it into a goal with a bicycle kick.

For each scenario, three video clips were rendered: with the motions provided by the current system, by Beyond Mocap with foot-skating still present and by Beyond Mocap with foot-skating minimized, totaling 15 video clips. In order to render these video clips, each motion was imported into MotionBuilder in a mockup environment so that the participant could focus on the motion itself, and finally rendered with the camera following the hips of the player at a set distance (as shown in Figure 14).

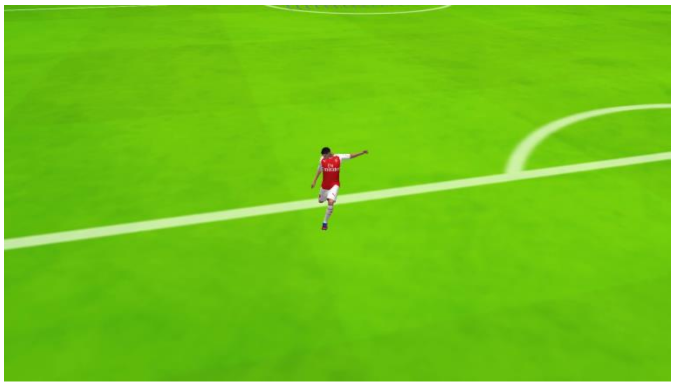


Figure 14: Screen capture from one of the rendered videos.

These clips were randomly presented to the participants so that they would not know which system output they were looking at. Not only randomly presented between system outputs but also between scenarios, meaning one survey order could be Scenario 1 Video 2, Scenario 4 Video 1, Scenario 4 Video 3, and so on; and between participants so that no participant would have the same order of videos being shown.

A total of 12 people took part in the survey, all employees of Triple, but none had seen the output by the new Beyond Mocap animation system up until the point of participating in the survey nor were they directly involved with Beyond Sports development. This should remove any potential bias in the answers given.

To collect the necessary data, an online survey was created (Appendix C). The survey was designed to let the participant know the objective is to analyze the naturalness of the character animation and to rate each animation shown in a 1-5 Likertscale: Really bad, Bad, Neither good nor bad, Good and Really good. Before any video clip was shown, one question was posed in order to have an indication of how much each participant knows and is aware of character animation.

At the beginning, two training examples are shown which demonstrate both extremes in the level of naturalness but without a mention about which one is being presented. These examples do not count towards the evaluation and are used only to give an indication of the difference in realism in the generated motions that the participant could expect. After the training examples, the set of videos with the animations to be evaluated in terms of level of realism are presented.

5.1.2. Analysis

Diverging stacked bar charts are used to present the results of the survey, as suggested by [Robbins and Heiberger 2011], where the bars are centered on the 0% line, with the percentage of participants who rated the motions positively in terms of naturalness shown to the right, and negatively to the left.

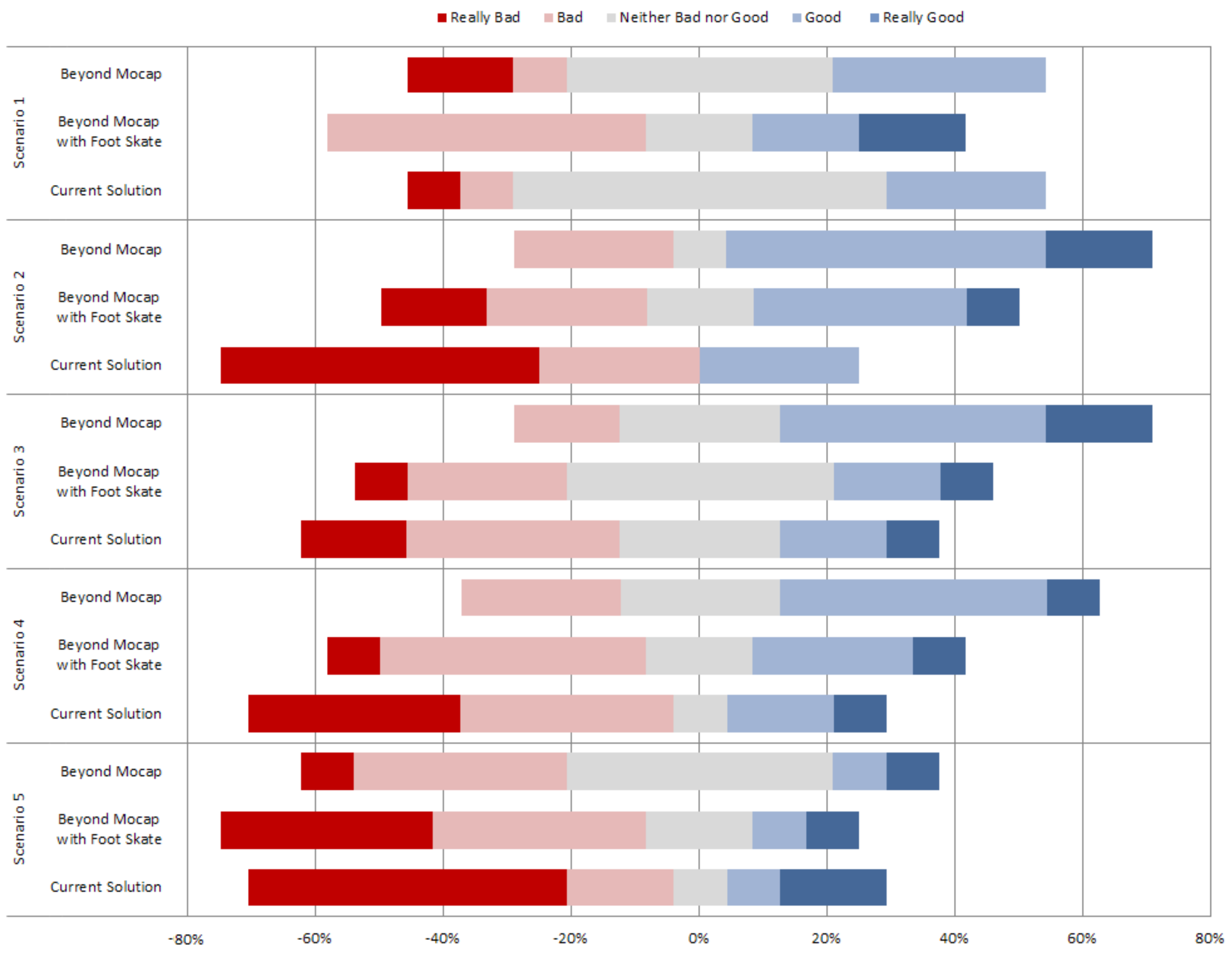


Figure 15: Pilot survey results grouped per scenario.

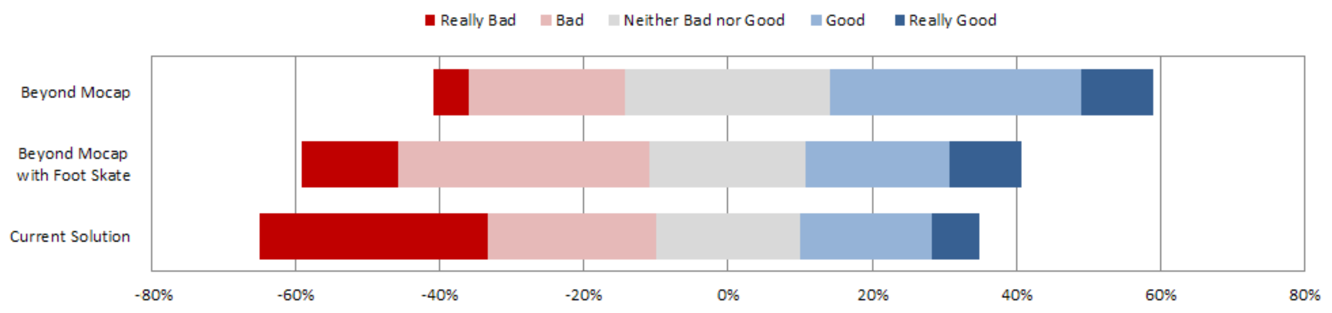


Figure 16: Pilot survey results grouped per system.

The results show participants rated the motions generated by the **Current Solution** (**CS**) as being generally the least realistic and natural looking, and the motions generated by the **Beyond Mocap** (**BM**) system as the most natural looking. Even though the motions synthesized by **Beyond Mocap with foot-skating present** (**BMFS)** were not expected to be rated significantly higher than those synthesized by CS, that is what the results show. This is could be due to them being smoother, as shown by the data presented in Chapter 5.2.1.

By performing a one-tailed t-test on the results shown in Figure 16, with a null hypothesis stating there is no difference in terms of naturalness between the motions obtained from Beyond Mocap and Current Solution, we observe the improvement in realism and naturalness of the motion synthesized by BM over CS is statistically significant for an alpha value of 0.05 (p-value of 0.0002). Consequently, we accept hypothesis **H1** presented in Chapter 5.1. The same can be concluded regarding hypothesis **H1.1**, for a corresponding "there is no improvement in naturalness with the foot-skate removal algorithm" null hypothesis, since the difference in naturalness of BM over BMFS is statistically significant for the same alpha value of 0.05 (p-value of 0.016), we accept hypothesis **H1.1**.

From this survey we can also notice that foot-skating is relatively accepted by the participants in general, to a much higher degree than the motion having discontinuities or not looking natural by itself. This could be due to the absence of shadow in the rendered video. It would be interesting to follow up on this study with a more extensive evaluation, with a larger pool of participants, a greater number of scenarios and videos with better lighting, and thus casting shadows, as well as the presence of the ball in the motion; thus providing better data for a more detailed analysis and representative results.

5.2. Quantitative Analysis

Other aspects of the synthesized motion and of the approach implementation were measured and analyzed quantitatively and, in the sub-chapters below, we present the most relevant ones for the project. These include *motion smoothness*, which is part of the main objective of this research, as well as *matching positional tracking data* and *matching annotations*. As for the implementation, since automation and minimal involvement of the user is desired, we measure the time it takes to annotate a scenario for one player and compare it with the current solution. This chapter ends with a performance analysis covering some of the choices taken during implementation.

5.2.1. Synthesized Motion

Smoothness

The smoothness of the synthesized animation can be computed by analyzing the feature vectors per frame throughout the whole duration of the motion. By measuring the distance between feature vectors of consecutive frames, we obtain information about corresponding breaks in smoothness. The results of these measurements are presented in full in Appendix D). Feature vector distance is without units since the values obtained are computed using the reduced feature vector through PCA as described in Chapter 3.2.1.

In these charts, we are only interested in the peaks shown since they occur whenever the character suddenly changes direction or in sudden transitions between different motions. Due to the first, Beyond Mocap outputs animations with some jerkiness. However, across all scenarios, the animations synthesized by Beyond Mocap have less jerkiness and are, therefore, smoother than the ones provided by the current solution.

Matching Positional Tracking Data

Another desired aspect of the synthesized motion is to match, as closely as possible, the position of the character in the synthesized motion to the position provided by the tracking data. This is all the more important during frame constraints since these refer to key moments of the match. A header in the synthesized animation several meters away from the actual position may not look unrealistic when looked at outside of any context, but once the animations are used in the Beyond Sports simulation, this would make the character perform a header meters away from the ball. In the table below the average distance error (in meters) throughout the synthesized motion is reported:

Average distance error (m)	*Scenario 1*	*Scenario 2*	*Scenario 3*	*Scenario 4*	*Scenario 5*
Whole animation	4.52	3.42	3.24	1.21	1.86
At Frame Constraints	1.20	2.14	3.50	3.19	1.20

Table 1: Average distance error between the position of the virtual character and the position given by the tracking data in meters.

Unfortunately, the approach used to “catch up” with the distance lost after frame constraints described in Chapter 3.3.2 does not provide natural looking animations for scenarios where non-locomotion motions are not spaced by several seconds. In these situations, it causes the character to move at an unnaturally slow or fast speed and, therefore, the results shown in Chapter 5 are obtained with motion synthesized without this approach. The distance error shown in the table above reflect both the error obtained with the catching up approach removed as well as due to the frames removed during continuity minimization as described in Chapter 3.2.5, which in turn alter the distance travelled and successively adds to the distance error.

The higher distance error at frame constraints in Scenarios 3 and 4 can be explained by the existence of long locomotion sections, where the discontinuity minimization approach progressively adds or removes distance travelled. In Scenario 4, this is compounded by the fact the mocap database does not include motions of players receiving the ball with the chest, followed by catching it with the foot, while moving forward. Therefore, for the second part of this constraint, the virtual character is still in the same place receiving the ball with his foot while the match data reports him being several meters away.

By having a different method of continuity minimization, or having it integrated in an adapted synthesis algorithm which also includes distance travelled constraints, we expect these distance errors to be minimized without negatively affecting the naturalness of the synthesized animation.

Matching Annotations

Finally, since the synthesized motion is essentially replicating a real match scenario, the motions the virtual character performs throughout should match those performed by the real soccer player. It is possible to get an indication of how similar the motions are in terms of overall movement by comparing the annotations of the synthesized motion with the desired set of annotations. These are only an indication since, for example, just a running motion can be performed in many different ways that cannot be discretely represented by text annotations.

Average Annotation Similarity	*Scenario 1*	*Scenario 2*	*Scenario 3*	*Scenario 4*	*Scenario 5*
Whole animation	-0.05	-0.54	-0.09	-0.34	-0.25
At Frame Constraints	-1	-0.24	-0.75	-0.44	-0.8

Table 2: Average annotation similarity between the synthesized motion and the desired annotations. A value of '-1' represents 100% match and a value of '1' represents a 0% match.

As shown in Table 2, all of the synthesized motions were more similar to the desired annotation than they were dissimilar. In the same way matching positional tracking data at frame constraints is more important than outside of such moments, having the exact desired motion, by way of annotation, at frame constraints is critical to the synthesis of a motion which replicates well its corresponding soccer match scenario.

Since the speed at which the player is moving is not annotated but computed based on the positions in the real match data, the corresponding annotation can change at a high frequency if the player is moving at a boundary speed between two different locomotion speeds (such as between running and jogging). Therefore, when compared with the annotations of the synthesized motion, a large quantity of frames will not have the exact same locomotion annotation because the synthesis algorithm chooses the best fitting motion per block of frames. The lower annotation similarity values at frame constraints for Scenario 2 when compared with the whole animation is due to the fact one of the constraints required a jogging header when the mocap database only contains standing and running headers.

5.2.2. Implementation

Annotation Effort

As stated in the introduction, due to the current solution being prohibitively time-consuming due to the amount of manual work required, Beyond Mocap should minimize the time it takes for a user to annotate motions for each player while ensuring key moments occur at the correct timestamp. For this measurement, we counted the time it takes to annotate the same scenarios as those used in the pilot study.

Average Annotation Time (s)	*Scenario 1*	*Scenario 2*	*Scenario 3*	*Scenario 4*	*Scenario 5*
Current Solution	437	436	377	460	230
Beyond Mocap without annotation tool	137	258	136	138	144

Table 3: Time in seconds that it takes to annotation motions for a player while ensuring key moments happen at the correct timestamp.

Although the annotation process for the Beyond Mocap system requires the user to manually type the annotations for each player with the corresponding timestamp, it is already about 2.5 times faster than with the current system. Once the Match Annotation Tool proposed in Chapter 4 is in place, this reduction in annotation time effort should be more noticeable.

Performance Analysis

Below we report the motion synthesis run time for each scenario used in the pilot study, for both Beyond Mocap with foot-skate as well as with the foot-skate minimization algorithm present. Each motion was synthesized 5 times in a release build on a laptop PC with 16GB of RAM and an Intel Core i7-5600U processor running at 2.60Ghz.

	Scenario 1		*Scenario 2*		*Scenario 3*		*Scenario 4*		*Scenario 5*	
Motion Duration (s)	20		22.5		22.5		21		19.5	
	FS	FSM	FS	FSM	FS	FSM	FS	FSM	FS	FSM
Synthesis Time (s)	65	148	95	215	34	353	199	477	94	614

Table 4: Average synthesis run time duration, in seconds, for each scenario split into motions generated with foot-skate (FS) and motions generated with the foot-skate minimized by the approach described in Chapter 3.2.4 (FSM).

Results show the original synthesis is quite fast but, once the foot-skate minimization algorithm is included, it can lead to a tenfold increase in total synthesis time. This is due to how the algorithm works when adding locomotion cycles to match the desired travel distance. In Scenario 5, for example, a large part of a locomotion section is composed of walking motions.

Since the computational cost is quadratic in the number of frame blocks needed to match the desired distance travelled, and each walking locomotion cycle travels a small distance, we will need a larger number of walking frame blocks to travel the same distance, versus using running frame blocks, thus increasing the final synthesis time as shown in the table above.

By having position constraint enforcement in the same main algorithm step of motion synthesis presented in Chapter 3.2, we expect synthesis time to be brought down since, as it stands, the synthesis of locomotion sections is being performed twice.

Finally, as mentioned in Chapter 4.3, through serialization of PCA's principal components and transformation matrix, feature vectors and reduced feature vectors of all motions in the database, and the result of k-means clustering for all frame block sizes (32, 16 and 8 frame blocks), we decrease the time it takes to synthesize each motion by approximately 187 seconds. Running k-means clustering on a database of 150 motion captures with an average duration of approximately 120 frames long we obtain the clusters for frame blocks 8 frames long in about 29.6 seconds, for frame blocks 16 frames long in about 49.5 seconds and for frame blocks 32 frames long in 94.7 seconds (average values over 3 runs). The difference in computation time between frame blocks of different lengths is due to larger augmented feature vectors in longer frame blocks, requiring more operations for distance calculation. Computing feature vectors for all motions take approximately 5.9 seconds and running PCA about 7 seconds. These are significant gains when synthesizing several motions.

6. Conclusion

Beyond Mocap is an automated offline data-driven motion synthesis framework, requiring no user input for the synthesis itself. The synthesized motion respects positional constraints provided by the positional tracking data, with a certain margin of error (see Chapter 5.2.1), although with better results for key moments. These key moments are defined as frame constraints and are provided by the user through an annotation system. Through the annotation system, Beyond Mocap also ensures these frame constraints occur at user defined time-stamps.

Motions synthesized by Beyond Mocap are more realistic-looking and have a higher degree of naturalness when compared to the motions in the current solution used by Beyond Sports, partially due to foot-skating minimization as well as smoother transitions between different motions.

Although the synthesis process takes longer when compared with the current solution, this is less important as the resulting animation is more natural looking. The time it takes to annotate the motion to be synthesized is significantly lower, however, allowing the algorithm to be run in the background while the user performs other tasks.

6.1. Limitations

Since Beyond Mocap synthesizes motion by splicing blocks of frames from a mocap database, it will not be able to synthesize new motions that do not have a corresponding base mocap to extract frames from. This means the vocabulary must match the database. Issues can arise from this limitation whenever the desired annotations describe inexistent motions, leading to a synthesis of possibly highly discontinuous motion, depending on the value set for α. As an example, if the desired annotation is "*jogging header*" and the database only contains motions with "*standing header*" and "*running header*", this could lead the algorithm to output a motion where the character is jogging and suddenly stops to perform a standing header. A consequence of this include the discontinuity minimization step (Chapter 3.2.5) removing, or adding, too many frames to increase better connectivity between frame blocks, which in turn decreases, or increases, the overall length of the motion as well as significantly changes the distance travelled by the virtual character. This is undesired as it further increases the distance between the virtual character's position and the real soccer player's position as provided by the positional tracking data.

Positional constraint enforcement through time-warping as proposed in Chapter 3.3.2 is also limited in its application. In the described approach, we change the speed at which the animation is played in order to have the character "catch up" with the data. For cases where the locomotion section between key moments is not long enough, the character will appear as running unrealistically fast, or unrealistically slow, causing the overall feeling of naturalness to disappear.

Because our approach is an improvement over the work from [Arikan et al. 2003] in terms of position constraint enforcement, most of the limitations of their framework are also present in Beyond Mocap. The main synthesis algorithm works by selecting any block of frames after any other block of frames, independent of the motion they are extracted from. Therefore, transitions with some discontinuity can be chosen in case there are no transition motions in the database between the selected frame blocks. Working with blocks of frames also limit the length of motions to be included in the database to motions longer than the set initial block size.

The magnitude of discontinuity still present in the output motion, as well as the degree to which the annotations are respected, can be specified by tweaking the α variable in order to give more weight to one or the other. It is essentially a decision between more continuous motions or motions that replicate the match scenario more faithfully. Its importance in the synthesis algorithm is, therefore, decreased by the addition of more motions to the mocap database.

Movement orientation for the synthesized motion is not ideal due to it being obtained from the positional tracking data. Information regarding facing direction of the soccer player is not provided, and the position coordinates are not tracked in detail either, causing the data to be rather noisy. Since rotation is computed from the difference between position coordinates, this can cause the character in the synthesized motion to rotate unrealistically. Consequently, motions that require the virtual character to move in a different direction than the facing direction, such as goalkeepers or a "*walking back*" motion, are not supported by our approach. By setting movement direction in the last stage of the motion synthesis, some minimal foot-skating is also introduced since it is not part of the main synthesis algorithm and is performed without any constraints on foot plants.

Finally, the initial setup of Beyond Mocap can be time intensive due to the necessity of having the underlying database to be fully annotated. Regarding the implementation itself, the whole synthesis process is not fully automated yet since it needs further testing with cases where the motion to be synthesized has several non-locomotion motions in a short time-frame.

6.2. Future Work

The described approach was chosen due to its simplicity and could be complemented with several other techniques to obtain more variation in the final output motion as well as higher degrees of realism. Motion adaptation (Chapter 2.2.1) could be applied on individual mocap clips in order to have the virtual soccer player limbs reach to the correct position of where the contact point should be, which would allow, for example, for kicks at any desired height. It could also provide locomotion at different speeds, possibly reducing the computational time of the proposed position constraint enforcement.

Blending (Chapter 2.2.2) between locomotion mocap clips would provide another method of increasing variation in locomotion animation instead of simply using the distance travelled matching and time warping approach we take in enforcing position constraints. Such blending techniques could also be an alternative to the discontinuity minimization technique described in Chapter 3.2.5 that would minimize or completely prevent changes in the character's distance travelled.

Using machine learning, as described in the reference paper, would eliminate much of the time it takes to annotate whenever new motions are added to the database. This could also be used for some sort of informed automatic annotation by analyzing the general character movement leading to, and away from, key moments, as well as by taking the ball coordinates as input since these are also provided by the positional tracking data.

On the implementation side, a visual interactive tool for player timeline annotation, presented in a decision-tree that could traversed through key sequence presses or similar, would further reduce the time it takes to provide the set of desired annotation, while simultaneously ensuring the vocabulary is used correctly and preventing the usage of annotations with no corresponding motion in the database.

With respect to the long synthesis running time caused by the foot-skate minimization through the addition, or removal, of locomotion cycles, improvements could be brought by investigating the possible inclusion of distance travelled and both character and motion orientation as constraint in the main synthesis algorithm. As consequence, this could enable synthesis of motions where the character moves in a different direction than where he faces, such as in motions for goalkeepers.

Finally, and as stated in the last paragraph of Chapter 5.1.2, a more extensive evaluation, with a larger pool of participants, a bigger number of scenarios, more detailed videos with a more realistic lighting which casts shadows, and the presence of the ball in the motion, would enable a more detailed analysis and possibly offer new insights to guide future work.

References

Okan Arikan and D.A. Forsyth. 2002. Interactive Motion Generation from Examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 483–490. DOI:http://dx.doi.org/10.1145/566570.566606

Okan Arikan, David A. Forsyth, and James F. O'Brien. 2003. Motion Synthesis from Annotations. In *ACM SIGGRAPH 2003 Papers*. New York, NY, USA: ACM, 402–408. DOI:http://dx.doi.org/10.1145/1201775.882284

G. Ashraf and Kok Cheong Wong. 2001. Constrained framespace interpolation. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*. 61–72. DOI:http://dx.doi.org/10.1109/CA.2001.982378

Armin Bruderlin and Lance Williams. 1995. Motion Signal Processing. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 97–104. DOI:http://dx.doi.org/10.1145/218380.218421

Samuel R. Buss and Jay P. Fillmore. 2001. Spherical Averages and Applications to Spherical Splines and Interpolation. *ACM Trans Graph* 20, 2 (April 2001), 95–126. DOI:http://dx.doi.org/10.1145/502122.502124

Benoît Le Callennec and Ronan Boulic. 2004. Interactive Motion Deformation with Prioritized Constraints. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 163–171. DOI:http://dx.doi.org/10.1145/1028523.1028545

Kwang-Jin Choi and Hyeong-Seok Ko. 2000. Online motion retargetting. *J. Vis. Comput. Animat.* 11, 5 (December 2000), 223–235. DOI:http://dx.doi.org/10.1002/1099-1778(200012)11:5<223::AID-VIS236>3.0.CO;2-5

Andrew Feng, Yazhou Huang, Marcelo Kallmann, and Ari Shapiro. 2012. An Analysis of Motion Blending Techniques. In Marcelo Kallmann & Kostas Bekris, eds. *Motion in Games*. Springer Berlin Heidelberg, 232–243.

T. Geijtenbeek and N. Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graph. Forum* 31, 8 (2012), 2492–2515. DOI:http://dx.doi.org/10.1111/j.1467-8659.2012.03189.x

Michael Gleicher. 2001. Comparing Constraint-Based Motion Editing Methods. *Graph Models* 63, 2 (March 2001), 107–134. DOI:http://dx.doi.org/10.1006/gmod.2001.0549

Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. 2003. Snap-together Motion: Assembling Run-time Animations. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*. I3D '03. New York, NY, USA: ACM, 181–188. DOI:http://dx.doi.org/10.1145/641480.641515

Rachel Heck and Michael Gleicher. 2007. Parametric Motion Graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. New York, NY, USA: ACM, 129–136. DOI:http://dx.doi.org/10.1145/1230100.1230123

Leslie Ikemoto, Okan Arikan, and David Forsyth. 2007. Quick Transitions with Cached Multi-way Blends. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. New York, NY, USA: ACM, 145–151. DOI:http://dx.doi.org/10.1145/1230100.1230125

Lucas Kovar and Michael Gleicher. 2004. Automated Extraction and Parameterization of Motions in Large Data Sets. In *ACM SIGGRAPH 2004 Papers*. SIGGRAPH ’04. New York, NY, USA: ACM, 559–568. DOI:http://dx.doi.org/10.1145/1186562.1015760

Lucas Kovar and Michael Gleicher. 2003. Flexible Automatic Motion Blending with Registration Curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 214–224.

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 473–482. DOI:http://dx.doi.org/10.1145/566570.566605

Lucas Kovar, John Schreiner, and Michael Gleicher. 2002. Foot-skate Cleanup for Motion Capture Editing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York, NY, USA: ACM, 97–104. DOI:http://dx.doi.org/10.1145/545261.545277

Jehee Lee, Jinxiang Chai, Paul S.A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’02. New York, NY, USA: ACM, 491–500. DOI:http://dx.doi.org/10.1145/566570.566607

Jehee Lee and Kang Hoon Lee. 2004. Precomputing Avatar Behavior from Human Motion Data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 79–87. DOI:http://dx.doi.org/10.1145/1028523.1028535

Jehee Lee and Sung Yong Shin. 1999. A Hierarchical Approach to Interactive Motion Editing for Human-like Figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 39–48. DOI:http://dx.doi.org/10.1145/311535.311539

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion Fields for Interactive Character Locomotion. In *ACM SIGGRAPH Asia 2010 Papers*. New York, NY, USA: ACM, 138:1–138:8. DOI:http://dx.doi.org/10.1145/1866158.1866160

Yan Li, Tianshu Wang, and Heung-Yeung Shum. 2002. Motion Texture: A Two-level Statistical Model for Character Motion Synthesis. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 465–472. DOI:http://dx.doi.org/10.1145/566570.566604

Wan-Yen Lo and Matthias Zwicker. 2008. Real-time Planning for Parameterized Human Motion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 29–38.

Michael Mandel. 2004. *Versatile and interactive virtual humans: Hybrid use of data-driven and dynamics-based motion synthesis.*

S. Ménardais, R. Kulpa, F. Multon, and B. Arnaldi. 2004. Synchronization for Dynamic Blending of Motions. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 325–335. DOI:http://dx.doi.org/10.1145/1028523.1028567

C. O'Brien, J. Dingliana, and S. Collins. 2011. Spacetime Vertex Constraints for Dynamically-based Adaptation of Motion-captured Animation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York, NY, USA: ACM, 277–286. DOI:http://dx.doi.org/10.1145/2019406.2019443

Masaki Oshita. 2008. Smart Motion Synthesis. *Comput. Graph. Forum* 27, 7 (October 2008), 1909–1918. DOI:http://dx.doi.org/10.1111/j.1467-8659.2008.01339.x

Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. 2002. Online Locomotion Generation Based on Motion Blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '02. New York, NY, USA: ACM, 105–111. DOI:http://dx.doi.org/10.1145/545261.545279

T. Pejsa and I.s. Pandzic. 2010. State of the Art in Example-Based Motion Synthesis for Virtual Characters in Interactive Applications. *Comput. Graph. Forum* 29, 1 (2010), 202–226. DOI:http://dx.doi.org/10.1111/j.1467-8659.2009.01591.x

Cheng Ren, Liming Zhao, and Alla Safonova. 2010. Human Motion Synthesis with Optimization-based Graphs. *Comput. Graph. Forum* 29, 2 (2010), 545–554. DOI:http://dx.doi.org/10.1111/j.1467-8659.2009.01624.x

Naomi B. Robbins and Richard M. Heiberger. 2011. Plotting Likert and other rating scales. In *Proceedings of the 2011 Joint Statistical Meeting*. 1058–1066.

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. 1998. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Comput. Graph. Appl.* 18, 5 (1998), 32–40.

Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. 1996. Efficient Generation of Motion Transitions Using Spacetime Constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 147–154. DOI:http://dx.doi.org/10.1145/237170.237229

Alla Safonova and Jessica K. Hodgins. 2007. Construction and Optimal Search of Interpolated Motion Graphs. In *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. New York, NY, USA: ACM. DOI:http://dx.doi.org/10.1145/1275808.1276510

Ari Shapiro, Fred Pighin, and Petros Faloutsos. 2003. Hybrid Control for Interactive Character Animation. In *Computer Graphics and Applications, Pacific Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 455. DOI:http://dx.doi.org/10.1109/PCCGA.2003.1238294

Hyun Joon Shin and Hyun Seok Oh. 2006. Fat Graphs: Constructing an Interactive Character with Continuous Controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '06. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 291–298.

Kwang Won Sok, Katsu Yamane, Jehee Lee, and Jessica Hodgins. 2010. Editing Dynamic Human Motions via Momentum and Force. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 11–20.

Madhusudhanan Srinivasan, Ronald A. Metoyer, and Eric N. Mortensen. 2005. Controllable Real-time Locomotion Using Mobility Maps. In *Proceedings of Graphics Interface 2005*. GI '05. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 51–59.

R. Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (June 1972), 146–160. DOI:http://dx.doi.org/10.1137/0201010

Deepak Tolani, Ambarish Goswami, and Norman I. Badler. 2000. Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs. *Graph. Models* 62, 5 (September 2000), 353–388. DOI:http://dx.doi.org/10.1006/gmod.2000.0528

H. Van Welbergen, B.J.H. Van Basten, A. Egges, Zs. M. Ruttkay, and M.H. Overmars. 2010. Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control. *Comput. Graph. Forum* 29, 8 (December 2010), 2530–2554. DOI:http://dx.doi.org/10.1111/j.1467-8659.2010.01822.x

Jing Wang and Bobby Bodenheimer. 2008. Synthesis and Evaluation of Linear Motion Transitions. *ACM Trans Graph* 27, 1 (March 2008), 1:1–1:15. DOI:http://dx.doi.org/10.1145/1330511.1330512

Andrew Witkin and Zoran Popovic. 1995. Motion Warping. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 105–108. DOI:http://dx.doi.org/10.1145/218380.218422

Weiwei Xing, Xiang Wei, Jian Zhang, Cheng Ren, and Wei Lu. 2014. Hybrid motion graph for character motion synthesis. *J. Vis. Lang. Comput.* 25, 1 (2014), 20–32. DOI:http://dx.doi.org/10.1016/j.jvlc.2013.10.001

Liming Zhao, Aline Normoyle, Sanjeev Khanna, and Alla Safonova. 2009. Automatic Construction of a Minimum Size Motion Graph. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New York, NY, USA: ACM, 27–35. DOI:http://dx.doi.org/10.1145/1599470.1599474

Liming Zhao and Alla Safonova. 2008. Achieving good connectivity in motion graphs. *Graph. Models* 71, 4 (2008), 139–152. DOI:http://dx.doi.org/10.1016/j.gmod.2009.04.001

Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. 2005. Dynamic Response for Motion Capture Animation. In *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. New York, NY, USA: ACM, 697–701. DOI:http://dx.doi.org/10.1145/1186822.1073249

Victor Zordan, Adriano Macchietto, Jose Medina, Marc Soriano, and Chun-Chih Wu. 2007. Interactive Dynamic Response for Games. In *Proceedings of the 2007 ACM SIGGRAPH Symposium on Video Games*. Sandbox '07. New York, NY, USA: ACM, 9–14. DOI:http://dx.doi.org/10.1145/1274940.1274944

Appendices

A) Annotation Vocabulary

The vocabulary was designed by taking into account the mocap present in the database. It was also cross-checked with another team-member at Beyond Sports more accustomed to soccer terms to make sure the vocabulary was comprehensive enough and that the motions were correctly annotated. It is composed of annotations for **continuous motions** and for **discrete motions**:

- Continuous motions annotations are used for cyclical motions which can be indefinitely extended and, when used, must have a start and end frame / second time stamp specified.

`standing, walking, jogging, running, sprinting, sitting, ground, celebration, hold, turning, dribble`

- Discrete motions annotations are used for discrete motions which contain a *key moment*: point in time where the actual described move is performed. In the example of a kick, the key moment would be when the foot touches the ball. When using these annotations, only the frame / second time stamp when the key moment happens should be specified.

There are also **modifier** annotations which must be present when annotating discrete motions: `right, left, front, back` which can be used for kicks, throws, headers, dives and so on. And `subtle, with_setup` which can be used for kicks.

All of the annotations are presented in a decision tree-like structure since each annotated frame block must contain all of the annotation words from parent to bottom child. As an example for the "stomp" motion:

```
stomp
    high
    middle
    low
    dive
        long_duration
            high
            low
        short_duration
            high
            low
```

When annotating with a dive stomp, one must annotate with the duration and if it is done high or low as well. Such as: `stomp dive long_duration high right.`

For Continuous Motions:

```
standing
    knee
    turning

walking
    back
    dribble
    turning

jogging
    dribble

running
    with
    dribble

sprinting
    dribble
```

```
sitting

ground

celebration
    running
        arms
            inside
    ender
        arms
            outside

hold
    high
    middle
    low
    ground
    sitting
```

For Discrete Motions:

```
receive
    foot
        low
            inside
        high
            inside
    body


kick
    low
        inside
            standing
            running
        outside
            standing
            running
        top
            standing
            running
        volley
            standing
            running
    high
        inside
            standing
            running
        top
            standing
            running
    bicycle
        half
```

```
throwb
    high
    middle


turning
    dribble
    letgo


header
    standing
        jump
    running
        jump


scissors


getting_up
    short_duration
    long_duration
    with


fall
    standing
    slide


fly
    slide
```

```
jump
    slide


intercept
    slide
    knee
        inside
        outside
            short_duration
            long_duration


catchb
    high
    middle
    low
    slide
    dive
        long_duration
            low
        short_duration
            high
            low


stomp
    high
    middle
    low
    dive
        long_duration
            high
            low
        short_duration
            high
            low
```

B) BVH Motion File Structure

The BVH motion file is composed of two sections:

- The HIERARCHY section describes the skeleton's hierarchy through its joints and their offsets plus which CHANNELS are available for motion specification.
- The MOTION section describes the motion itself by specifying the values for the CHANNELS of each joint (defined in the HIERARCHY section) per row. Each row contains the values for one frame with the values in the same order as the order of the joints in the HIERARCHY section.

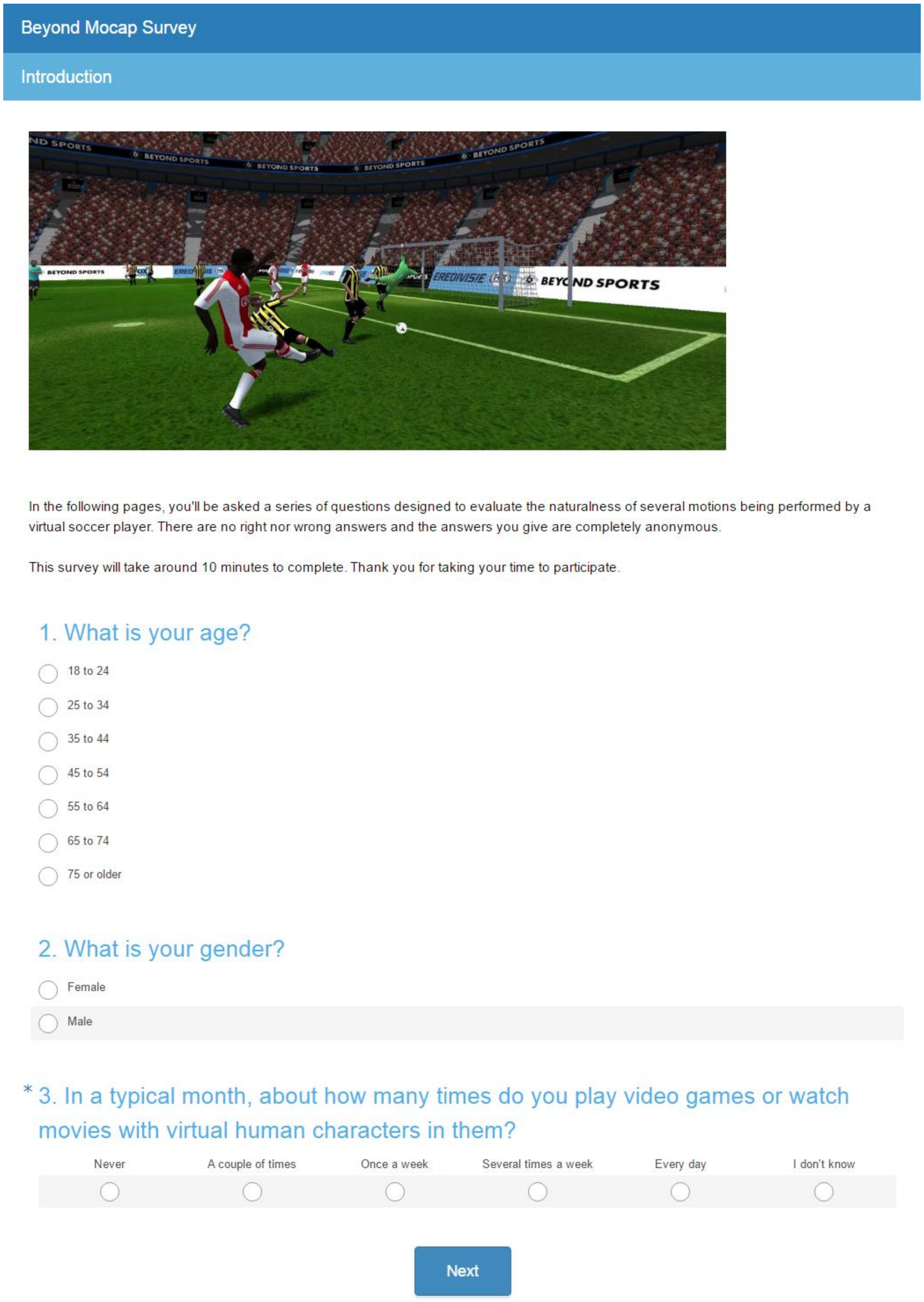
All distance units are usually specified in *centimeters* and rotation in *degrees*. Its general structure is as follows:

```
HIERARCHY
ROOT Hips
{
  OFFSET [x_value] [y_value] [z_value]
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT Spine
  {
    OFFSET [x_value] [y_value] [z_value]
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Ribcage
    {
      OFFSET [x_value] [y_value] [z_value]
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT Neck
      {
        OFFSET [x_value] [y_value] [z_value]
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT Head
        {
          OFFSET [x_value] [y_value] [z_value]
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET [x_value] [y_value] [z_value]
          }
        }
      }
      JOINT LeftShoulder
      {
[...]
}
MOTION
Frames: [n, motion duration in frames]
Frame Time: [duration in seconds of one frame. For a 30 frame per second motion, this value is 1/30 = 0.0333333]
[value of channel 1 in frame 1] [value of channel 2 in frame 1] [...] [value of channel c in frame 1]
[value of channel 1 in frame 2] [value of channel 2 in frame 2] [...] [value of channel c in frame 2]
[...]
[value of channel 1 in frame n] [value of channel 2 in frame n] [...] [value of channel c in frame n]
```

Where c represents the total number of channels: $c = nr_{joints} * 3 + 3$.

C) Pilot Study Survey

Page 1



Beyond Mocap Survey

Introduction

BEYOND SPORTS
EREDIVISIE

In the following pages, you'll be asked a series of questions designed to evaluate the naturalness of several motions being performed by a virtual soccer player. There are no right nor wrong answers and the answers you give are completely anonymous.

This survey will take around 10 minutes to complete. Thank you for taking your time to participate.

1. What is your age?

- ○ 18 to 24
- ○ 25 to 34
- ○ 35 to 44
- ○ 45 to 54
- ○ 55 to 64
- ○ 65 to 74
- ○ 75 or older

2. What is your gender?

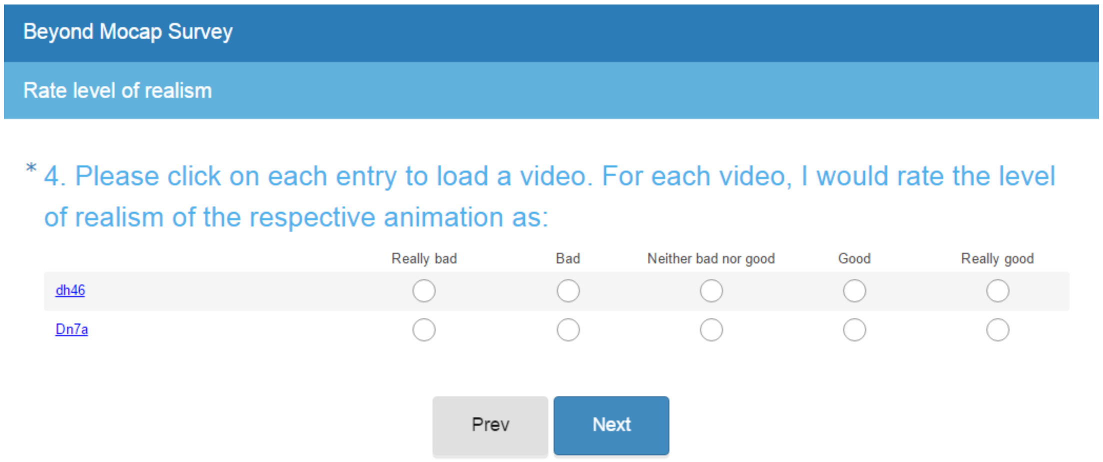
- ○ Female
- ○ Male

* 3. In a typical month, about how many times do you play video games or watch movies with virtual human characters in them?

Never	A couple of times	Once a week	Several times a week	Every day	I don't know
○	○	○	○	○	○

Next

Page 2



Beyond Mocap Survey

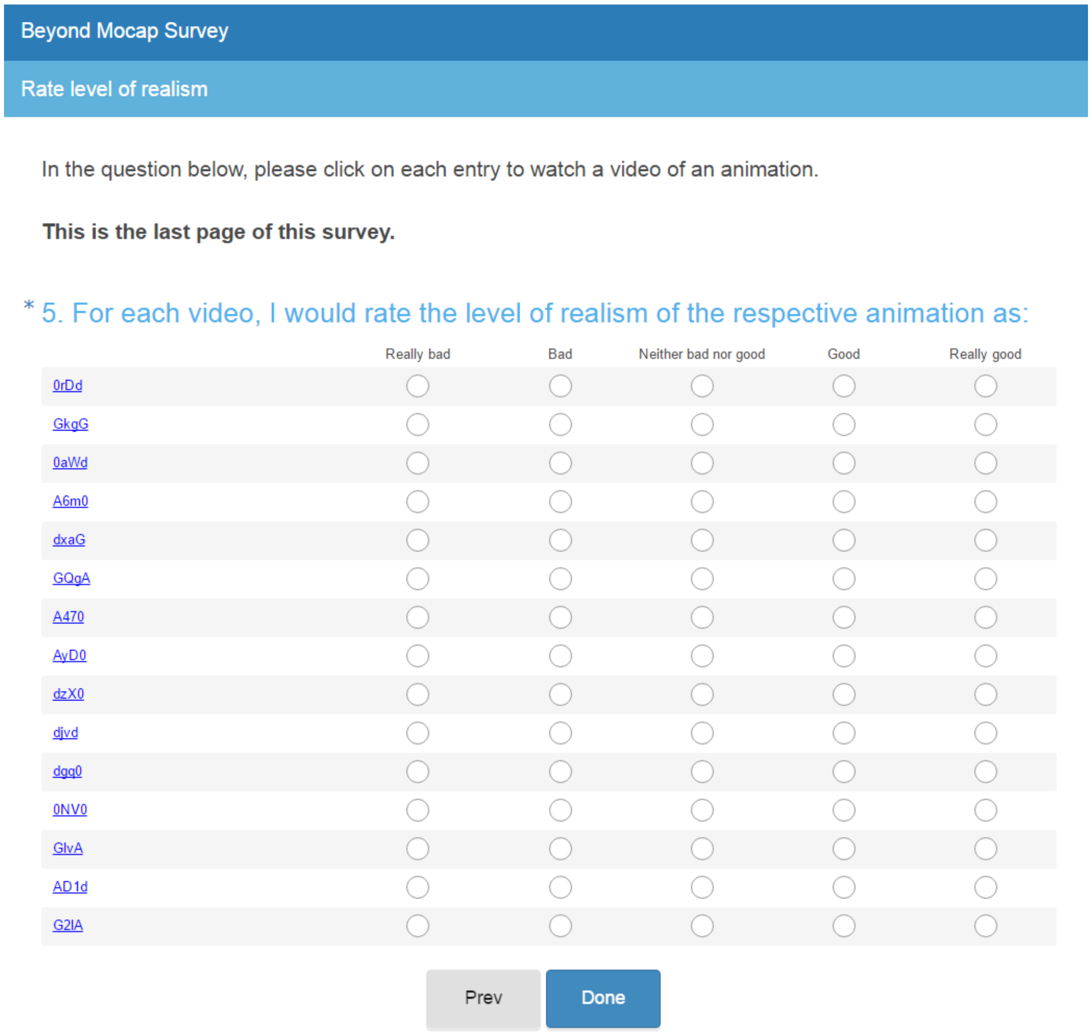
Rate level of realism

* 4. Please click on each entry to load a video. For each video, I would rate the level of realism of the respective animation as:

	Really bad	Bad	Neither bad nor good	Good	Really good
dh46	○	○	○	○	○
Dn7a	○	○	○	○	○

Prev Next

Page 3



Beyond Mocap Survey

Rate level of realism

In the question below, please click on each entry to watch a video of an animation.

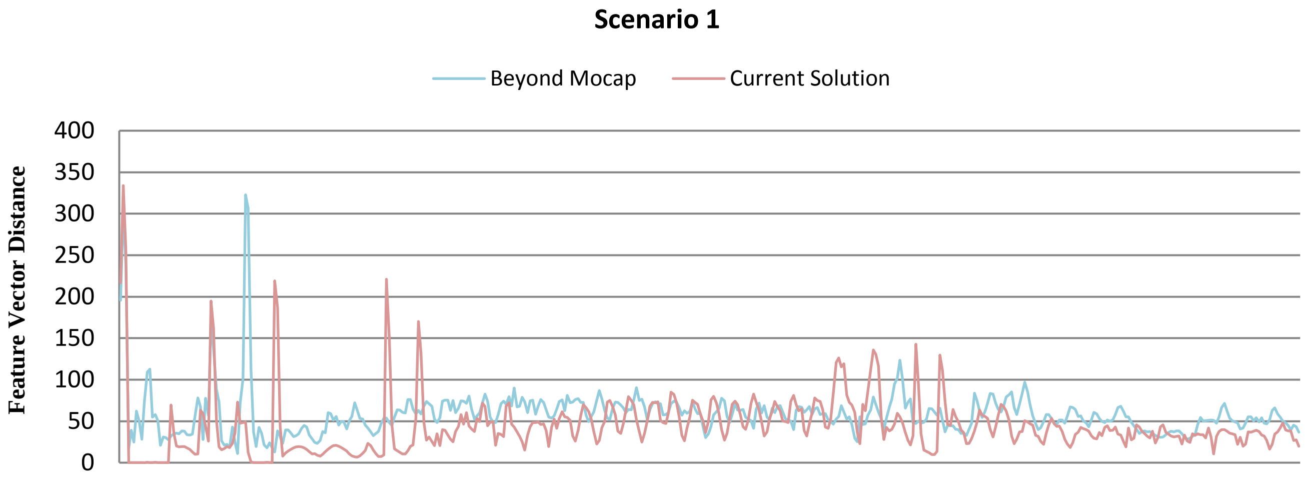
This is the last page of this survey.

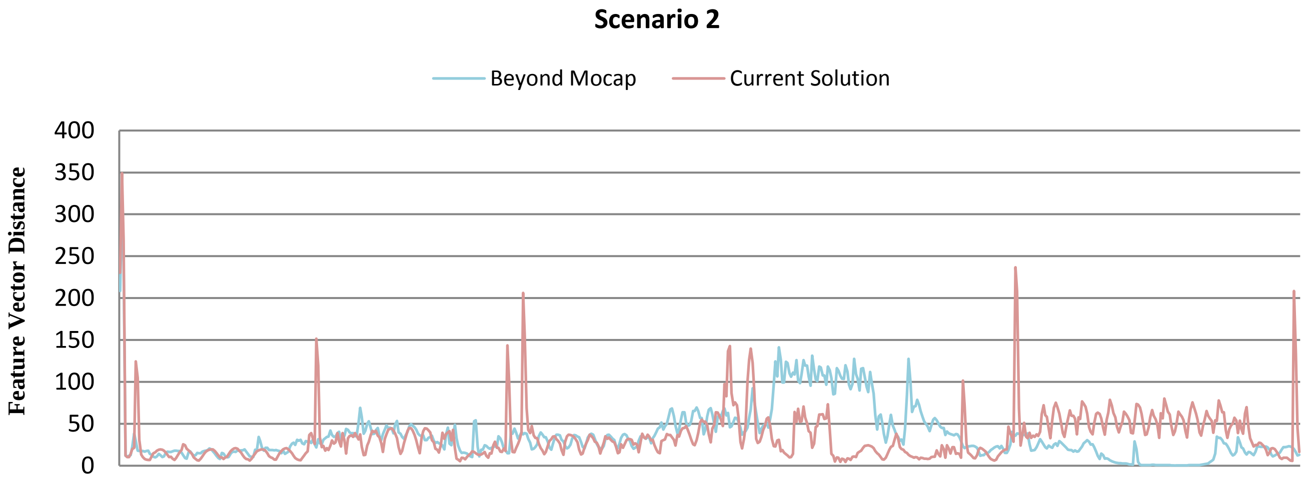
* 5. For each video, I would rate the level of realism of the respective animation as:

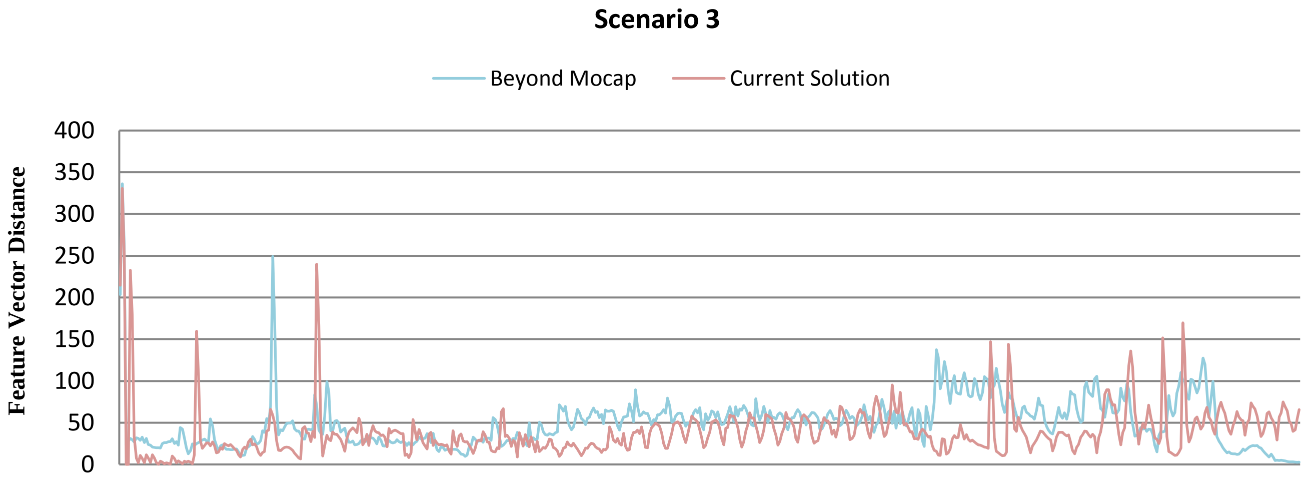
	Really bad	Bad	Neither bad nor good	Good	Really good
0rDd	○	○	○	○	○
GkgG	○	○	○	○	○
0aWd	○	○	○	○	○
A6m0	○	○	○	○	○
dxaG	○	○	○	○	○
GQgA	○	○	○	○	○
A470	○	○	○	○	○
AyD0	○	○	○	○	○
dzX0	○	○	○	○	○
djvd	○	○	○	○	○
dgq0	○	○	○	○	○
0NV0	○	○	○	○	○
GlvA	○	○	○	○	○
AD1d	○	○	○	○	○
G2lA	○	○	○	○	○

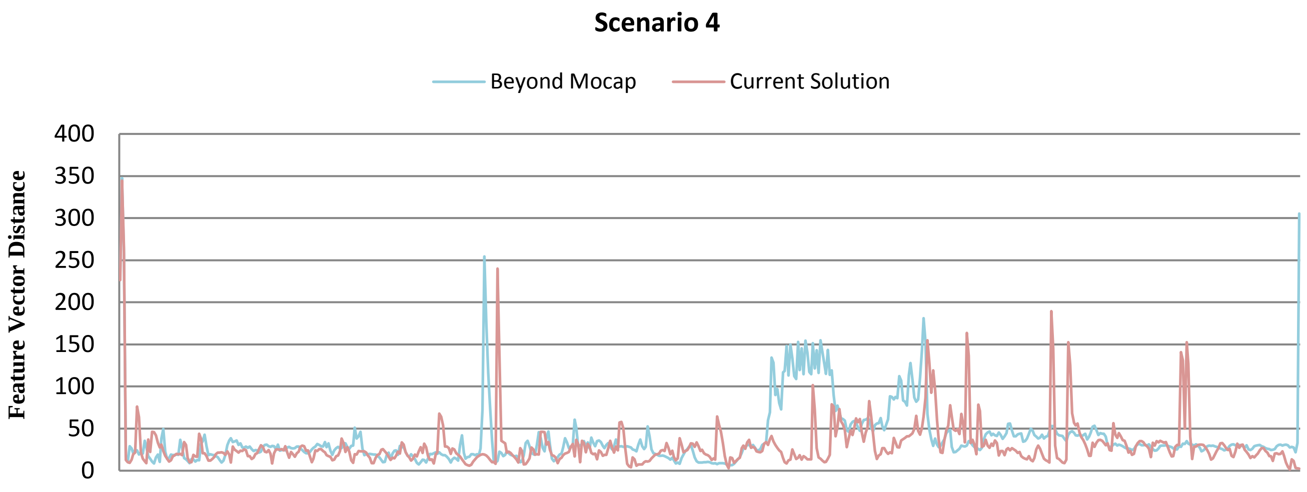
Prev Done

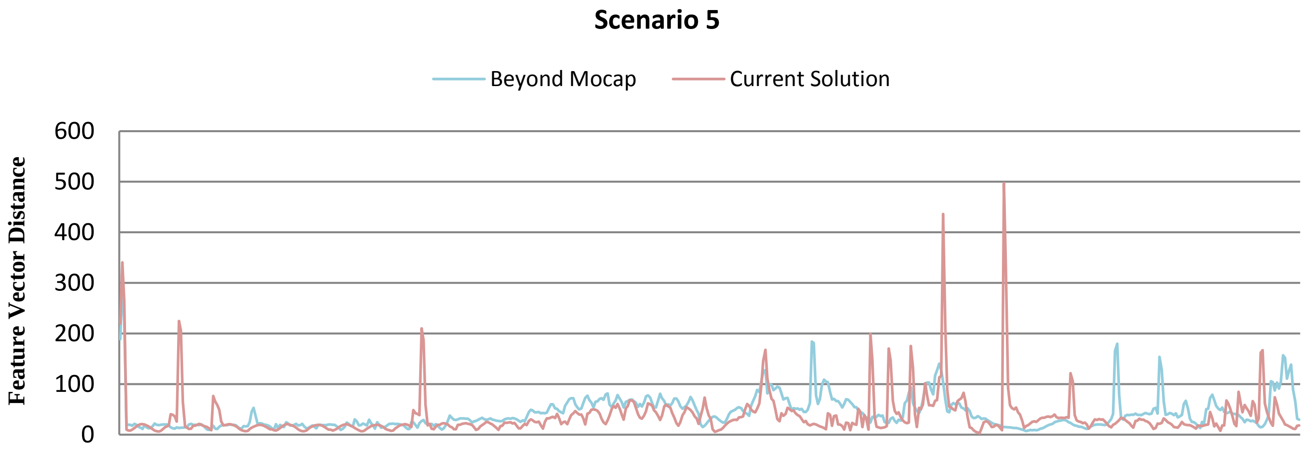
D) Smoothness Analysis








Scenario 4
Beyond Mocap
Current Solution
Feature Vector Distance
400
350
300
250
200
150
100
50
0


Scenario 5
Beyond Mocap
Current Solution
Feature Vector Distance
600
500
400
300
200
100
0