This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

Motion Segmentation via Generalized Curvatures

Robert T. Arn, *Student Member, IEEE,* Pradyumna Narayana, Tegan Emerson, Bruce A. Draper, *Member, IEEE,* Michael Kirby, *Member, IEEE,* Chris Peterson, *Member, IEEE*

***Abstract*—New depth sensors, like the Microsoft Kinect, produce streams of human pose data. These discrete pose streams can be viewed as noisy samples of an underlying continuous ideal curve that describes a trajectory through high-dimensional pose space. This paper introduces a technique for generalized curvature analysis (GCA) that determines features along the trajectory which can be used to characterize change and segment motion. Tools are developed for approximating generalized curvatures at mean points along a curve in terms of the singular values of local mean-centered data balls. The features of the GCA algorithm are illustrated on both synthetic and real examples, including data collected from a Kinect II sensor. We also applied GCA to the Carnegie Mellon University Motion Capture (MoCaP) database. Given that GCA scales linearly with the length of the time series we are able to analyze large data sets without down sampling. It is demonstrated that the generalized curvature approximations can be used to segment pose streams into motions and transitions between motions. The GCA algorithm can identify 94.2% of the transitions between motions without knowing the set of possible motions in advance, even though the subjects do not stop or pause between motions.**

***Keywords*—Generalized Curvature Analysis, Local SVD, Motion Segmentation, Video Segmentation.**

I. Introduction

Recent advances in depth sensor technology have created a new type of signal to be analyzed: streams of high-dimensional pose data. The best-known examples are in the Microsoft Kinect family of devices [1] due to their popularity in the computer game industry. Other, similar devices include the Asus Xtion Pro Live [2], which also produces real-time body poses, and the LeapMotion [3] and Intel RealSense [4] which produce detailed hand poses. The Microsoft Kinect II, which is the device used in this paper, produces (x, y, z) coordinates for 25 body parts at approximately 30 frames per second. Figure 1 shows example poses extracted by the Microsoft Kinect II (left side) and Intel RealSense (right side).

Streams of body poses are usually analyzed in terms of motions, while hand poses are analyzed for gestures. In both cases, the goal is to determine *when* motions occur and *what* motions occur. This often requires segmenting pose streams into motions and classifying the motions. When the set of possible motions is known *a priori*, segmentation and classification can be solved jointly, as in [5]–[8]. However, there are applications where the motions are not known in advance. Examples include labeling tools, where the goal is to segment a stream prior to labeling, and some health care applications, where the goal is to measure the frequency, duration and magnitude of motions rather than to identify specific motions or actions. In these cases, pose streams must be segmented into sets of unknown motions.

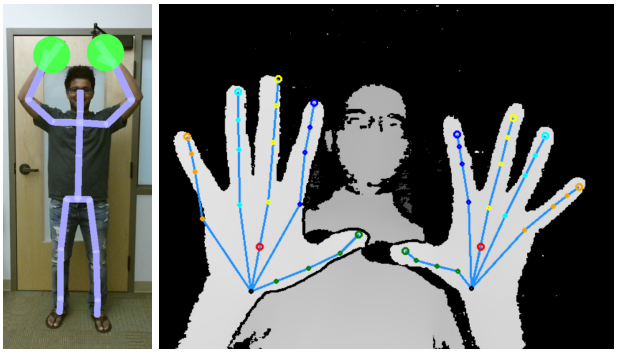

Fig. 1. Example poses. The left side shows 25 body pose points extracted by a Microsoft Kinect II. The right side shows 44 hand points (22 per hand) extracted by the Intel RealSense.

This paper presents an algorithm for segmenting unconstrained pose streams into arbitrary motions. It is based on two fundamental observations. The first observation is that human motions have three distinct stages: initialization, transport, and conclusion. The initialization and conclusion stages are relatively brief, but involve complex changes in direction. The transport stage is longer but involves comparatively smooth trajectories. The second observation is that the trajectory of a body's joints over time can be viewed as a curve in a high-dimensional space. 3D sensors like the Microsoft Kinect are therefore devices which sample points along high-dimensional curves, albeit with noise. Together, these observations suggest that pose streams can be segmented by separating the low-curvature transport phases of motions from the high-curvature transitions between motions. Moreover, this approach to motion segmentation should work even when subjects do not pause or slow down between actions, thereby combining the conclusion of one motion with the initialization of the next.

To go into more detail, the body parts tracked by a sensor can be used to define a pose space. For example, the Microsoft Kinect II tracks 25 body points in 3 dimensions. By stacking these 25 3-dimensional coordinates, every pose can be identified with a point in a 75 dimensional space. As the subject moves, their poses trace out a curve in 75 dimensions. The poses recorded by the Kinect are therefore discrete samples of a curve, lying in $\mathbb{R}^{75}$, corrupted by sensor noise.

The proposed algorithm segments pose curves into discrete motions by estimating generalized curvatures along the curve and dividing it at local curvature maxima. The technical challenge is to robustly estimate generalized curvatures from a series of noisy samples in a high dimensional space. Several traditional approaches rely on estimating local derivatives from point differences and are highly susceptible to noise. We

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

approach the problem differently with the goal of estimating the curvature using the local singular value decomposition which can be shown to provide rigorous estimates for curvature while simultaneously reducing the susceptibility to noise.

Given a curve in an n-dimensional Euclidean space, the k^{th} extrinsic curvature is a measure of the rate of separation of the curve from the $(k-1)$-dimensional osculating linear space. In a recent theoretical study, it has been shown that the values of these generalized curvatures can be expressed in terms of local singular values along the curve [9]. In addition, the ordered set of local left singular vectors correspond to the well-known Frenet frame of the curve [10]. We extend the theoretical framework developed in [9] to produce a computationally robust algorithm for estimating generalized curvatures. In this current approach the curvatures are estimated at mean values of local windows, or balls, of data rather than on the ideal theoretical curve. Again, the local singular value decomposition is center stage but the resulting formula for the generalized curvatures must be modified. The result is a robust local-SVD based algorithm for approximating generalized curvatures along a discretely sampled noisy representation of an ideal curve. At the heart of the computations is the need to automatically adapt the window size to determine the data in the local ball. We propose two methods for adjusting the local data ball independently for each generalized curvature. The result is a ***G***eneralized ***C***urvature ***A***nalysis (GCA).

In summary, the main contributions of this paper are the development of an algorithm for adaptive curvature approximation, and the application of the algorithm to motion segmentation of pose trajectories. The algorithm has two major components. First, we formulate a curvature computation to estimate the curvature at a local center of sampled data. This involves approximating points on a presumed underlying curve with the centers of data within local balls of adaptive size. The sequence of points associated with the centroids of these local balls provides a proxy for the underlying curve. The second component of the algorithm is an adaptive window used to determine the extent of the local ball for the computation of the centers and the computation of curvature.

The result is an SVD-based algorithm for estimating curvatures from noisy data sampled in high-dimensions. The algorithm provides a new tool for analyzing curvatures from noisy time data, such as data collected from the Microsoft Kinect II. In sample applications, the algorithm exhibits sufficient robustness to noise as to allow segmentation of pose curves.

This paper is organized as follows: Section II summarizes related work and motivates the need for GCA. Section III introduces generalized curvature formulae for a sampled mean curve in terms of the local singular value decomposition; the details of the derivation may be found in the appendix. Section IV presents two techniques for automatically adapting the local data region. Section V evaluates the ability of GCA to estimate generalized curvatures on a synthetic curve for which ground truth curvatures are known. Section VI evaluates the ability of GCA to segment human motions in streams of Kinect II and Motion Capture data. Section VII concludes and provides instructions for downloading the algorithm (written in Python) and data.

II. Related Work

We briefly discuss human motion below, with the goal of clarifying our terminology. We then discuss different approaches to temporal segmentation, the role of differentiable curves in computer vision, and curvature estimation techniques from computational geometry.

A. Segmenting Motions

Many applications require segmenting motions and labeling the resulting segments. When the set of motions (or actions) is finite and known in advance, segmentation and labeling can be solved jointly, as in [5]–[8]. When the motions are not known in advance, the problem gets harder. There are two basic approaches to open-set motion segmentation. One is to look for minima in kinetic energy [11], [12]. The other is to segment the motion into fragments that cluster, in order to detect repeated motions [13]–[18]. Along these lines, Zhou *et al.* 2008 pose temporal clustering as an energy minimization problem and use dynamic time warping (DTW) as the distance measure [19]. Zhou *et al.* 2013 extend this work to hierarchical decompositions at multiple scales [20]. Kruger *et al.* propose unsupervised segmentation based on self-similar structures using neighborhood graphs [21], [22]. Koppula *et al.* use the sum of Euclidean distances between skeleton joints as edge weights for graph-based segmentation [23].

Segmentation techniques that rely on minima in kinetic energy assume that subjects pause or slow down between motions. This may not always be true, particularly for motions that combine to form familiar actions. Techniques that rely on clustering work well for repeated, rhythmic motions within actions such as steps within walking, but may not work well for less cyclical motions. The approach proposed here uses curvature to segment motions, even in situations where there are no pauses between non-repeated motions.

B. Differentiable Curves in Computer Vision

Differentiable curves have a long history in computer vision. Generally, differentiable curves in $\mathbb{R}^3$ are described in terms of their curvatures in a local frame of reference, called the Frenet frame, defined by the tangent, normal, and binormal vectors. Koenderink analyzed Frenet frames in the context of computer vision [24], and Faugeras further developed this analysis [25]. Zucker gives the most thorough explication of the role of differential geometry in computer vision [26], including the differential geometric description of curves in more than 3 dimensions. Generalized cylinders were a popular representation defined in terms of smooth differentiable curves, for example Pegna [27], Bronsvoort & Klok [28], and Zerroug & Navatia [29]. Wagner & Ravani described rational generalized cylinder models as Frenet curves [30]. Differentiable curves have also been used to describe the motion of cameras through stationary environments [31], the motion of tools as seen from stationary cameras [32], and the motion of moving cameras in complex domains [33]. More recently, Kim *et al.* analyzed space-time curves in terms of curvature and torsion [34]. Differentiable curves have also been

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

used to compare trajectories. Chern [35] and Qu [36] solved kinematics using Frenet frames. Wang *et al.* [37] and Vochten *et al.* [38] propose invariant trajectory descriptors based on Frenet-Serret formulae.

C. Equations for Curvature and Torsion

Generalized curvatures for curves residing in n-dimensional space have not attracted a lot of attention, primarily because they are hard to compute accurately. It is more usual to examine curves in three dimensions, where the quantities of curvature κ and torsion τ are defined. Curvature is a measure of the rate of change of the tangent vector along a curve while torsion measures the rate at which the curve is leaving the *osculating plane*. Together, they fully characterize the shape of any curve in 3-dimensions. The formula for computing curvature is given by

$$\kappa(s) = \frac{\|\dot{c} \times \ddot{c}\|}{\|\dot{c}\|} \tag{1}$$

where $c(s)$ is the parametrically defined curve. For torsion we have the equation

$$\tau(s) = \frac{(\dot{c} \times \ddot{c}) \cdot \dddot{c}}{\|\dot{c} \times \ddot{c}\|^2} \tag{2}$$

One can extend these ideas to determine formulae for generalized curvatures in n-space. However, the k^{th} generalized curvature κ_k depends on the $k+1^{th}$ derivative of the curve making this approach impractical for computation in the presence of noise.

Álvarez-Vizoso *et al.* [9] provide a theoretical starting point for the practical computation of generalized curvatures for curves in n-dimensions. They derived explicit formulae for generalized curvatures for points on the curve in terms of local singular values in the setting of noise-free samples. In addition, they showed that the local left singular vectors are equivalent to the Frenet frame. In this paper we extend this approach to the practical problem of estimating generalized curvatures from noisy sampled data. We reformulate the problem to estimate curvature at mean values of the curve. The mean curve approach employed here is a non-iterative method which generates a proxy for the presumed "central curve" through the data, and has similar goals to the principal curve algorithm [39]. We note that Solis also proposed that curvature could be estimated from local singular values [40] but did not provide explicit formulae for their computation.

III. Curvature from Eigenvalues

The curvature of a curve $C \subset \mathbb{R}^n$ at a point P on C may be determined by finding the circle of best fit to C in an infinitesimally small neighborhood of P. This approximation is known in differential geometry as the *osculating circle* [10]. It resides in the *osculating plane* of best fit to the curve at P. The task to be addressed in what follows is how to convert Equation (5) into a robust algorithm for computing generalized curvatures from data, possibly in the presence of noise. Of key importance is determining the local region for computing the eigenvalues at each point on the curve.

A. The Generalized Curvature Formulae

Given data on a curve in a local ball of radius ϵ it is shown in the appendix that the curvature at the mean value of the points in the ball may be computed using

$$\kappa_1^2 = 5 \lim_{\epsilon \to 0} \frac{\lambda_2}{\lambda_1^2}. \tag{3}$$

and the second generalized curvature (i.e., the torsion if $n = 3$) by

$$\kappa_2^2 = \frac{35}{3} \lim_{\epsilon \to 0} \frac{\lambda_3}{\lambda_1 \lambda_2} \tag{4}$$

where the λ_i are the eigenvalues of the covariance matrix of the data in the local ϵ ball, or equivalently, the squares of the singular values; see [41] for additional details.

Further, the k^{th} generalized curvature for a curve in n-dimensions is given by

$$\kappa_k^2 = \frac{(2k+1)(2k+3)}{3} \lim_{\epsilon \to 0} \frac{\lambda_{k+1}}{\lambda_1 \lambda_k} \tag{5}$$

for $k = 1, \ldots, n-1$.

We note that Equation (5) determines generalized curvatures at *mean* values on the curve. In our previous work [9] it is assumed that the curvature was being computed at a point exactly on the curve. In that case we proved

$$\hat{\kappa}_k^2 = \frac{k+1}{k+1+(-1)^{k+1}} \frac{4(k+1)^2 - 1}{3} \lim_{\epsilon \to 0} \frac{\lambda_{k+1}}{\lambda_1 \lambda_k} \tag{6}$$

for the generalized curvatures $k = 1, \ldots, n-1$.[1] Note that the basic form of the equation does not change, i.e., ratios of eigenvalues, from [9], but the leading coefficient does.

In the appendix we derive these equations by first considering curves of *constant* curvature in two and three dimensions, i.e., the circle and the helix. Considering these canonical curves is sufficient for estimating curvature at a point on an arbitrary curve since we are computing the generalized curvatures in the limit as $\epsilon \to 0$.

In the next section we outline how to adapt Equation (5) to a practical algorithm for determining the curvature of a curve lying in an n-dimensional space.

B. The Curvature Algorithm

Equation (5) provides a formula for the generalized curvatures κ_k from the view of the local mean of data sampled from the curve. The formulae for the generalized curvatures is derived in the setting of a shrinking ϵ-ball about a point on the curve identified by t_0, i.e., $x(t_0)$ where the domain of the curve is a continuous variable. In practice, however, we can't actually achieve $\epsilon \to 0$ given data sampled from the curve at discrete times. This is not a major obstacle though given that the formula for curvature is in terms of the eigenvalues of the covariance matrix which may be readily computed from sampled data using principal component analysis, or the related singular value decomposition.

[1] We use $\hat{\kappa}_k$ to emphasize the difference from the κ_k computed above in Equation (5).

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

We assume that we have M samples of the curve $x(t)$ collected at times $t_i, i = 1, \ldots, M$[2] and denote the indices of the first and last points in the i^{th} ball (or time-window) by l_i and r_i, respectively, corresponding to times t_{l_i} and t_{r_i}. At the sample point $x(t_i)$ we compute the curvature about the local mean of the points at time t_i, i.e., $\overline{x}(t_i)$, and *not* at the point $x(t_i)$. We define the local mean-centered data matrix as

$$X(t_i) = [x(t_{l_i}) - \bar{x}(t_i)| \ldots |x(t_i) - \bar{x}(t_i)| \ldots |x(t_{r_i}) - \bar{x}(t_i)].$$

In the next section we discuss an automated algorithm for locating the boundary indices in the local ball. Once l_i and r_i have been determined, we may compute the singular values σ_i of the matrix $X(t_i)$, or the eigenvalues of the matrix

$$C(t_i) = X(t_i)X(t_i)^T/M$$

The formula for the k^{th} generalized curvature κ_k at time t_i requires eigenvalues λ_1, λ_k and λ_{k+1}. The singular values of the SVD of X are related to the eigenvalues of C via $\lambda_i = \sigma_i^2$.

IV. Adapting Data Windows

We have outlined a procedure above for approximating the curvature of a curve that requires the computation of the singular values associated with data in a local ball. In practice, we anticipate that the optimal size of a discrete data ball will depend on the local curvatures, relative to the level of noise. If the rate of change of the curvature is small near a point $x(t_i)$, then we expect to be able to include more data by extending the radius of the ball, thereby compensating for the noise. In contrast, if the curvature is changing rapidly, the size of the ball needs to be small to prevent data away from the point from corrupting the estimate of the local curvature. Experiments suggest that adapting the size of the data ball to reflect curvature variance and noise improves the performance of GCA. Hence, instead of using a fixed window size, we develop automated methods for computing a window size for each point on the curve.

A. An approach based on projections

GCA allows for the possibility that the number of points included in the ball may be different to the left or right of the center, given that curvature may vary non-uniformly in any ball. This is potentially useful when the data is not equally sampled in time, or if the velocity of the curve changes. Hence, our goal is to determine the integer values l_i^* and r_i^* which will be used in the formation of a local window $[x(t_{l_i^*})| \ldots |x(t_i)| \ldots |x(t_{r_i^*})]$ to be used in the local SVD after mean subtraction.

We determine the optimal window-size for computing each κ_k by projecting the mean-centered data on the curve onto the first k dimensions in the local coordinate system $E(t_i) = [e_1(t_i) \,|\, \ldots \,|\, e_M(t_i)]$ provided by the singular value decomposition of the data window $X(t_i)$, i.e.,

$$X(t_i) = E(t_i)\Sigma(t_i)(F(t_i))^T$$

[2]Note that, in general, there is no requirement that these times t_i be equispaced.

We grow the window size around the point $x(t_i)$ and compute the basis $E_k(t_i) = [e_1(t_i) \,|\, \ldots \,|\, e_k(t_i)]$. The span of this subspace is the best k-dimensional linear approximation to the data in the ball. The projector onto this space is then $\mathbb{P} = E_k E_k^T$ and we compute the ratio $\|\tilde{x}\|/\|x\|$ where we employ the notation $\tilde{x} = (I - \mathbb{P})x$ to represent the complementary projection of a vector. We stipulate that if $\|\tilde{x}\|/\|x\| \geq \gamma$ then the point on the curve is deviating too much from the linear space and the local region should stop growing.

Algorithm 1: The Generalized Curvature Analysis (GCA) algorithm with Singular Vector Window Estimation. In practice, we set w = 30, r = 20 and p = 25.

```
   Input: k_list                          ▷ list of k curvatures to calculate
   Output: GC
 1 N = frames
 2 foreach point ∈ [0, N) do
 3     singularVectorList = []
 4     foreach grow ∈ [0, w) do
 5         start = point-grow                          ▷ break if start<0
 6         end = point+grow                            ▷ break if end>=N
 7         window = data[start:end]
 8         uList = []
 9         foreach random ∈ [0, r) do
10             sampledWindow = Sample p points from
                window with replacement
11             Mean subtract sampledWindow
12             U,S,V = svd(sampledWindow)
13             uList.append(U[k_list])
14         end
15         Align singular vectors in same direction
16         U = mean(uList)
17         singularVectorList.append(U)
18     end
19     foreach k ∈ k_list do
20         angleList = dot product of adjacent vectors in
            singularVectorList[k]
21         Smooth angleList
22         optimalIndex = index(max(angleList))
23         optimalWindow = optimalIndex*2+3
24         start = point-optimalWindow/2
25         end = point +optimalWindow/2
26         window = data[start:end]
27         Mean subtract window
28         [U,S,V] = svd(window)
29         GC[point,k] =
               constant * sqrt(optimalWindow) * S[k] / (S[k-1] * S[0])
30     end
31 end
```

For a given sampled point on the curve $x(t_i)$, candidate boundary points for the i^{th} interval are the times t_l and t_r. For each t_i, define the line segments

$$p_l = x(t_l) - \bar{x}(t_i), \quad p_r = x(t_r) - \bar{x}(t_i)$$

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

where optimal values for l and r are to be determined.

Now for each curvature κ_k we construct the projection matrix onto the best k-dimensional subspace. In practice, the optimal indices l^*, r^* may be found by solving the optimization problems

$$l^* = \arg\min_{l<i} \left| \frac{\|\tilde{p}_l\|}{\|\mathbb{P}p_l\|} - \gamma \right|, \quad r^* = \arg\min_{l>i} \left| \frac{\|\tilde{p}_l\|}{\|\mathbb{P}p_l\|} - \gamma \right|.$$

Here $\gamma > 0$ is an *ad hoc* cutoff parameter which ensures that the set of points $\{x(t_l*), \ldots, x(t_r*)\}$ represents a local region of the curve x. Through extensive empirical tests, we have found that values of γ in the range $0.05 \leq \gamma \leq 0.5$ produce robust bounding intervals. Once we have the appropriate window size for each point along the curve the eigenvalues are recomputed and the curvatures at time t_i, i.e. the $\kappa_k(t_i)$, are estimated.

B. An approach based on singular vectors

The projection method described above captures the geometric intuition that is useful to define a local region, but experimentation has shown that its performance can be impacted by noise when there is limited data. The second method we propose exploits information associated with the left singular vectors of the data ball and is robust to higher levels of noise.

Here we consider the effect of a growing ball on the direction of the left singular vectors of the data and use this information to determine window size. For the data point $x(t_i)$ sampled on the curve we form the sequence of (mean-subtracted) data matrices

$$B^{(1)} = [x(t_{i-1}) \,|\, x(t_i) \,|\, x(t_{i+1})]$$

$$B^{(2)} = [x(t_{i-2}) \,|\, x(t_{i-1}) \,|\, x(t_i) \,|\, x(t_{i-1}) \,|\, x(t_{i+2})]$$

and in general

$$B^{(j)} = [x(t_{i-j}) \,|\, \ldots \,|\, x(t_i) \,|\, \ldots \,|\, x(t_{i+j})] \tag{7}$$

where the number of points in the j^{th} data matrix is $2j + 1$. For each data matrix we compute the left singular vectors. We denote $u_j^{(k)}$ to be the k^{th} left singular vector of data matrix j (associated with point i which is not explicitly represented). As the ball grows we compute the measure of change of the k^{th} singular vector direction via the dot product

$$d_{j,j+1}^{(k)} = u_j^{(k)} \cdot u_{j+1}^{(k)} \tag{8}$$

When this value $d_{j,j+1}^{(k)}$ peaks as a function of j we conclude that the optimal window size has been reached, i.e.,

$$j_k^* = \arg\max_j d_{j,j+1}^{(k)} \tag{9}$$

determines the optimal index j_k^* for each sampled point on the curve, and hence determines the best size of the window for estimating the k^{th} generalized curvature κ_k. For values $j < j^*$ there is insufficient data relative to the level of noise, and the vectors approximating the Frenet frame, i.e., the left singular vectors, are still changing direction. For $j > j^*$ the ball is becoming too large resulting in a change in direction of the frame vectors. For $j = j^*$ we obtain maximal alignment of the Frenet frame over the sequence of data matrices.

In practice, adding points to the data matrix has diminishing impact on the direction of the singular vectors because they represent a decreasing fraction of the total variance of the data. To overcome this problem we sample the data in a given window with replacement creating a fixed size data matrix, regardless of the value of j. In practice, we take 25 samples per window and then compute the left singular vectors. This process is repeated 20 times and the resulting singular vectors are averaged. This approach, while ad hoc, uses random sampling to produce stable directions for the singular vectors in the presence of noise.

C. Complexity Analysis

The Generalized Curvature Analysis (GCA) algorithm described in Section III with the adaptive window size method presented in Section IV-B is shown in Algorithm 1. The complexity of computing the first k generalized curvatures for n points is $O(nwr(d^2p + p^3))$, where $2w + 1$ is the maximum window size, d is the dimensionality of the points, r is the number of times each data window is resampled, and p is the number of points randomly selected when resampling windows. In practice, we treat w, r and p as constants, and set $w = 30$, $r = 20$, and $p = 25$, values we have never needed to change. When w, r and p are treated as constants, the data complexity is $O(nd^2)$, i.e. squared in the number of feature dimensions, but linear in the number of points to be computed.

The complexity of calculating k curvatures at a single point $x(t_j)$ is $O(wr(d^2p + p^3))$. It is dominated by the body of the foreach loop that begins on line 4 of Algorithm 1 and ends on line 18. For every point $x(t_j)$, we generate the sequence of data matrices $B^{(1)}$ through $B^{(w)}$, as described in Equation 7. All of these matrices are the same size, namely $d \times p$, since every column is a d dimensional point, and we sample p points (with replacement) every time we resample a window. According to [42], computing the SVD of a $d \times p$ matrix is $O(d^2p + p^3)$. We repeat this r times, so the complexity of computing k curvatures at $x(t_j)$ is $O(wr(d^2p + p^3))$. If w, r and p are treated as constants, this simplifies to $O(d^2)$.

Note that the complexity of GCA is dominated by the cost of the SVDs in the search for the best window size, not the calculation of the curvatures themselves, and the SVDs do not have to be recalculated for each value of k. The foreach loop code from lines 19 to 30 in Algorithm 1 is repeated k times, but the code in this loop is independent of all the other variables, and since by definition $k < d$, the overall complexity is $O(nd^2)$.

In one example, we computed the first five generalized curvatures for 4579 points in 90-dimensions points from the CMU MoCaP data set (from the first MoCap video; see below). Using the parameters $w = 30$, $r = 20$ and $p = 25$ the calculation took 913 seconds ($\approx$ 5 points per second) on a 4-core, 2.2GHz machine, running in Python with no parallelization or other code optimizations.

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

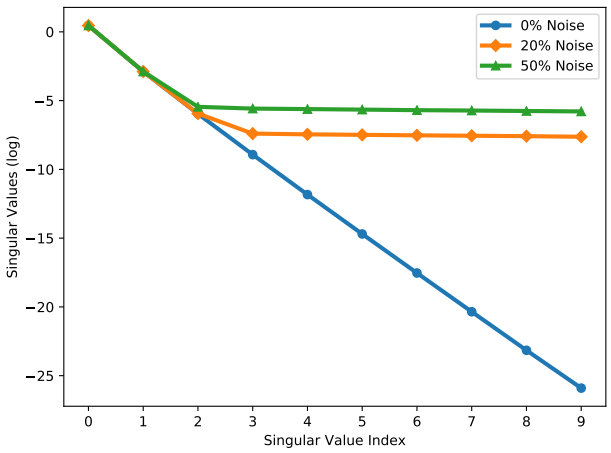

Fig. 2. Mean singular values averaged over 1000 windows for the 10 dimensional helix used in Table I. The blue curve connects singular values computed with no noise, while the orange curve shows the singular values with 20% data noise and the green curve shows the singular values with 50% data noise. Since the kth curvature depends on the (k+1)th eigenvalue, this explains the breakdowns in Table I.

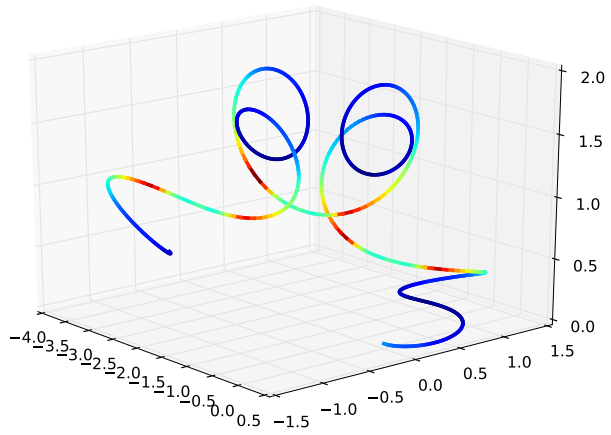

Fig. 3. A synthetically generated 3D curve using the curvature and torsion given in Equation 11. The curve is sampled at 500 points from $s = 0$ to $s = 30$. Color is used to encode velocity. The dark blue parts of the curve are where the sampled points are closest together, while the red parts of the curve show where they are farthest apart.

V. Synthetic Experiments

Sections III and IV present a novel technique for estimating generalized curvatures of high-dimensional curves, given a sequence of noisy, discrete samples. This technique is captured as an algorithm in Algorithm 1. In this section we evaluate GCA (Algorithm 1) on noisy data samples drawn from synthetic curves, where the goal is to determine how well GCA estimates generalized curvature. The next section will then come full circle and test GCA's ability to segment human motions in data from depth sensors.

A. Constant Curvature

We begin by evaluating GCA on synthetic curves for which the true generalized curvatures are known, and where we can control the amount of added noise. The first question we want to answer is how many generalized curvatures can be estimated before numerical stability issues make the results unreliable, and how does this change in the presence of noise?

To explore this question we create a 10-dimensional helix with constant curvatures, where $\kappa_i = i, i = 1, \ldots, 9$. As shown in Table I, when no noise is added to a curve with constant curvature, GCA computes the first 8 curvatures accurately and consistently. The 9th curvature κ_9, which should have a value of 9, is instead estimated at 9.32, suggesting that data sampling and numerical round-off are starting to introduce error.

Table I also shows the estimated curvatures when noise with a standard deviation equal to 20% of the distance between the sample points is added. At 20% noise, the first 3 curvatures are estimated accurately, although a little variance is introduced into the estimate of the 3rd curvature. The 4th through 9th curvature estimates, however, are now grossly in error. At 50% noise, the breakdown point is earlier. The first 2 curvatures are estimated accurately, but the 3rd is in error.

What explains this behavior? When noise is added to a curve, it can be shown that the eigenvectors are unchanged, but the eigenvalues are shifted upwards by the variance of the noise α^2, i.e.,

$$\hat{\lambda}_i = \lambda_i + \alpha^2 \tag{10}$$

(see [41] for a derivation). Figure 2 shows the eigenvalues for the helix at different levels of noise. The blue line represents the eigenvalues when no noise is added, resulting in 9 roughly accurate, generalized curvature estimates. The orange line shows the eigenvalues when 20% noise is added. The first 3 eigenvalues are accurate, which explains why the first 2 curvature estimates in Table I are accurate. The 4th eigenvalue is somewhat off, explaining the small error in the estimate of the 3rd curvature. All the eigenvalues after that are dominated by the noise, however, explaining why the curvatures above 3 are gross errors. The green line shows the eigenvalues at 50% noise. In this case, the first 3 eigenvalues are accurate, so only the 1st and 2nd curvatures are reliable. All higher generalized curvature estimates are dominated by the noise.

B. Synthetic Data Results

The helix in the previous section had constant curvatures. To look at the impact of changes in curvature, we return to the 3D synthetic curve shown in Figure 3, whose 1st and 2nd curvatures (κ_1 and κ_2) are defined by

$$\kappa_1(s) = \sin(s) + 2, \quad \kappa_2(s) = \frac{1}{\sin^2(s) + 9\cos^2(s) + 0.1} \tag{11}$$

The associated curve in $\mathbb{R}^3$ is shown in Figure 3.

For the numerical experiment we integrated the system using Runge-Kutta-Fehlberg and a local truncation error of $O(1e^{-10})$ which produced 500 unevenly sampled points in the range from

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE I. ESTIMATED CURVATURES FOR 10-DIMENSIONAL HELIX SUCH THAT THE TRUE iTH-CURVATURE IS i. CURVATURES ARE ESTIMATED FOR EVERY POINT ON THE HELIX, AND THEN AVERAGED (SINCE THE TRUE CURVATURES ARE CONSTANT). WITHOUT NOISE, THE FIRST 8 CURVATURES ARE HIGHLY ACCURATE, BUT NUMERICAL STABILTY EFFECTS BEGIN TO INTRODUCE NOISE AT THE 9TH CURVATURE. WITH 20% NOISE, THE FIRST 3 CURVATURES ARE ACCURATE. AT 50% NOISE, ONLY THE FIRST TWO ARE ACCURATE.

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
True	1	2	3	4	5	6	7	8	9
0% noise	1.00 ± 0.00	1.99 ± 0.02	2.99 ± 0.01	3.98 ± 0.02	4.98 ± 0.02	5.97 ± 0.02	6.97 ± 0.02	7.95 ± 0.02	9.33 ± 0.03
20% noise	0.99 ± 0.00	1.97 ± 0.01	3.06 ± 0.03	10.63 ± 0.22	41.82 ± 0.70	49.94 ± 0.72	57.51 ± 0.54	64.08 ± 1.10	71.14 ± 0.64
50% noise	0.99 ± 0.00	2.00 ± 0.01	5.81 ± 0.15	32.98 ± 1.04	42.75 ± 0.61	50.19 ± 0.68	57.62 ± 0.61	64.63 ± 0.55	71.11 ± 1.08

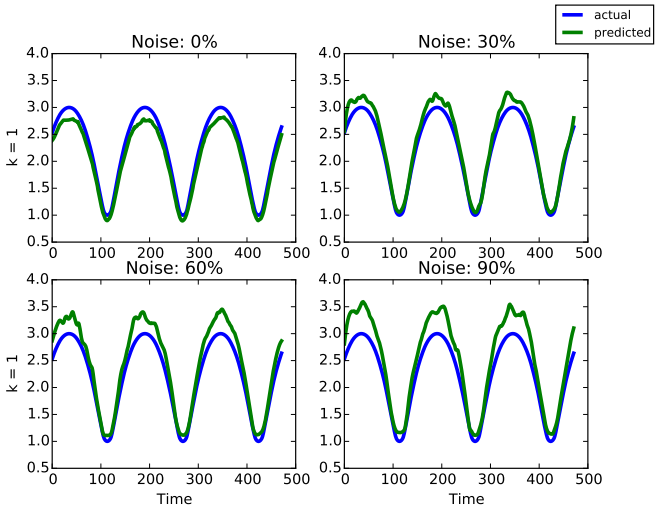

Fig. 4. The true and approximated first curvatures (κ_1) for the synthetic data curve given in Equation 11 and Figure 3. The true curvatures is shown in blue, while the estimated curvatures are shown in green. The four subplots corresponds to varied amounts of uniform noise added. Uniform noise added is at 0%,30%,60%,90% of the mean distance between points.

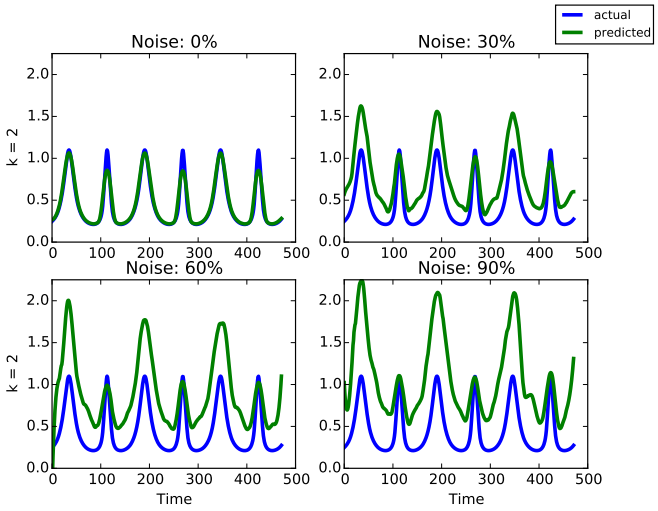

Fig. 5. The true and approximated second curvatures (κ_2) for the synthetic data curve given in Equation 11 and Figure 3. As in Figure 4, the true second curvature is shown in blue, and the estimated in green. The four subplots corresponds to varied amounts of uniform noise added. Uniform noise is added of 0%,30%,60%,90% of mean distance between points.

$s = 0$ to $s = 30$. (Note that in our example the sampling is not uniform due to the adaptive step size intrinsic to the Runge-Kutte-Fehlberg algorithm.) The data was subsequently corrupted by adding varying levels of noise. We corrupted the data samples by adding 3D uniformly distributed noise over a fixed range, where the range is expressed as a percent of the average distance between consecutive data samples. We estimate curvatures for four levels of added noise: 0%, 30%, 60% and 90%.

Figure 4 shows the true 1st curvature (κ_1) in blue and the estimated 1st curvature in green for all four noise levels. With 0% noise, the difference between the true and estimated first curvature is small, as shown in the upper left quadrant of Figure 4. The small amount of error that does appear is small, and occurs where the curvature is changing rapidly and the data samples are spread out (see Figure 3). Because these errors are the result of having to take too large an epsilon ball relative to the rate of change of the true curvature, they can lead to underestimates.

The upper right quadrant of Figure 4 shows the true and estimated κ_1 values when 30% noise is added. At 30% noise, the estimate κ_1 values still estimate the true κ_1 values very well, but there are some changes. The noise adds a little to the first two eigenvalues as in Equation 10, leading to slight overestimates in terms of curvature. Also, we begin to see some more variance in estimates from point to point, creating a green line that wiggles a little bit. Nonetheless, the estimates remain highly accurate.

When the noise is raised to $\pm 60\%$ of the distance between the samples, the error increases, particularly where both the curvature and the rate of change in the curvature are high, and the error is always an overestimate. At $\pm 90\%$ error this trend continues, with bigger errors and higher overestimates. Even at $\pm 90\%$ error, however, the shapes of the blue and green curves are similar, suggesting that the estimates continue to reflect the true shape of the curve.

Figure 5 shows the equivalent true and estimated values for the second curvature, κ_2. When no noise is added (beyond the round-off error inherent in generating sampled data), the estimated κ_2 approximates the true κ_2 closely. This is similar to the result for κ_1, and once again errors occur only where the curvature is both high and changing rapidly, at which points κ_2 is slightly underestimated.

The other three panels of Figure 5 show the true and estimated values of κ_2 for 30%, 60% and 90% noise, respectively. As we would expect, κ_2 is more sensitive to noise than κ_1. At 30% noise, it consistently overestimates the true values of κ_2 because of Equation 10. The general shape of the curve

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

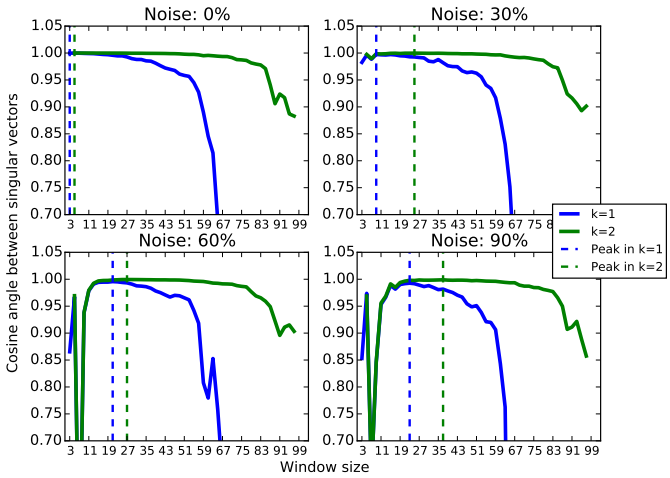

Fig. 6. Cosine of the angle between singular vectors (κ_1 and κ_2) for the synthetic data curve given in Equation 11 and Figure 3. The angles for κ_1 is shown in blue, while the angles for κ_2 are shown in green. The dashed blue line represents the peak in κ_1 and is the optimal window selected by GCA for κ_1. The dashed green line represents the peak in κ_2 and is the optimal window selected by GCA for κ_2. The four subplots corresponds to varied amounts of uniform noise added. Uniform noise added is at 0%,30%,60%,90% of the mean distance between points.

is preserved, however; the estimates of κ_2 peak where they should and have local minima where they should. The same is true at 60% and 90% noise, but more exaggerated. Note that these errors could be removed by adjusting the eigenvalues, if a model of the noise was known in advance.

Another question is how well the technique described in Section IV-B works for estimating window sizes. This technique works by maximizing the stability of the left singular eigenvectors as the size of the data window around a point grows. Figure 6 shows the change in angle in the eigenvectors as a function of window size as described in Equation 8 for a point on the same synthetic curve as above, with the same four levels of noise (0%, 30%, 60% and 90%) as above. The blue curves shows the changes in angle for the 2nd eigenvector (used to select the window size for κ_1), while the green curves show the changes in angle for the 3rd eigenvector (used to compute κ_2). The dashed vertical lines show the peaks in the curves, which are the window sizes that the algorithm will select.

Figure 6 shows that at 0% noise, the technique in Section IV-B selects the smallest possible window sizes, namely a window size of 3 for κ_1 and a window size of 5 for κ_2. This makes sense, because with no noise there is no advantage to bringing in additional data samples. In fact it hurts, because the additional samples are farther away and the curvature of the underlying curve is changing with distance. As the noise level increases, both the κ_1 and κ_2 data windows get larger to bring in more samples to smooth over the noise. The κ_2 windows are always larger than the κ_1 windows, because κ_2 is more sensitive to noise.

VI. Motion Segmentation

Human motions can be divided into three phases. The initial phase recruits muscles to overcome the inertia of a previous state, whether that state was "at rest" or involved a previous motion. Once a motion is initiated, the majority of the movement, sometimes called the transport phase is smooth. This is followed by a conclusion, where the body either stops moving or transitions to the next motion. *Motions* are different from *actions*. A motion is a single set of coordinated muscle movements, for example taking a step. An action is a goal-directed sequence of motions. Thus, while a step (or stride) is a motion, walking is an action composed of many steps.

Section V measured the ability of GCA to estimate generalized curvatures from noisy discrete samples, at least for synthetic curves. We now explore the use of generalized curvature to segment human motions. Modern depth sensors can detect human body poses, and these poses can be represented as points in a high-dimensional space. As people move, their poses trace out a high-dimensional curve. At the initialization and conclusion of any motion, multiple body parts accelerate or decelerate, creating high curvature sections of the curve, whereas the middle transport phase of the motion is relatively smooth, leading to low curvature portions of the curve. Therefore we should be able to segment pose curves into individual motions by dividing the curve at local curvature maxima.

To test GCA's ability to segment human motions, we need a depth data set of human motions with three properties: (1) state of the art signal-to-noise ratio, (2) continuous motions, without pauses in between, and (3) ground truth labels at the level of motions, not actions. Unfortunately, no standard data sets meet these criteria. In a recent survey by Firman [43], none of the depth data sets included were collected using a Kinect v2 sensor. All were collected using a Kinect v1, which is much noisier than the v2. In addition, almost all of the data sets surveyed by Firman were labeled at the level of actions, not motions.

To overcome these problems. we evaluate GCA on two different data sets. The first set of experiments are on PALKA, a data set we collected. Every video in PALKA shows a person performing three activities with no break in between. More importantly, the videos were collected using a Kinect v2 sensor and have ground truth labels at the level of motions, not actions. The second set of experiments are on the CMU MoCaP data set [19], [20]. This data set is older, but the pose data was captured using highly sensitive motion capture technology, so the signal to noise ratio is comparable to the Kinect v2 (perhaps better), and the subjects move seamlessly from one activity to another. Unfortunately, the ground truth labels are at the level of actions, not motions, which restricts our analysis. However, the MoCaP data set has been widely used, so these experiments may help readers compare GCA to other algorithms.

A. Experiments on the PALKA data set

The PALKA data set contains pose data for 234 videos of three subjects recorded with a Kinect v2 sensor. Each video

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

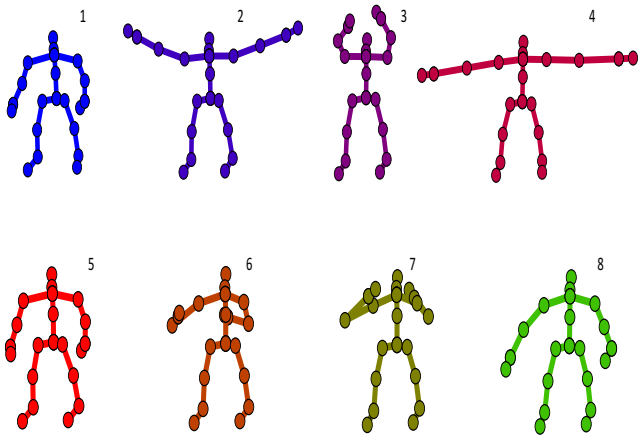

Fig. 7. Frames from Kinect v2 Skeleton data of part of one PALKA video. This video fragment shows four motions: (1) raising arms, (2) lowering arms, (3) raise hands to the eyes ("goggles") and (4) returning the hands to a neutral position. The figure shows 8 of the poses in the video. The poses are color coded to match points in Figure 8

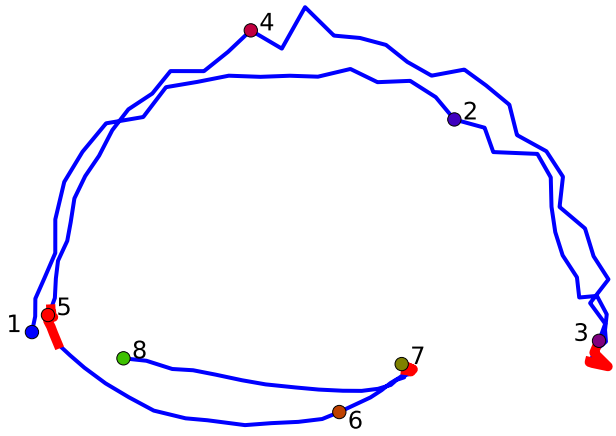

Fig. 8. Projection of the 75-dimensional pose curve from Figure 7 onto its first two eigenvectors. Low curvature sections are shown in blue and correspond to motions, while high curvature sections are shown in red and correspond to transitions between motions. The indices 1-8 are color coded to match the poses in Figure 7 in the projected trajectory. Discarding the endpoints of the video, motion transitions occur around poses 3, 5, and 7.

shows a person performing three actions (in general, more than three motions) drawn from the MSRC-12 [44] action set, with no pauses between actions. The videos are scripted to make sure that every possible transition between actions occurs the same number of times, and the videos are hand-labeled to mark the start and end of every motion within each action. Videos from the first two subjects are used as training data, while videos of the third subject are held out as a test set.

Figure 7 shows Kinect v2 skeletons extracted from a segment of one of the PALKA videos. This example contains two MSRC-12 actions, but four motions. The first MSRC-12 action, wave arms, is a combination of two motions: arm raising and arm lowering. Similarly, the second MSRC-12 action, goggles, is two motions: raising the hands to the eyes, and bringing them back down. Figure 8 shows the 75-dimensional pose curve projected onto the first two eigenvectors (computed from all the poses in the video). Low curvature sections of the curve (as estimated by GCA) are colored blue, while high curvature sections are colored red. The numbered points on the curve in Figure 8 correspond to the poses in Figure 7 with matching indices. Discarding the endpoints of the video, motion transitions occur around poses 3, 5, and 7.

Figure 9 shows the estimated 1st curvature (κ_1, shown in blue) and estimated 2nd curvature (κ_2, shown in red) for the video fragment whose poses were shown in Figure 7 and whose projected curve was shown in Figure 8. The four interior white regions in the figure correspond to the four motions (arms up, arms down, hands up, hands down) as determined by the ground truth segmentation. The shaded regions in between correspond to transitions between motions. (The exterior white regions correspond to the initial and ending standing positions.)

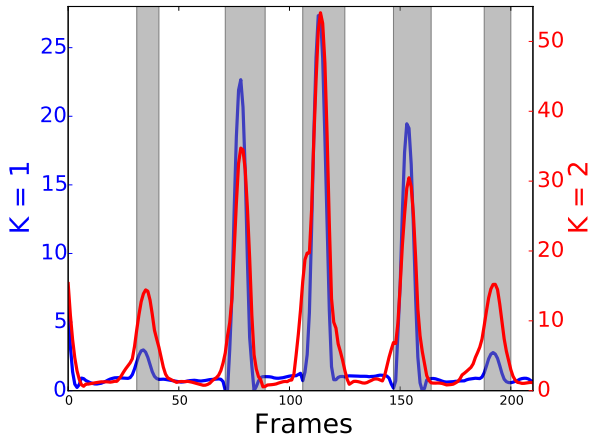

Fig. 9. Curvatures κ_1 and κ_2 computed using GCA on the PALKA video whose projection is shown in Figure 8. The white areas show motions (raising or lowering arms, kicking outward or bringing the leg back). The gray areas show transitions between motions. As hypothesized, curvatures are low during motions and high during transitions, although some transitions show up more clearly in κ_1, while others show up more clearly in κ_2.

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

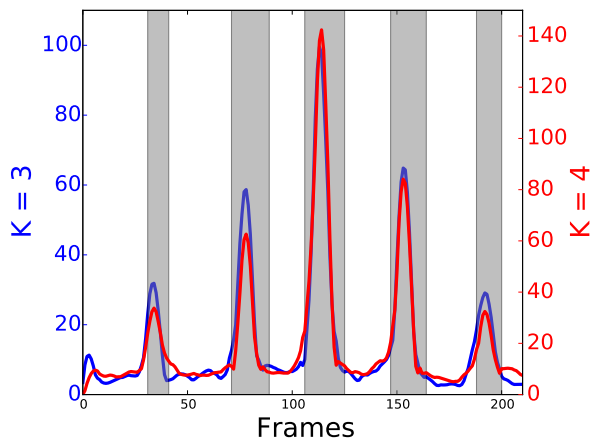

Fig. 10. Similar to Figure 9, this figure shows the third (κ_3) and fourth (κ_4) curvatures for the same video. Once again, although there are differences among the various curvatures, all tend to be high during transitions and low during motions.

Remember the premise being tested: transitions between motions should have high curvature, while the motions themselves should have low curvatures. At least in this example, the premise is true. Both κ_1 and κ_2 peak on every transition, and are relatively flat during the motions. This is true even though there is more kinetic energy during the motions, because the body parts are moving faster. The motions are smooth, however, so the curvature is low. The transitions, on the other hand, may be fast or slow, but always have higher curvature.

We are not limited to κ_1 and κ_2, however. The Kinect v2 sensor captures the 3D positions of 25 body parts, resulting in a 75 dimensional curve. In principle there are 74 curvatures (κ_1 through κ_{74}), although most would be nothing but noise. Figure 10 shows the 3rd (κ_3, in blue) and 4th (κ_4, in red) estimated curvatures. As with the first two, they peak during transitions and are small during the motions themselves. κ_3 and κ_4 are not just copies of the first two curvatures, however. Each curvature seems to respond differently to different transitions, and the correlation of κ_1 to κ_4, for example, is only 0.736. Looking at the source video, it seems as though the 1st and 5th transitions are smooth and fast, whereas the changes in direction for transitions 2 through 4 cause the subject to slow down and almost pause. Interestingly, κ_1 barely peaks on transitions 1 and 5, while the other three curvatures peak at all five transitions. Each curvature is a feature of the curve, and we are only beginning to understand what information each holds and how best to combine them.

Figures 7 through 10 are based on a single example video. Our goal is to evaluate the use of GCA to segment human motions across the PALKA data set. In particular, we use GCA (using both methods of selecting window sizes) and traditional numeric derivatives to estimate the first curvature for every frame in a video. We then use a traditional peak detection algorithm[3] to identify (1) the peaks and (2) the troughs on both sides of each peak. Frames lying between the troughs of a peak are identified as transition frames. Frames that are not part of a peak are motion frames. We then evaluate how well each curvature estimation technique segments videos into motion and transition frames.

[3]Python peakutils library. This algorithm takes two parameters: the minimum peak height and minimum distance between two peaks

TABLE II. A COMPARISON OF SEGMENTATION RESULTS ON PALKA DATASET. GCA IS BASED ON THE LOCAL SINGULAR VALUES WHILE NUMERICAL DIFFERENTIATION DIRECTLY COMPUTES CURVATURES AND TORSION NUMERICALLY FROM EQUATIONS (1) AND (2). USING GCA WITH METHODS A. AND B. FROM SECTION V. ARE DENOTED GCA V.A AND GCA V.B, RESPECTIVELY.

	Motion Frame Accuracy	Transition Frame Accuracy	Total Accuracy	Transitions Detected
Numeric Derivatives	99.9%	1.2%	63.6%	3.1%
GCA V.A.	91.4%	70.5%	83.8%	90.0%
GCA V.B.	91.3%	73.7%	84.9%	94.2%

The PALKA data set contains 156 training videos and 78 test videos. We use the training images to find the peak detection parameters for each algorithm through an exhaustive search of parameter space. We then evaluate each curvature estimation technique with its best peak detection parameters on the test images.

Table II shows the results. The first column gives the name of the curvature estimation algorithm. We tested three algorithms: traditional numeric derivatives, as described in Section II-C, GCA with projection-based dynamic windows as described in Section IV-A, and GCA with singular vector based dynamic windows, as described in Section IV-B. The second column shows the labeling accuracy for motion frames, while the third column shows the accuracy for transition frames. The fourth column is the overall frame-level accuracy, while the fifth column is the percent of transitions that contain at least one detected peak. As a point of comparison, the trivial algorithm that labels every frame as a motion frame has a total accuracy of 63.2%.

As shown in Table II, estimating curvatures using traditional numerical derivatives yields an accuracy of only 63.6%, just slightly better than the accuracy of labeling every frame as a motion frame. When GCA is used to estimate curvature, the total accuracy becomes much better. When the projection method is used to select the dynamic window size, the total accuracy is 83.8%, and 90% of all transitions contain at least one peak. The projection method is best when the data points are finely sampled in time, which is not the case with a 30fps Kinect v2 sensor. When the singular vector technique is used to select the dynamic window size, the total accuracy rises to 84.9%, and 94.2% of all transitions contain at least one peak.

The CMU MoCaP data set was released in 2008, and contains 3D position data for 41 body parts [19]. MoCap predates modern time-of-flight depth sensors; the data was collected using a traditional marker-based multi-view motion capture system. Motion capture is expensive and invasive, but the data is highly accurate with little apparent noise, and is therefore well suited to curvature analysis. It is also sampled densely in time at 120 fps.

Each video shows a subject performing a connected sequence of roughly 10 natural activities such as walking,

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

punching, drinking, and running. MoCaP therefore contains high-quality data of continuous motions, and has the added advantage of being widely known and cited. Unfortunately, MoCap was designed to support research in activity segmentation, not motion segmentation as defined here. Therefore the provided ground truth data is at the level of activities, not motions. For example, *punching* is labeled as a single activity. Each instance of *punching* is actually a sequences of punches, and each punch is a combination of two motions: (1) throwing the punch, and (2) retracting the arm afterward. As a result, the MoCaP data set contains many, many more motions and transitions between motions than it does labeled activities and transitions between labeled activities.

We applied GCA to the 16 videos with ground truth data in MoCaP (see supplementary material). Everywhere the ground truth data suggests a transition between actions, GCA detects a transition between motions. This is to be expected, since any action transition is also a motion transition. Unfortunately, the vast majority of motion transitions are not action transitions, so there is no way to verify or refute the majority of motion transitions detected by GCA on MoCaP data.

To gain an intuition as to whether the motion transitions detected by GCA in MoCaP data are meaningful, we need to look at an example in more detail. Figure 11 shows the first and second curvatures estimated by GCA for a MoCap video. Looking at the bottom of Figure 11, the horizontal axis is once again time, represented as frame numbers. The green, teal, and gray regions represent activities in the ground truth data, namely *jumping, punching,* and *kicking*. The white regions are examples of the *walking* activity, except for the last white regions, where the subject throws a punch.

The blue curve in Figure 11 is κ_1 as estimated by GCA; the red curve is κ_2. The identified peaks in κ_1 and κ_2 are marked with colored dots, and the pose of the subject during the identified peaks are shown above in the matching color. By visual inspection, the peaks line up with transitions between motions in the activities, for example jumping, landing, and then jumping again. We can count the number of punches thrown in the teal region by counting the peaks that correspond to transitions between punch and retract, and between retract and punch. We did not mark the peak during walking activities, but again there is a series of distinct (albeit smaller) peaks that correspond to strides.

VII. Conclusions

This paper proposes Generalized Curvature Analysis (GCA) for estimating generalized curvatures of curves lying in an n-dimensional space. We derive expressions for generalized curvatures at the mean of an ϵ-ball of data in terms of the singular values. The Frenet frame is approximated by the left singular vectors. Two adaptive algorithms are proposed for automatically determining the local data set for computing the singular values and vectors. The complexity of these algorithms is $O(nd^2)$, where n is the number of sampled data points and d is the dimensionality of the points. The first several generalized curvatures appear to be stable in the presence of noise and hence this local SVD approach provides a robust alternative to numerical differentiation for estimating curvature in high dimensions.

We illustrate that the generalized curvatures allow us to segment pose streams into motions without knowing the set of motions in advance. The beginnings and endings of human motions are marked by high curvatures, while the main part of the motion – the so-called transport phase – is characterized by low curvature. This observation leads to a simple, curvature-based temporal segmentation algorithm that divides pose streams into motions with intervening transitions, without assuming that subjects pause between motions or that all motions rhythmically repeat.

There are many open questions relating to understanding and characterizing the motion signatures in terms of curvature. The application of machine learning approaches to feature vectors of curvatures could provide a powerful approach to understanding motion in data streams. The construction of complete actions out of a dictionary of motions is also of potential interest. Further, the temporal evolution of the Frenet frame has not been exploited in this study and may provide additional features to characterize the curve.

To encourage other researchers to explore generalized curvature analysis, we are releasing unrestricted GCA source code written in Python. The code can be downloaded [4] and used to estimate generalized curvatures for any set of N-dimensional points sampled in time from an underlying curve. The same site also includes the PALKA data set, including both the raw data and hand-labeled ground truth data, and videos based on Figure 11 that illustrate how κ_1 and κ_2 vary over the course of a PALKA and MoCaP video.

Appendix
Derivation of Curvature Equations

In this appendix we derive the expressions for the generalized curvatures κ_1, κ_2 that serve to illustrate the computation for higher dimensions. We begin by describing the general setting. A curve in n-dimensions is a map

$$x : [a, b] \to \mathbb{R}^n$$

In two dimensions we consider the circle parameterized by t via the function

$$x : [0, 2\pi] \to \mathbb{R}^2$$

where $(x_1(t), x_2(t)) = (a\cos(\alpha t), a\sin(\alpha t))$ and the helix three dimensions is $x(t) = (a\cos(\alpha t), a\sin(\alpha t), bt)$. Moreover, these closed form solutions can be readily extended to n-dimensional curves with constant curvature [45].

The mean value of an arbitrary curve in an ϵ-ball about the point τ is given by

$$\overline{x} = \frac{1}{2\epsilon}\int_{\tau-\epsilon}^{\tau+\epsilon} x(t)dt.$$

Without loss of generality we can shift the origin of the parameterization by setting $s = t - \tau$. Thus, for the circle $\overline{x} = (a\sin(\alpha\epsilon)/\alpha\epsilon, 0)$ and for the helix $\overline{x} = (a\sin(\alpha\epsilon)/\alpha\epsilon, 0, 0)$.

[4] Code available at the URL site https://www.cs.colostate.edu/~vision/gecat_toolset

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

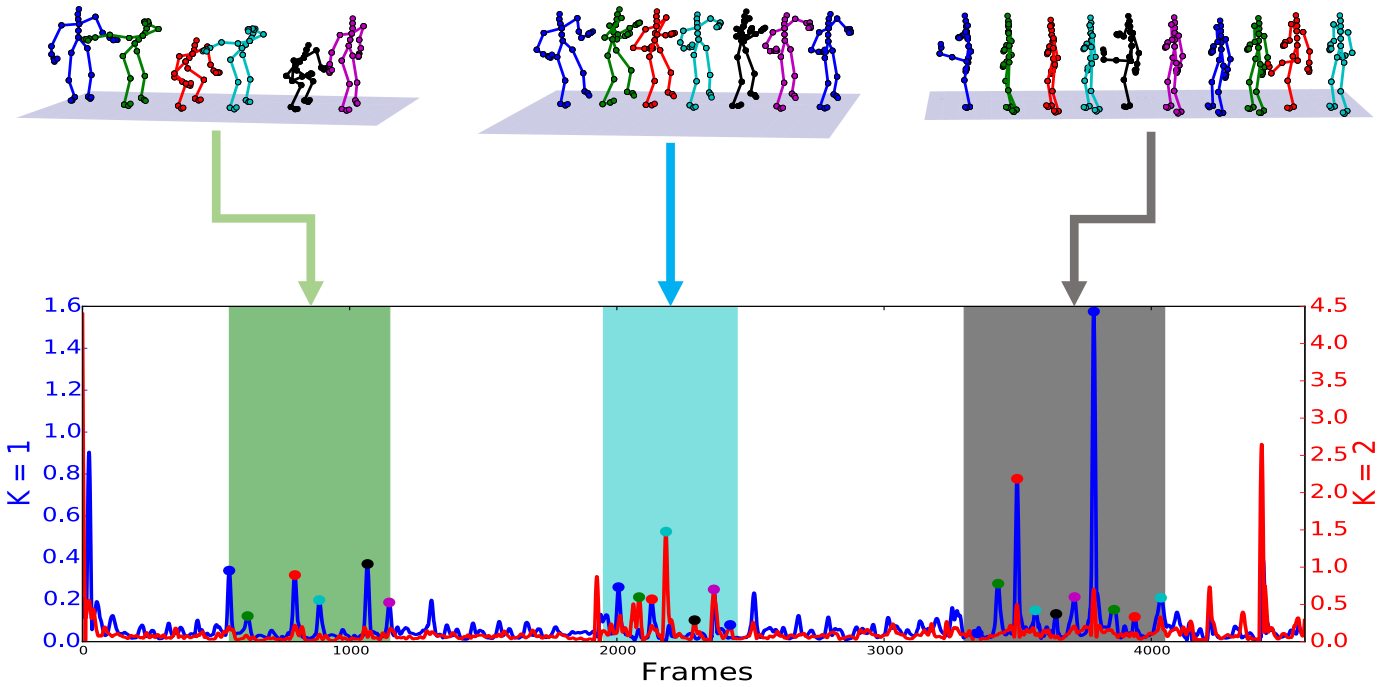

Fig. 11. First and second curvatures on the first video from the MoCaP data set. The horizontal axis on the bottom represents time, measured in frames (at 120 fps). The shaded areas represent jumping, punching and kicking. The first three white areas represent walking activities, whereas last white area represents punching. The peaks representing transitions between motions are identified with colored dots. For each dot, there is a skeleton pose in the same color above. This shows the body position when the curvature peaked, and suggests that the peaks represent transitions between motions. There is also a regular pattern of smaller peaks in the waking activities, representing individual steps. The last white area corresponds to a second punching activity and has high curvature peaks. A video representation of this figure showing curvatures and poses over time is available at https://www.cs.colostate.edu/~vision/gecat_toolset

The components of the *curve-mean* centered covariance matrix C defined along the interval $s \in [-\epsilon, \epsilon]$ can be written as

$$C_{ij} = \frac{1}{2\epsilon} \int_{-\epsilon}^{\epsilon} (x_i(s) - \overline{x}_i)(x_j(s) - \overline{x}_j) ds$$

where $x_j(s)$ is the component of the vector $x(s)$ representing a point on the curve. Hence,

$$C_{11}(\epsilon) = \frac{1}{2\epsilon} \int_{-\epsilon}^{\epsilon} \left(a\cos(\alpha s) - \frac{a\sin(\alpha\epsilon)}{\alpha\epsilon} \right)^2 ds$$

$$C_{22}(\epsilon) = \frac{1}{2\epsilon} \int_{-\epsilon}^{\epsilon} a^2 \sin^2(\alpha s) ds.$$

The off-diagonal terms are $C_{12} = C_{21} = 0$. since the integrand is an odd function. Given that the covariance matrix is diagonal for $n = 2$, the eigenvalues of the Karhunen-Loève transformation are given by C_{11} and C_{22}. We follow the usual convention of ordering the eigenvalues by decreasing magnitude so

$$\lambda_1 = \frac{1}{3} a^2 \alpha^2 \epsilon^2 + O(\epsilon^4), \quad \lambda_2 = \frac{1}{45} a^2 \alpha^4 \epsilon^4 + O(\epsilon^6)$$

from which it follows

$$\lim_{\epsilon \to 0} \frac{\lambda_2}{\lambda_1^2} = \frac{1}{5} \frac{1}{a^2}$$

where we observe that the scale α cancels out from the equation, i.e., this equation holds for arbitrary α. Hence, given the curvature of a circle is $\kappa_1 = 1/a$, we obtain the expression for curvature in terms of the eigenvalues of the covariance matrix in the limit, i.e.,

$$\kappa_1^2 = 5 \lim_{\epsilon \to 0} \frac{\lambda_2}{\lambda_1^2}.$$

Now taking $n = 3$ the curve-mean covariance matrix for the helix along the interval $[-\epsilon, \epsilon]$ is

$$C = \begin{bmatrix} \frac{1}{45} a^2 \epsilon^4 & 0 & 0 \\ 0 & \frac{1}{3} a^2 \epsilon^2 - \frac{1}{15} a^2 \epsilon^4 & \frac{1}{3} ab\epsilon^2 - \frac{1}{30} ab\epsilon^4 \\ 0 & \frac{1}{3} ab\epsilon^2 - \frac{1}{30} ab\epsilon^4 & \frac{1}{3} b^2 \epsilon^2 \end{bmatrix}$$

up to the order $O(\epsilon^6)$ terms which have been truncated. The Taylor Series for the eigenvalues are then computed to be

$$\lambda_1 = \frac{1}{3}(a^2 + b^2)\epsilon^2 + O(\epsilon^4), \lambda_2 = \frac{1}{45} a^2 \epsilon^4 + O(\epsilon^6)$$

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

$$\lambda_3 = \frac{1}{1575}\frac{a^2b^2}{a^2+b^2}\epsilon^6 + O(\epsilon^8)$$

Observing that

$$\lim_{\epsilon\to 0}\frac{\lambda_3}{\lambda_1\lambda_2} = \frac{3}{35}\frac{b^2}{(a^2+b^2)^2}$$

we conclude that

$$\kappa_2^2 = \frac{35}{3}\lim_{\epsilon\to 0}\frac{\lambda_3}{\lambda_1\lambda_2}$$

where we have used $\kappa_1 = a/(a^2+b^2)$ and $\kappa_2 = b/(a^2+b^2)$, respectively.

Employing the equation for the helix in n-dimensions and proceeding analogously as above, we have the general form for the k^{th} generalized curvature

$$\kappa_k^2 = \frac{(2k+1)(2k+3)}{3}\lim_{\epsilon\to 0}\frac{\lambda_{k+1}}{\lambda_1\lambda_k}$$

for $k = 1, \ldots, n-1$ for curves residing in n-dimensions.

ACKNOWLEDGMENT

This paper is based on research partially supported by the National Science Foundation under Grant No. DMS-1322508, and the Defense Advanced Projects Research Agency (DARPA) and the U.S. Army Research Laboratory (ARL) under contract W911NF-15-1-0459. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE MulitMedia*, vol. 19, pp. 4–10, 2012.

[2] *ASUS Xtion PRO LIVE*, https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/.

[3] *Leap Motion*, https://www.leapmotion.com/.

[4] *Intel RealSense Technology*, https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html.

[5] C. Tang, W. Li, C. Hou, P. Wang, Y. Hou, J. Zhang, and P. O. Ogunbona, "Online action recognition based on incremental learning of weighted covariance descriptors," *arXiv preprint arXiv:1511.03028*, 2015.

[6] W. Ding, K. Liu, F. Cheng, and J. Zhang, "Learning hierarchical spatio-temporal pattern for human activity prediction," *Journal of Visual Communication and Image Representation*, vol. 35, pp. 103–111, 2016.

[7] G. Zhu, L. Zhang, P. Shen, and J. Song, "An online continuous human action recognition algorithm based on the Kinect sensor," *Sensors*, vol. 16, no. 2, p. 161, 2016.

[8] G. Yu, Z. Liu, and J. Yuan, "Discriminative orderlet mining for real-time recognition of human-object interaction," in *Computer Vision–ACCV 2014*. Springer, 2014, pp. 50–65.

[9] X. Álvarez-Vizoso, R. Arn, B. Draper, M. Kirby, and C. Peterson, "Geometry of curves in $\mathbf{R^n}$, Singular Value Decomposition, and Hankel determinants," *arXiv preprint arXiv:1511.05008v2*, 2017.

[10] M. Spivak, "Differential geometry, volume 1." *Inc.,*, 1979.

[11] H. Shuzi, Y. Jing, and C. Huan, "Human actions segmentation and matching based on 3d skeleton model," in *Control Conference (CCC), 2013 32nd Chinese*. IEEE, 2013, pp. 5877–5882.

[12] J. Shan and S. Akella, "3d human action segmentation and recognition using pose kinetic energy," in *Advanced Robotics and its Social Impacts (ARSO), 2014 IEEE Workshop on*. IEEE, 2014, pp. 69–75.

[13] L. Xia, C.-C. Chen, and J. Aggarwal, "View invariant human action recognition using histograms of 3d joints," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012, pp. 20–27.

[14] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, "Sequence of the most informative joints (SMIJ): A new representation for human skeletal action recognition," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 24–38, 2014.

[15] M. Barnachon, S. Bouakaz, B. Boufama, and E. Guillou, "Ongoing human action recognition with motion capture," *Pattern Recognition*, vol. 47, no. 1, pp. 238–247, 2014.

[16] I. Lillo, A. Soto, and J. Niebles, "Discriminative hierarchical modeling of spatio-temporally composable human activities," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 812–819.

[17] L. Miranda, T. Vieira, D. Martínez, T. Lewiner, A. W. Vieira, and M. F. Campos, "Online gesture recognition from pose kernel learning and decision forests," *Pattern Recognition Letters*, vol. 39, pp. 65–73, 2014.

[18] M. Raptis, D. Kirovski, and H. Hoppe, "Real-time classification of dance gestures from skeleton animation," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*. ACM, 2011, pp. 147–156.

[19] F. Zhou, F. Torre, and J. K. Hodgins, "Aligned cluster analysis for temporal segmentation of human motion," in *Automatic Face & Gesture Recognition, 2008. FG'08. 8th IEEE International Conference on*. IEEE, 2008, pp. 1–7.

[20] F. Zhou, F. De la Torre, and J. K. Hodgins, "Hierarchical aligned cluster analysis for temporal clustering of human motion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 3, pp. 582–596, 2013.

[21] B. Krüger, A. Vögele, T. Willig, A. Yao, R. Klein, and A. Weber, "Efficient unsupervised temporal segmentation of motion data," *arXiv preprint arXiv:1510.06595*, 2015.

[22] A. Vögele, B. Krüger, and R. Klein, "Efficient unsupervised temporal segmentation of human motion." in *Symposium on Computer Animation*. Citeseer, 2014, pp. 167–176.

[23] H. S. Koppula, R. Gupta, and A. Saxena, "Learning human activities and object affordances from RGB-D videos," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 951–970, 2013.

[24] J. J. Koenderink, *Solid Shape*. Cambridge, MA: MIT Press, 1990.

[25] O. Faugeras, *Cartan's moving frame method and its application to the geometry and evolution of curves in the Euclidean, affine and projective planes*. Berlin: Springer, 1994.

[26] S. W. Zucker, "Differential geometry from the Frenet of view: Boundary detection, stereo, texture and color," in *Handbook of Mathematical Models in Computer Vision*. Springer US, 2006, pp. 357–373.

[27] J. Pegna, *Variable sweep geometric modeling*. UMI, 1988.

[28] W. F. Bronsvoort and F. Klok, "Ray tracing generalized cylinders," *ACM Transactions on Graphics (TOG)*, vol. 4, no. 4, pp. 291–303, 1985.

[29] M. Zerroug and R. Nevatia, "Quasi-invariant properties and 3-d shape recovery of non-straight, non-constant generalized cylinders," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*. IEEE, 1993, pp. 96–103.

[30] M. G. Wagner and B. Ravani, "Curves with rational Frenet-Serret motion," *Computer Aided Geometric Design*, vol. 15, no. 1, pp. 79–101, 1997.

[31] Z. Duric, A. Rosenfeld, and L. S. Davis, "Egomotion analysis based on the Frenet-Serret motion model," in *International Journal of Computer Vision*. Citeseer, 1995.

[32] Z. Duric, E. Rivlin, and A. Rosenfeld, "Understanding object motion," *Image and vision computing*, vol. 16, no. 11, pp. 785–797, 1998.

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2869741, IEEE Transactions on Pattern Analysis and Machine Intelligence

[33] Z. Duric, R. Goldenberg, E. Rivlin, and A. Rosenfeld, "Estimating relative vehicle motions in traffic scenes," *Pattern Recognition*, vol. 35, no. 6, pp. 1339–1353, 2002.

[34] K.-R. Kim, P. T. Kim, J.-Y. Koo, and M. R. Pierrynowski, "Frenet-Serret and the estimation of curvature and torsion," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 4, pp. 646–654, 2013.

[35] S. Chern, "Moving frames," *The Mathematical Heritage of Élie Cartan (Lyon, 1984), Astérique*, vol. 1985, pp. 67–77, 1985.

[36] C. Qu, "Invariant geometric motions of space curves," in *Computer Algebra and Geometric Algebra with Applications*. Springer, 2005, pp. 139–151.

[37] W.-C. Wang, P.-C. Chung, H.-W. Cheng, and C.-R. Huang, "Trajectory kinematics descriptor for trajectory clustering in surveillance videos," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 1198–1201.

[38] M. Vochten, T. De Laet, and J. De Schutter, "Comparison of rigid body motion trajectory descriptors for motion representation and recognition," in *2015 IEEE International Conference on Robotics and Automation*, 2015.

[39] T. Hastie and W. Stuetzle, "Principal curves," *Journal of the American Statistical Association*, vol. 84, pp. 502–526, 1989.

[40] F. J. Solis, "Geometry of local adaptive Galerkin bases," *Applied Mathematics and Optimization*, vol. 41, pp. 331–342, 2000.

[41] M. Kirby, *Geometric data analysis: an empirical approach to dimensionality reduction and the study of patterns*. John Wiley & Sons, Inc., 2000.

[42] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[43] M. Firman, "RGBD datasets: Past, present and future," in *IEEE Conference on Computer Vision and Machine Intelligence*, 2016.

[44] S. Fothergill, H. M. Mentis, P. Kohli, and S. Nowozin, "Instructing people for training gestural interactive systems," in *CHI*, J. A. Konstan, E. H. Chi, and K. Höök, Eds. ACM, 2012, pp. 1737–1746.

[45] W. Kühnel, *Differential geometry: curves-surfaces-manifolds*. American Mathematical Soc., 2006, vol. 16.

Robert Arn received his Ph.D. from the Department of Mathematics, Colorado State University in 2016. His dissertation was entitled "On the Formulation and Uses of SVD Based Generalized Curvatures". He was hired after graduation by Northrop Grumman Corporation.

Pradyumna Narayana received his B.S. in Information Technology from Jawaharlal Nehru Technological University, Hyderabad, India in 2011 and M.S. from the Department of Computer Science, Colorado State University in 2014. He is working towards his Ph.D. and is expected to receive his degree from Colorado State University in 2018. His research interests include gesture recognition, action recognition and image recognition.

Tegan Emerson received her B.S. in Mathematics from Oregon State University in 2011 and her M.S. and Ph.D. in Mathematics at Colorado State University in 2013 and 2017. Currently, she is a Karle Fellow working as a Mathematician at the Naval Research Laboratory. Her research interests include geometric and topological data analysis, dimensionality reduction, algorithms for image and video processing, and optimization. She won "Best Paper" at the 2016 Workshop on Hyperspectral Imaging and Signal Processing: Evolutions in Remote Sensing.

Bruce Draper received his B.S. from Yale University in 1984, and his M.S. and Ph.D. from the University of Massachusetts (Amherst) in 1987 and 1993, respectively. He has been on the faculty at Colorado State University since 1996, and a full professor since 2011. An active computer vision researcher, he is a member of the IEEE who has been actively involved in community, including serving as the General Co-chair for CVPR in 1999.

Michael Kirby received his S.B. degree in Mathematics from the Massachusetts Institute of Technology and Ph.D. from the Division of Applied Mathematics, Brown University. He is currently a Professor in the Department of Mathematics, Colorado State University with a joint appointment in the Department of Computer Science. His research interests include low dimensional modeling, geometric models for data and optimization. He authored the textbook Geometric Data Analysis. He was an Alexander von Humboldt Fellow at the Institute for Information Verarbeitung at the University of Tuebingen, Germany. He also was awarded an IBM Faculty Fellowship, and the College of Natural Sciences Award for Graduate Education.

Chris Peterson received the Ph.D. degree in Mathematics from Duke University (Durham, North Carolina) in 1994. He is currently a professor in the Department of Mathematics at Colorado State University (Fort Collins, Colorado). Previously he held postdoctoral research positions at The University of Notre Dame and Washington University in Saint Louis. His research interests include pure and applied algebraic geometry, optimization, and geometric data analysis.

0162-8828 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.