Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2006)
M.-P. Cani, J. O'Brien (Editors)

Robust Kinematic Constraint Detection for Motion Data

Benoît Le Callennec and Ronan Boulic

Virtual Reality Lab, Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland [†]

Abstract
Motion capture data is now widely available to create realistic character animation. However, it is difficult to reuse without any additional information. For this reason, annotating motion data with kinematic constraints is a clever step to ease further operations such as blending or motion editing. Unfortunately, prior automatic methods prove to be unreliable for noisy data and/or lack genericity. In this paper, we present a method for detecting kinematic constraints for motion data. It detects when an object (or an end-effector) is stationary in space or is rotating around an axis or a point. Our method is fast, generic and may be used on any kind of objects in the scene. Furthermore, it is robust to highly noisy data as we detect and reject aberrant data by using a least median of squares (LMedS) method. We demonstrate the accuracy of our method in various motion editing contexts.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation

1. Introduction

One popular technique to rapidly produce high-quality character animations is *motion capture*: the animation is recorded by mimicking the motion of an actor. The final animation must hence be carefully planned before the capture is done and is only valid for characters having the same proportions as the live performer. For this reason, these animations are not directly reusable and need additional adaptations.

Constraint-based motion editing methods are designed to this end. They are useful to change existing motion sequences while retaining as many of their initial characteristics as possible. These latter are often made explicit using kinematic constraints. In particular, kinematic constraints emphasizing that an object (or an end-effector) is stationary in space or is rotating around an axis or a point are often of interest for the animators. Unfortunately, manually defining all of them is tedious and time-consuming. For example, manually defining all the footplants of an animation may take minutes if not more. Consequently, it is often desirable to automatically detect such constraints in order, for example, to simplify and speed up the motion editing process.

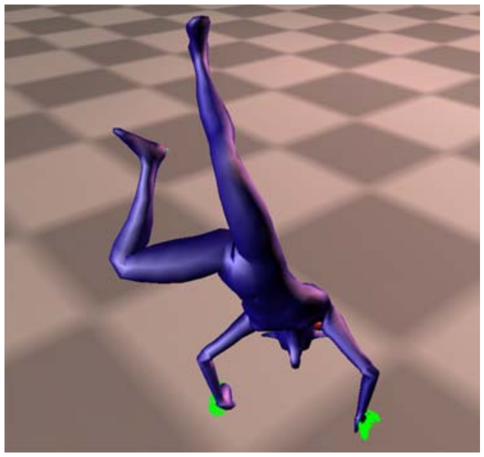
Figure 1: ***A break dance motion.*** *Robust constraint detection on raw motion capture data. Only active constraints are shown. Though the motion is highly noisy, our algorithm accurately detects when the hands should be planted.*

Methods relying on the position, velocity and/or acceleration of the end-effectors are clearly not reliable when working with motion capture data (see Figure 2).

Indeed, derivatives tend to amplify noise in signals. Other

[†] email:{benoit.lecallennec|ronan.boulic}@epfl.ch

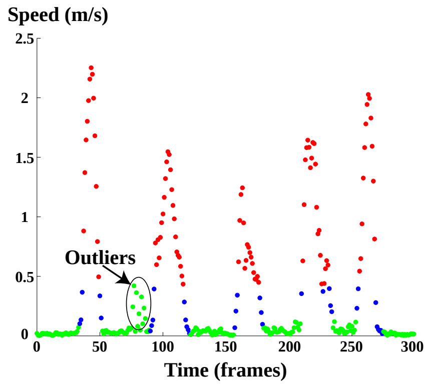

Figure 2: *Velocity-based methods are not reliable for noisy motions. In this example, the character is climbing a ladder (see accompanying video).* ***Red dots:*** *unlabeled frames.* ***Green dots:*** *manually labeled frames.* ***Blue dots:*** *mislabeled frames. Due to the presence of outliers in the data, the velocity threshold is overestimated to ensure that we detect* at least *all the expected constraints (green dots). This leads to a high number of mislabeled frames (blue dots).*

methods relying on learning algorithms lack genericity and are difficult to generalize to any object or end-effector.

This paper presents a fast and generic method that automatically detects kinematic constraints in potentially highly noisy data such as motion capture. Our algorithm successfully identifies events like a character body part being stationary, rotating around an axis or even rotating around a point. The animator is then free to edit or remove them to retain only those of interest. We use these constraints to edit long motion sequences though they are generic enough to index a database or for motion blending.

In the next section, we review previous work. Section 3 gives an overview of our algorithm. Sections 4, 5, 6 and 7 detail its important stages. Section 8 presents several applications and experimental results. Finally, we present future work and conclude this paper in Section 9.

2. Previous Work

Very few constraint detection techniques can be found in the literature. A classical application of constraint detection is the identification of footplants in motions. Several methods [KSG02, MKMA04] use specific thresholds on the position and velocity of the feet to detect them. Similarly, Lee et al. extend this approach in [LCR*02] to body segments and objects in the environment. They consider their relative velocity and position to decide whether a body segment is in contact with an object in the scene or not. However, these methods are not reliable for motion capture animation as derivatives tend to amplify noise in signals.

Bindiganavale and Badler [BB98] present a method mapping the animation of a subject being motion captured to another character having different proportions. They essentially focus on motions containing interactions with the surrounding environment. To avoid checking for collisions at every frame of the animation, they suppose that potential frames of interest are located at the zero-crossing of the second derivative of the end-effectors' trajectories. For the same reason as above, this method is then not reliable when working with noisy signals. Moreover, they require manually tagged-objects to avoid checking for collisions with all the objects in the scene.

Liu and Popović [LP02] propose a generic method based on geometric transformation properties to extract constraints. This method is dedicated to keyframed animation and is not intended to be applied to motion capture as it does not consider noise in the data.

Finally, Ikemoto et al. [IAF06] use a learning method to automatically identify footplants and correct them using IK. This method is dedicated to footplants detection and would be difficult to generalize to any kind of effectors and/or constraints. Indeed, detecting another type of constraints would require to build a new kind of feature vectors and to train the classifier once more. Additionally, they rely on the character's leg configuration to detect a footplant. As a result, they need to annotate frame examples for each different footplant configuration, however much noise is present in the data. Conversely, we detect a constraint by modeling the noise in the data itself. Thus, for the particular case of footplant detection, we only need one example constraint to detect all the remaining constraints even for very dissimilar footplant configurations.

All these methods prove to be unreliable for noisy signals and/or lack in genericity to be applied in other contexts. In this paper we present an algorithm for constraint detection for motion capture animation. We demonstrate, using several examples, that our algorithm is generic, robust and that it accurately detects constraints even on highly noisy data.

3. Method Overview

In this section, we first define specific terminology used in this paper. We then give an overview of our algorithm.

3.1. Definition and Terminology

Instantaneous Constraints

Given an object O, an instantaneous constraint represents all the points in space remaining stationary with respect to some displacement $\mathbf{D}$ *from one frame to the next*. Instantaneous constraints then last only one frame. These sets of stationary points may be of dimension:

- 3: all the points in space are stationary i.e. O is stationary.

- 1: all the points on a line in space are stationary i.e. O is rotating around an axis.

Note that no rigid transformation such that a single point or all the points belonging to a plane remain stationary in space exists [Cha30].

Kinematic Constraints

A kinematic constraint (called constraint in the remainder) is built by merging neighboring instantaneous constraints. In particular, point constraints result from the merging of several intersecting line constraints. As a result, constraints may be of three types depending on whether object O is:

- Stationary in space. This is a **space constraint**.
- Rotating around an axis. This is a **line constraint**.
- Rotating around a point. This is a **point constraint**.

A constraint only occurs during a specific time interval $[kc_{begin}, kc_{end}]$. Sliding constraints are more difficult to detect and necessitate complex minimization techniques. Hence, we do not consider this group of constraints as we mainly focus on interactive techniques.

Template Constraints

A template constraint is an example constraint provided by the animator to help the algorithm compute important thresholds. We call these thresholds the *noise pattern* as they are fundamental to take into account the noise in the data.

Outliers

The main feature of our algorithm is to deal with noise in the data. While *Gaussian noise* is handled using thresholds, it is much more difficult to deal with outliers. An outlier is an observation that lies outside the overall pattern of a distribution [MM99]. In our case, we consider outliers over the duration of template constraints only: an outlier is a frame which is annotated as being part of a template constraint while it should not. It is important to note that outliers usually induce an important bias into the results and must be taken into account when present in the data.

3.2. Algorithm Overview

To simplify the exposition and without loss of generality, we focus our discussion on detecting constraints related to a single animated object in the scene. Given an animated object O, the steps of the algorithm are as follows (see Figure 3):

1. **Discretize the animation.**
2. **Express the results as displacements** between each pair of consecutive frames.
3. **Project the displacements onto the $\sigma\epsilon$-space.** This space characterizes the noise for each displacement.
4. **Compute the noise pattern**, given a small set of template constraints. This step is critical as it must consider outliers to avoid detecting too many constraints.
5. **Generalize the noise pattern** to the rest of the data to detect all the instantaneous constraints.
6. **Merge neighboring instantaneous constraints** as much as possible to end up with meaningful constraints.

Steps 4. and 5. may be repeated for long motion sequences to hierarchically refine the noise pattern. Steps 1. and 5. are trivial and are thus not explained further. Steps 2., 3., 4. and 6., are respectively detailed in Sections 4, 5, 6 and 7.

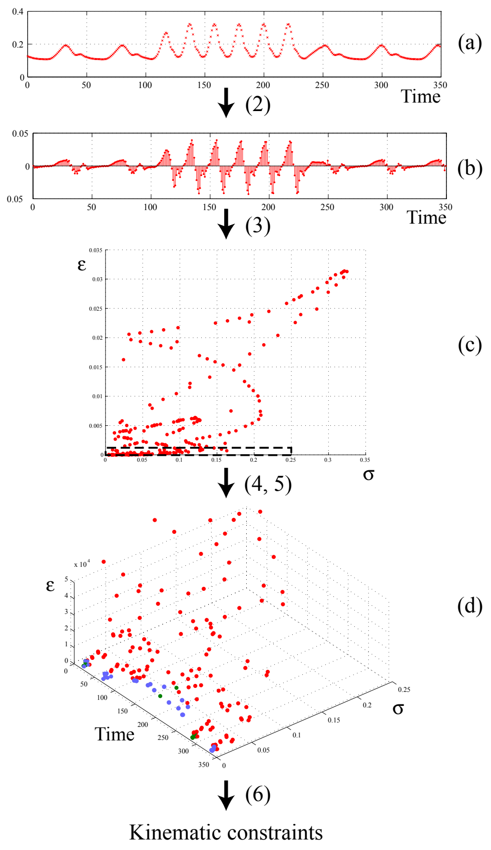

Figure 3: *Main steps of our constraint detection algorithm (see Section 3.2 for an explanation of steps 2, 3, 4, 5 and 6).* ***(a)*** *Discretized animation.* ***(b)*** *Associated displacements.* ***(c)*** *Displacements after projection onto the $\sigma\epsilon$-space.* ***(d)*** *Close up view of (c) after computation of the noise pattern.* ***Red dots:*** *original data.* ***Green dots:*** *template constraints.* ***Blue dots:*** *detected instantaneous constraints.*

4. Displacement computation

Given a discretized animation of an object O, we first express its motion as displacements from one frame to the next. Let us consider $\mathbf{W}_i$ the matrix transforming, at frame i, a point $\mathbf{p}$ expressed in the O local coordinate system to $\mathbf{x_i}$ expressed in the world coordinate system. Then we have:

$$\hat{\mathbf{x}}_\mathbf{i} = \mathbf{W_i}\hat{\mathbf{p}} \tag{1}$$

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

$$\begin{aligned}\mathbf{\hat{x_{i+1}}} &= \mathbf{W_{i+1}\hat{p}} \\ &= \mathbf{W_{i+1}W_i}^{-1}\mathbf{W_i\hat{p}} \\ &= \mathbf{W_{i+1}W_i}^{-1}\mathbf{\hat{x}_i} \\ &= \mathbf{D_i\hat{x}_i}\end{aligned}$$

with $\mathbf{D_i}$ the displacement of point $\mathbf{x}$ between frame i and frame $i+1$. This formulation is similar to the one presented in [LP02]. It represents the displacement of O from frame i to frame $i+1$ with respect to the world coordinate system. However, this *global formulation* is not efficient when working with noisy data, as it introduces a bias in subsequent stages of the algorithm. Indeed, in this formulation, $\mathbf{D_i}$ directly dependents on the global position of O at frames i and $i+1$. We therefore reformulate the problem by expressing $\mathbf{D_i}$ with respect to the previous position of O:

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

This *local formulation* is more accurate as it is independent of the global position of O. We give a demonstration in Appendix A and we show numerical comparisons between both formulations.

5. Projection onto the $\sigma\epsilon$-space

We now have all the displacements $\mathbf{D_i}$ related to O. Each $\mathbf{D_i}$ is then expressed as a $\sigma\epsilon$-point in the $\sigma\epsilon$-space. Given the displacement $\mathbf{D_i}$ of O from frame i to frame $i+1$, we need to find *all the points* $\mathbf{p}$ remaining stationary in space. More formally, we have to solve:

$$\mathbf{D_i\hat{p}} = \mathbf{\hat{p}} \tag{2}$$

$$(\mathbf{D_i} - \mathbf{I_4})\mathbf{\hat{p}} = 0 \tag{3}$$

where $\mathbf{\hat{p}}$ is the homogeneous coordinates of a point $\mathbf{p}$ expressed in the O local coordinate system. In our framework, we are interested in finding *all* the solutions of Equation (3). Suppose that $\mathbf{D_i}$ represents a rotation along an axis $\mathbf{R_{axis}}$. It is clearly not satisfactory to know that a specific point (actually lying on the axis $\mathbf{R_{axis}}$) is stationary is space: we need to precisely determine $\mathbf{R_{axis}}$. As a consequence, a straightforward method finding a single solution (the least squares for example) is not usable. As $\mathbf{D_i}$ is a rigid transformation, it can be rewritten as:

$$\mathbf{D_i} = \begin{bmatrix} \mathbf{R_i} & \mathbf{t_i} \\ \mathbf{0_3} & 1 \end{bmatrix} \tag{4}$$

with $\mathbf{R_i}$ and $\mathbf{t_i}$ respectively denotating the rotation and translation components of $\mathbf{D_i}$. We then reformulate Equation (3):

$$(\mathbf{R_i} - \mathbf{I_3})\mathbf{p} + \mathbf{t_i} = 0 \tag{5}$$

$$\mathbf{Ap} = -\mathbf{t_i} \tag{6}$$

Using a singular value decomposition, we express $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U\Sigma V}^T \tag{7}$$

with $\mathbf{U}$ and $\mathbf{V}$ being 3×3 orthogonal matrices and $\mathbf{\Sigma}$ a 3×3 diagonal matrix [PTVF92]. Matrix $\mathbf{\Sigma}$ contains the 3 *singular values* $\sigma_{i=3,2,1}$ (with $\sigma_3 > \sigma_2 > \sigma_1$) of $\mathbf{A}$. To completely solve Equation (6) we must compute:

1. The particular solution $\mathbf{p_{particular}} = -\mathbf{V\Sigma}^{-1}\mathbf{U}^T\mathbf{t_i}$ to Equation (6).
2. A basis of the nullspace of $\mathbf{A}$.

Matrices $\mathbf{U}$ and $\mathbf{V}$ span the *range* and the *nullspace* of matrix $\mathbf{A}$. In particular, the columns of $\mathbf{V}$, whose same-numbered elements σ_i are *zero*, are an orthonormal basis for the nullspace of $\mathbf{A}$ [PTVF92]. The complete solution (i.e. the instantaneous constraint) is then given by the particular solution $\mathbf{p_{particular}}$ and the basis of the nullspace of $\mathbf{A}$. However, no σ_i is exactly zero due to the noise. High-dimensional instantaneous constraints are then difficult (actually impossible) to detect. Moreover, Equation (6) sometimes produces a solution while it should not. We then need to additionally check whether the current solution is relevant or not. To do so, we compute the residual error ϵ of Equation (6) as:

$$\epsilon = ||\mathbf{Ap_{particular}} + \mathbf{t_i}|| \tag{8}$$

Depending on ϵ, we then determine whether the solution is acceptable or not.

In summary, we can influence the solution of Equation (6) using two values:

- σ_{max}: threshold below which the singular values are zeroed. The number of null singular values defines the dimension of the instantaneous constraint.
- ϵ_{max}: residual error of equation (6) (after zeroing the singular values smaller than σ_{max}) below which the solution is acceptable.

σ_{max} and ϵ_{max} are called the noise pattern in the remainder. Each $\mathbf{D_i}$ is then expressed as a $\sigma\epsilon$-point $P_{\sigma\epsilon}(\sigma_3, \epsilon_3, \sigma_1, \epsilon_1)$ with ϵ_i the residual error of Equation (6) after zeroing the singular values equal or smaller than σ_i. The problem of detecting instantaneous constraints can then be reformulated as estimating the noise pattern $(\sigma_{max}, \epsilon_{max})$ so that we effectively detect the *expected* instantaneous constraints.

6. Noise Pattern Computation

An accurate estimation of the noise pattern is crucial for the algorithm. Indeed, if it is underestimated (resp. overestimated), the algorithm detects too few (resp. too many) instantaneous constraints. Several issues then arise:

- It is not reasonable to ask the animator to provide such parameters as the results tend to be difficult to foresee.
- These parameters are very different from one motion to another.

- Motions containing outliers are common. In such a situation, the estimation is much more arduous.

Hence, we rely on the animator to provide a small set of template constraints to help the algorithm calibrate the noise pattern and then accurately detect the instantaneous constraints for the entire animation.

The user specifies a template constraint with a time interval $[t_{begin}, t_{end}]$ and a dimension (i.e. space, line or point). A template constraint may then be thought of as a set $S_{\sigma\epsilon}$ of $\sigma\epsilon$-points $P_{\sigma\epsilon i}$ for which we precisely know in advance the solution of Equation (6). A straightforward and naive method is to first estimate σ_{max} so that the solution of Equation (6) is of the required dimension for *all* $P_{\sigma\epsilon i}$. Then, ϵ_{max} is computed to accept the solution at frames where a template constraint has been specified. In other words, if the template constraint is:

- **A space constraint then** $\sigma_{max} \geq \sigma_{3_i}$ and $\epsilon_{max} \geq \epsilon_{3_i}$ for all $P_{\sigma\epsilon_i}$ in $S_{\sigma\epsilon}$.
- **A line or a point constraint then** $\sigma_{max} \geq \sigma_{1_i}$ and $\epsilon_{max} \geq \epsilon_{1_i}$ for all $P_{\sigma\epsilon_i}$ in $S_{\sigma\epsilon}$. Indeed, a point constraint is the intersection of a set of line constraints.

However, such an approach is not efficient as it leads to an overestimation of σ_{max} and ϵ_{max} in the presence of outliers. Suppose that a template space constraint is specified between frame a and frame b. Suppose also that during this period of time, the frame i (with $a < i < b$) is an outlier. Then, σ_{max} is overestimated to ensure that the solution of Equation (6) is of dimension 3 for the specified interval and particularly for the intervals $[i-1, i]$ and $[i, i+1]$. Moreover, overestimating σ_{max} directly leads to an overestimation of ϵ_{max} as well. Finally, the algorithm tends to detect too many instantaneous constraints over the rest of the animation.

We therefore introduce in the next section a robust technique to identify and reject outliers during the computation of the noise pattern.

6.1. Robust Computation of the noise pattern

In this section, we focus our discussion on the estimation of the noise pattern based on a single template constraint only. In the next section, we detail how this method is extended to handle several template constraints simultaneously.

We introduce a robust computation of the noise pattern based on the least median of squares method (LMedS) [RL87] to identify and reject potential outliers. Let SV be the set of singular values containing:

- All the σ_1 of the $\sigma\epsilon$-points if the specified template constraint is a line constraint.
- All the σ_3 of the $\sigma\epsilon$-points if the specified template constraint is a space constraint.

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

1. For each $\sigma_i \in SV$, we compute the median of its squared residuals M_i as:
$$M_i = med\ r_i^2(\sigma_i, SV)$$
where $r_i^2(\sigma_i, SV)$ are the residual errors associated to σ_i with respect to each element in SV.
2. We retain M_{min} (and its associated singular value σ_{med}) the smallest M_i among all the M_is.
3. We then compute the robust standard deviation as:
$$\hat{\sigma} = 1.4826[1 + 5/(N_{SV} - 1)]\sqrt{M_{min}}$$
where M_{min} is the minimal median and N_{SV} the number of singular values.
4. Finally, we reject all the singular values such that:
$$r_i^2(\sigma_i, SV) \geq (2.5\hat{\sigma})^2$$

The reader can refer to [RL87] for a more detailed explanation of the LMedS method. It is important to note that in our case, we *do not* perform any random selection as the space of possible solutions is SV. We can therefore afford to estimate *all* the possible solutions as they are relatively few. σ_{max} is then the maximum of all the *good* singular values remaining in SV. Finally, ϵ_{max} is estimated to detect the expected instantaneous constraints using the same method. Figure 4 shows a comparison of both methods. While our LMedS method accurately detects two footplants, the naive method yields to an erroneous estimation by merging both footplants into a single one. Figure 5 shows the associated numerical values. The animation represents two footplants. While first frames are part of the first footplant, they should not be retained during the computation of the noise pattern. As a result, the naive method only detects one footplant. Using the LMedS method, this bias is avoided and our method produces accurate results.

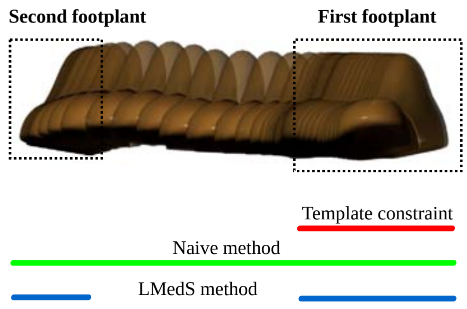

Figure 4: *Footplant detection.* ***Red:*** *given template constraint.* ***Green:*** *detected constraint using the naive method.* ***Blue:*** *detected constraints using our LMedS method.*

6.2. Dynamic Noise Pattern

Given a single template constraint, we can estimate the noise pattern for the whole motion. However, if the motion is long,

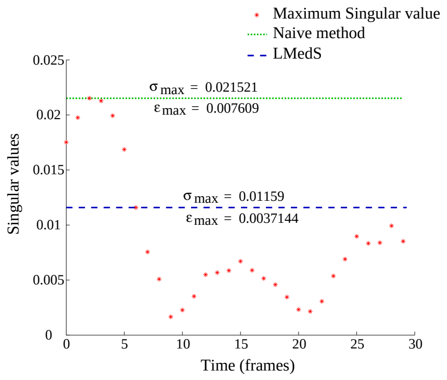

Figure 5: *Comparison between naive and LMedS methods.*

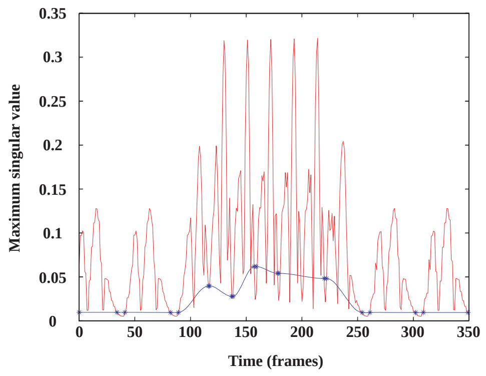

Figure 6: *An example of the σ_{max} curve of a dynamic noise pattern.* ***Red:*** *Maximum singular values.* ***Blue:*** *The σ_{max} curve associated to the dynamic noise pattern.*

it often contains different actions. Hence, a correct noise pattern for a particular part of the motion may not be suited for the entire animation. We thus propose to represent the noise pattern as cubic interpolation splines instead of static thresholds. For each given template constraint, we robustly compute the associated noise pattern. Each noise pattern then defines two control points: one control point for the first frame of the template constraint, and one for its last one. We then ensure that the noise pattern is accurate for the period of time defined by a given template constraint. Figure 6 shows in particular the σ_{max} curve of a noise pattern. This dynamic noise pattern provides a flexible way to define different values of thresholds depending on the time in the animation. Subsequent additions of template constraints may refine the curves associated with the noise pattern ensuring a more accurate instantaneous constraint detection.

7. Constraint Merging

Given a list of instantaneous constraints, we need to compute the minimal set of constraints. For instance, if during n consecutive frames there are $n-1$ space instantaneous constraints, we want to replace all of them by a single space constraint that lasts for n frames. We thus need to merge instantaneous constraints as much as possible by checking for two requirements:

- **In space:** we need to check for some kind of *spatial intersection* between both constraints. For example, two line constraints may intersect and result in a line if they are the same, in a point otherwise.
- **In time:** we need to check whether the constraints are *temporally connected* or not: the last frame of the first constraint corresponds to the first frame of the second one. For example, two line constraints may intersect in space but occur at different times (they are separated by several frames). In this case, they should not be merged.

Two constraints have to meet both requirements to be merged. The concept of spatial intersection between space, line or point constraints is self-explanatory and is not explained further. Note that the noise pattern additionally contains two values:

- α_{max}: threshold below which two lines are parallel.
- $dist_{max}$: threshold below which two lines or points are intersecting.

7.1. Temporal Connection

When the motion contains outliers, the method presented in the previous section discards the associated frames during the instantaneous constraint detection step: they are computed with respect to thresholds that are estimated as conservatively as possible to avoid detecting too many of them. It may then happen that a constraint is artificially sliced in spite of being continuous in time.

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

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

See appendix B for more details on the construction of f_{tol}. We can then robustly compute whether two constraints are temporally connected or not.

8. Experimental Results

In this section, we present experimental results of our constraint detection algorithm. Each example is presented in the accompanying video. The final system is integrated into Alias^{TM}/Maya 5 as plug-ins and MEL scripts. The first section shows constraint detection with synthetic data. We then show results using raw motion capture animations. Detected constraints are finally directly enforced using our motion editing method presented in [LB04]. Note that *no detected constraint* is edited before applying our motion editing algorithm.

8.1. Synthetic Data

The initial animation contains 141 frames. It represents two dice rolling on a dice carpet. We detect space constraints, line constraints and point constraints on both dice. The results are shown in the accompanying video. Note that as we work with synthetic animations, the noise is not as large as in motion capture animations. We thus use minimal thresholds to handle numerical errors instead of template constraints.

8.2. Walking Around Motion

In this example, the character walks around during 400 frames (16 seconds). We detect the space constraints associated to both feet. We only need to specify 2 template constraints: one for each foot. The results are shown in Figure 7. All the constraints are correctly detected. However, some may be ambiguous. When the character turns, footplants are quite difficult to identify even for the animator. In this case, it may happen that the detected constraints are too short or even sliced while they *maybe* should not. Refining the dynamic noise pattern using one additional template constraint solves this problem.

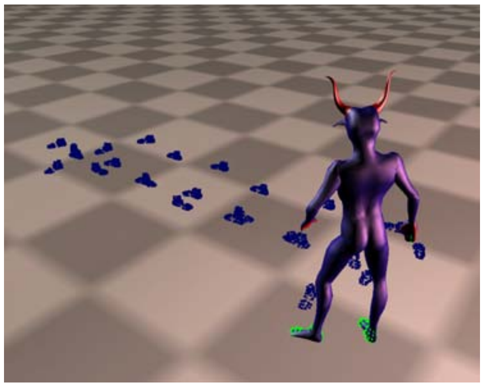

Figure 7: ***Walking around motion.*** *The constraint detection is applied to both feet. The constraints are displayed in green when they are active and in blue otherwise.*

8.3. Climbing a Ladder

The original animation is composed of 300 frames (12 seconds). We detect the space constraints associated to the feet and the hands. We only need to specify 4 template constraints: one for each foot and one for each hand. The results are shown in the accompanying video. This example is particularly useful to highlight an important issue when dealing with animations that potentially interact with the environment. Indeed, it is very difficult to edit the animation without precisely viewing when the character should have the hands or the feet planted on the ladder. Hence, our constraint detection algorithm is also of great help to the animators as it shows all the important instants of the animation at once. As a result, future positioning of a ladder is much easier.

8.4. Break Dance

The original animation is composed of 380 frames (15 seconds approximately). In this example, we show the detected space constraints associated to the hands only. We only need to specify 2 template constraints: one for each hand. The results are shown in Figure 1. Note that it is very difficult to compare the results of our algorithm with potentially *perfect* results as even animators sometimes have difficulties agreeing when manually labeling frames. However, the last three constraints may not be of interest for the animators though they represent instants in the animation when the hands are *as stationary* as during the template constraints.

8.5. Computational Cost Consideration

The detection algorithm may be divided into three parts: the displacements computation and projection, the noise pattern computation and finally, the constraint detection itself. The following table summarizes benchmarks performed on a Pentium 4, 3.2 GHz, 1Go RAM. Computational costs are for one template constraint and one object only. Our algorithm is very fast. Indeed, given a template constraint, it takes between 25.5 ms and 41 ms to label between 300 and 400 frames. Our algorithm is therefore totally suited for interactive applications as animators are able to perform several constraints detection by changing requirements (i.e. type of needed constraints, minimal duration, etc) with immediate visual feedback.

Motion (section)	8.2	8.3	8.4
Nb frames	400	300	380
Nb template constraints	2	4	2
Nb detected constraints	27	23	19
Displacements + projection	13.1	9.6	12.4
Noise pattern computation	18.1	13.7	25.7
Constraint detection	2.8	2.2	2.9
Total	34	25.5	41

Table 1: *Costs (in ms) for the examples mentioned so far.*

8.6. Validation

To validate our results, we first hand-labeled each motion. Then, for each manually-defined constraint, we created a template constraint and applied our constraint detection algorithm. Table 2 shows the minimum and the maximum number of mislabeled frames. In the vast majority of the

Motion (section)	8.2	8.3	8.4
Nb frames	400	300	380
Body part	Left toe	Left toe	left hand
Nb constraints	13	6	9
Constrained frames	186	182	96
Min/Max error	9/87	5/95	20/44

Table 2: *Minimum and the maximum number of mislabeled frames for the examples of table 1.*

cases we have tested, our algorithm performs well. Note that it is difficult to perfectly label a motion according to some manual labeling because first and last frames of constraints are often ambiguous and difficult to hand-label. In some rare cases however, our algorithm produces inaccurate results. When a template constraint is too short in duration or is defined during a part of the motion that does not contain noise, then our algorithm cannot efficiently deduce the needed noise pattern. As a result, the expected constraints may not be correctly detected. However, this problem rarely occurs because "useful" template constraints are easily identified. Moreover, in case the results are not good enough and thanks to the dynamic noise pattern estimation, adding a new template constraint usually leads to accurate results.

9. Discussion and Conclusion

This paper presents a fast method for kinematic constraint detection for motion capture data. In particular, we have introduced a robust approach to detect and reject potential outliers in the data. Given a small set of template constraints, we thus automatically compute a *dynamic noise pattern* modeling the noise in the data using cubic splines. This noise pattern is then applied to the whole animation to detect instantaneous constraints. These latter are finally merged when possible to produce a minimal and meaningful set of constraints for the animator.

Distance Between Constraints. The frame tolerance f_{tol} needs to compute a distance between constraints. This distance may only consider the information available at that time. However, it depends on the dimension of the constraint: a space or line constraint position is defined by a translation and a rotation, whilst a point constraint is only defined by a position. We may then define two distance functions: one considering the Euclidean and angular distance when working with space and line constraints and another one only considering the Euclidean distance when working with point constraints. However, this solution is far from being practical: indeed, two different distance functions imply two different frame tolerance functions. For genericity purposes, we therefore chose to only use the Euclidean distance.

Small Periodic Motions. Small periodic motions may be hardly handled by our algorithm directly. Indeed, if we consider a quick tap of the foot for example, two problems may then arise. Firstly, the constraints are to small in duration to be efficiently detected. Secondly, the duration between two successive constraints is so short that our method may tend to merge them into a single one. A simple solution to overcome these problems may be to simply slow down the motion (by diminishing the frame rate for example) before detecting the constraints.

Manual Intervention. Animators actively participate in the detection process, as some detected kinematic constraints may not be of interest for the user. The final result is thus left to the appreciation of the animator who confirms, adjusts and/or deletes constraints depending on their subjective "importance". Moreover, as we aim at working with highly noisy data, it is difficult to compare our results with manually labeled constraints. Indeed, some constraints are very difficult to clearly identify and lead to diverging opinions even among animators.

Visualization According to animators' suggestions, the visualization could be improved in several ways. We only rely on the vertices of an object to display its related space constraints. Instead of displaying point clouds, we could simply duplicate the mesh itself. Furthermore, using different constraint colors depending on their associated object could help the animators to rapidly recognize important and useful constraints depending on the task they have to do. Finally, visually numbering the constraints could also give useful insights on the important instants of the animation.

Template Constraints One key point of our method is that it computes suitable thresholds using template constraints only. They must then be chosen so that they "represent" the characteristics of the noise in the motion. "Wrong" template constraints then lead to inaccurate results. However, while our LMedS method detects and rejects outliers in the raw data, it also detects wrong labeling from the user. As a result, even if a template constraint is not perfect in duration, our algorithm implicitly corrects it offering the animators more flexibility and more robustness during manual labeling.

We have shown in this paper that our method is fast, generic and robust even when the original motion is extremely noisy. Our constraint detection method has been used in particular to enforce environmental constraints using our motion editing framework detailed in [LB04]. Finally, we believe that this method can be successfully applied in many other contexts such as database indexation or motion blending for example.

Acknowledgments

The authors would like to thank Mirouille, Nicolas Elsig and Thierry Michellod for the characters, the scenes and the video editing. We also thank Helena Grillon for careful proofreading. Many thanks also to the reviewers for their very helpful comments. This research was supported by the Swiss National Science Foundation under the grant 200020-109989. The Maya licenses have been granted by Alias through their research donation program. The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

References

[BB98] Bindiganavale R., Badler N. I.: Motion Abstraction and Mapping with Spatial Constraints. *Lecture Notes in Computer Science 1537* (1998), 70–83.

[Cha30] Chasles M.: Note sur les propriétés générales du système de deux corps semblables entr'eux. In *Bulletin des Sciences Mathématiques, astronomiques, physiques et chimiques* (1830), pp. 321–326.

[IAF06] Ikemoto L., Arikan O., Forsyth. D. A.: Knowing When to Put Your Foot Down. In *Proceedings of ACM Symposium on Interactive 3D Graphics* (2006).

[KSG02] Kovar L., Schreiner J., Gleicher M.: Footskate Cleanup for Motion Capture Editing. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation* (2002), pp. 97–104.

[LB04] Le Callennec B., Boulic R.: Interactive Motion Deformation with Prioritized Constraints. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (aug 2004), pp. 163–171.

[LCR*02] Lee J., Chai J., Reitsma P. S. A., Hodgins J. K., Pollard N. S.: Interactive Contol of Avatars Animated With Human Motion Data. In *Proceedings of ACM SIGGRAPH, Annual Conference Series* (2002), pp. 491–500.

[LP02] Liu C. K., Popovic Z.: Synthesis of Complex Dynamic Character Motion from simple animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series* (2002), pp. 408–416.

[MKMA04] Menardais S., Kulpa R., Multon F., Arnaldi B.: Synchronization of interactively adapted motions. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (aug 2004).

[MM99] Moore D. S., McCabe G. P.: *Introduction to the Practice of Statistics, 3rd ed. New York*. W. H. Freeman, 1999.

[PTVF92] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes in C, 2nd. edition*. Cambridge University Press, 1992.

[RL87] Rousseeuw P. J., Leroy A. M.: *Robust Regression and Outlier Detection*. John Wiley & Sons, 1987.

Appendix A: Global versus Local Displacement Matrix Formulations

Let us consider the case of a space constraint. All the sigmas in matrix $\mathbf{\Sigma}$ are thus small enough to be considered as zero. The particular solution $\mathbf{p_{particular}}$ corresponding to Equation (6) is then:

$$\mathbf{p_{particular}} = [\mathbf{0},\mathbf{0},\mathbf{0}]^T$$

The residual error is $\epsilon = ||\mathbf{A}\mathbf{p_{particular}} + \mathbf{t_i}|| = \mathbf{t_i}$.

Let us consider $\mathbf{W}_i$ the matrix transforming, at frame i, a point $\mathbf{p}$ expressed in the O local coordinate system to $\mathbf{x_i}$ expressed in the world coordinate system. $\mathbf{W_i}$ can be decomposed as:

$$\mathbf{W_i} = \begin{bmatrix} \mathbf{R_{W_i}} & \mathbf{t_{W_i}} \\ \mathbf{0_3} & 1 \end{bmatrix} \tag{9}$$

In the next sections, we demonstrate that the residual error is modified by translation when using the global formulation. Conversely, the residual error is independent of the animation global position when using the local formulation.

Global Formulation of the Residual Error

Using the global formulation, the displacement matrix $\mathbf{D_i}$ is expressed as:

$$\mathbf{D_i} = \mathbf{W_{i+1}}\mathbf{W_i}^{-1} \tag{10}$$

Using Equation (9), we can rewrite $\mathbf{D_i}$ as:

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

The residual error is then defined as:

$$\epsilon = \mathbf{t_i} = \mathbf{t_{W_{i+1}}} - \mathbf{R_{W_{i+1}}}\mathbf{R_{W_i}}^T\mathbf{t_{W_i}} \tag{12}$$

Adding a translation component $\mathbf{t_\Delta}$ to $\mathbf{W_i}$ and $\mathbf{W_{i+1}}$ gives:

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

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

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

Figure 8 clearly shows that the more the character moves away from the origin, the higher the residual error.

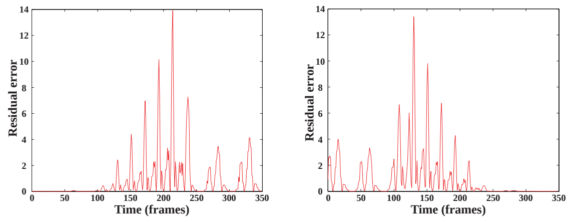

Figure 8: *Residual errors using the global formulation. The character starts walking, then runs and finally walks. **Left:** The character initial position is* $[\mathbf{0},\mathbf{0},\mathbf{0}]$. *Its final position is* $[\mathbf{17},\mathbf{0},\mathbf{0}]$. ***Right:*** *The character initial position is* $[\mathbf{-17},\mathbf{0},\mathbf{0}]$. *Its final position is* $[\mathbf{0},\mathbf{0},\mathbf{0}]$.

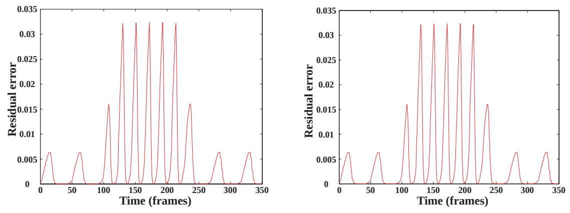

Figure 9: *Residual errors using the local formulation. **Left:** The character initial position is* $[\mathbf{0},\mathbf{0},\mathbf{0}]$. *Its final position is* $[\mathbf{17},\mathbf{0},\mathbf{0}]$. ***Right:*** *The character initial position is* $[\mathbf{-17},\mathbf{0},\mathbf{0}]$. *Its final position is* $[\mathbf{0},\mathbf{0},\mathbf{0}]$.

Local Formulation of the Residual Error

Using the local formulation, the displacement matrix $\mathbf{D_i}$ is expressed as:

$$\mathbf{D_i} = \mathbf{W_i}^{-1}\mathbf{W_{i+1}} \tag{13}$$

Using Equation (9), we can rewrite $\mathbf{D_i}$ as:

$$\mathbf{D_i} = \begin{bmatrix} \mathbf{R_{W_i}}^T\mathbf{R_{W_{i+1}}} & \mathbf{R_{W_i}}^T(\mathbf{t_{W_{i+1}}} - \mathbf{t_{W_i}}) \\ \mathbf{0_3} & 1 \end{bmatrix} \tag{14}$$

The residual error is then defined as:

$$\epsilon = \mathbf{t_i} = \mathbf{R_{W_i}}^T(\mathbf{t_{W_{i+1}}} - \mathbf{t_{W_i}}) \tag{15}$$

Similarly, we add a translation component $\mathbf{t_\Delta}$ to $\mathbf{W_i}$ and $\mathbf{W_{i+1}}$. If $||\mathbf{t_\Delta}||$ tends to infinity, we then have:

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

Figure 9 shows that our formulation is independent from the global position of the animation in the scene: for the same walking motion, we obtain same residual error.

Appendix B: Construction of f_{tol}

Given some distance $d(kc_1, kc_2)$ between two constraints kc_1 and kc_2, we construct a function $f_{tol} : \mathbb{R} \longrightarrow \mathbb{N}$ returning a number of frames with respect to the chosen distance $d(kc_1, kc_2)$. We require f_{tol} to respect the following conditions:

$$f_{tol}(d) = \begin{cases} F_{max} & \text{if } d = 0 \\ 0 & \text{if } d > d_{max} \end{cases} \tag{16}$$

F_{max} is the maximum number of frames allowed between two constraints in order to consider them as temporally connected. d_{max} is the maximum acceptable distance between two constraints in order to consider them as temporally connected. We additionally want f_{tol} to severely decrease when the parameter d increases: we then consider the functions of the form $f(d) = \lfloor \alpha\ exp^{-\beta d} \rfloor$ where $\lfloor x \rfloor$ is the floor function. Using the first condition of Equation (16), we have:

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

Then, using the second condition of Equation (16), we have:

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

We then have:

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

Finally the frame tolerance function f_{tol} is defined as:

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

Figure 10 shows the frame tolerance function f_{tol}.

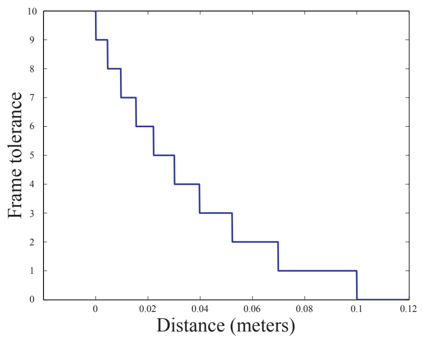

Figure 10: *Frame tolerance with* $F_{max} = 10$ *frames and* $d_{max} = 10$ *centimeters.*