

DISSERTATION

ON THE FORMULATION AND USES OF SVD-BASED GENERALIZED
CURVATURES

Submitted by

Robert T. Arn

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2016

Doctoral Committee:

Advisor: Michael Kirby

Co-Advisor: Chris Peterson

Dan Bates

Raoul Reiser

Copyright by Robert T. Arn 2016

All Rights Reserved

ABSTRACT

ON THE FORMULATION AND USES OF SVD-BASED GENERALIZED CURVATURES

In this dissertation we consider the problem of computing generalized curvature values from noisy, discrete data and applications of the provided algorithms. We first establish a connection between the Frenet-Serret Frame, typically defined on an analytical curve, and the vectors from the local Singular Value Decomposition (SVD) of a discretized time-series. Next, we expand upon this connection to relate generalized curvature values, or curvatures, to a scaled ratio of singular values. Initially, the local singular value decomposition is centered on a point of the discretized time-series. This provides for an efficient computation of curvatures when the underlying curve is known. However, when the structure of the curve is not known, for example, when noise is present in the tabulated data, we propose two modifications. The first modification computes the local singular value decomposition on the mean-centered data of a windowed selection of the time-series. We observe that the mean-center version increases the stability of the curvature estimations in the presence of signal noise. The second modification is an adaptive method for selecting the size of the window, or local ball, to use for the singular value decomposition. This allows us to use a large window size when curvatures are small, which reduces the effects of noise thanks to the use of a large number of points in the SVD, and to use a small window size when curvatures are large, thereby best capturing the local curvature. Overall we observe that adapting the window size to the data, enhances the estimates of generalized curvatures. The combination of these two modifications produces a tool for computing generalized curvatures with reasonable precision and accuracy. Finally, we compare our algorithm, with and without

modifications, to existing numerical curvature techniques on different types of data such as that from the Microsoft Kinect 2 sensor. To address the topic of action segmentation and recognition, a popular topic within the field of computer vision, we created a new dataset from this sensor showcasing a pose space skeletonized representation of individuals performing continuous human actions as defined by the MSRC-12 challenge. When this data is optimally projected onto a low-dimensional space, we observed each human motion lies on a distinguished line, plane, hyperplane, etc. During transitions between motions, either the dimension of the optimal subspace significantly, or the trajectory of the curve through pose space nearly reverses. We use our methods of computing generalized curvature values to identify these locations, categorized as either high curvatures or changing curvatures. The geometric characterization of the time-series allows us to segment individual, or geometrically distinct, motions. Finally, using these segments, we construct a methodology for selecting motions to conjoin for the task of action classification.

TABLE OF CONTENTS

Abstract	ii
List of Tables	viii
List of Figures	ix
Chapter 1. Introduction	1
Chapter 2. Theory of Curves.....	6
2.1. Introduction to Curves.....	6
2.2. Introduction to Properties of Curves in \mathbb{R}^3	10
2.2.1. Curvature.....	12
2.2.2. Torsion	15
2.2.3. Frenet Frame in Three Dimensions	17
2.3. The Frenet-Serret Frame and Curves in \mathbb{R}^n	18
Chapter 3. Generalized Curvature Analysis Via Singular Values.....	23
3.1. Introduction	23
3.1.1. Local approximation of curves in \mathbb{R}^3 and \mathbb{R}^4	23
3.1.2. Curvature relations $n = 5$	25
3.1.3. Curvature relations $n = 6$	26
3.2. The Local Singular Value Decomposition.....	27
3.2.1. Formulation.....	28
3.2.2. Two Dimensions	32
3.2.3. Three Dimensions	33
3.2.4. Four Dimensions	34

3.2.5.	Patterns in higher dimensions	34
3.2.6.	Five and Six Dimensions	35
3.3.	Algorithm: Curvature Estimation on Discretely Sampled Smooth Curve	38
3.4.	An example	39
3.5.	Conclusion	41
Chapter 4. A Practical Method for Estimating Generalized Curvatures from Noisy		
	Sample Curves in n-dimensions	43
4.1.	Introduction	43
4.2.	Curvature from Eigenvalues	44
4.2.1.	Formulation	44
4.2.2.	The Curvature Algorithm	46
4.3.	Generalized Curvature from Noisy Points	48
4.3.1.	Algorithm: Estimating Curvature on Noisy Sampled Curve	49
4.4.	Optimizing Data Windows	50
4.4.1.	Algorithm: Adaptive Windowing	53
4.4.2.	Algorithm: Noisy Sampled Adaptive Windowing	54
4.4.3.	Algorithm: Numerical Derivative Curvature Estimation	55
4.5.	Synthetic Example	56
4.6.	Conclusion	59
Chapter 5. Human Motion Segmentation via Generalized Curvature		
5.1.	Introduction	61
5.2.	Related Work	63
5.2.1.	Human Motion	63

5.2.2.	Segmenting Pose Streams	64
5.2.3.	Differential Curves in Computer Vision	65
5.3.	Experiment: Preliminary Human Motion	65
5.3.1.	Evaluation Methodology	66
5.3.2.	Experiment Design	68
5.3.3.	Preliminary Higher-Dimensional Curvatures	73
5.4.	Experiment: Full PALKA Dataset	76
5.4.1.	Algorithm: Temporal Segmentation	77
5.4.2.	Adaptive Parameter on Segmentation Results	80
5.5.	Conclusion	82
Chapter 6.	Human Action Classification	83
6.1.	Introduction	83
6.2.	Algorithm: Dynamic Time Warping	84
6.3.	Clustering	89
6.3.1.	Algorithm: Agglomerative Clustering	92
6.4.	Hidden Markov Models	92
6.4.1.	A Basic HMM Example	93
6.4.2.	HMMs on PALKA	95
6.5.	Classifying PALKA Human Actions	96
6.5.1.	Curvature Classification	97
6.5.2.	Frame Classification	98
6.5.3.	Random Length Classification	99
6.5.4.	Interpolation Classification	99
6.5.5.	Random Results	99

6.5.6. Classification Results.....	99
6.6. Conclusion.....	102
Chapter 7. Conclusions.....	103
Bibliography.....	106
Appendix A. Datasets.....	114
A.1. MSRC-12.....	114
A.2. Pattern Analysis Laboratory Kontinuous Actions (PALKA).....	115
A.2.1. Collection Details.....	116
A.2.2. Data Action Descriptions.....	117
A.2.3. Skeleton Data Example.....	118
Appendix B. Code.....	122
B.1. SVD-based generalized curvature using on the curve subspaces with a fixed window size.....	122
B.2. SVD-based generalized curvature using off the curve subspaces with adaptive window sizes.....	127
B.3. Numerical Derivative Based Curvature.....	135
B.4. Dynamic Time Warping.....	137

LIST OF TABLES

5.1	Comparison of in-motion to in-transition curvatures estimated by five techniques. The technique names (algorithms) are described in the first paragraph of Section 5.3.2. The measures are ratios of estimated curvatures during transitions between motions over curvatures within motions. Since curvatures should be high during transitions and low during motions, high values are better.....	71
5.2	Counts of motion/transition sequence pairs for which each technique had the highest (best) ratio. The first set of comparisons are among all five techniques: numerical derivatives (Algorithm 5), method in Chapter 3 (Algorithm 1), extended off-curve (Algorithm 2), extended with adaptive windows (Algorithm 3), and with both extensions (Algorithm 4). The second set of comparison compares numerical derivatives (the previous state of the art), to the method proposed in Chapter 3 (Algorithm 1) for on-the-curve and off-the-curve proposed here with and without adaptive window size. The values being compared are ratios of means, medians or maximums.....	72
5.3	Comparisons of quality measures for κ_1 through κ_{12} , as computed using both proposed extensions to the method in Chapter 3.....	76
5.4	Segmentation results on PALKA dataset.....	80
6.1	Probabilities between all possible states in a Markov Model Example. Current state (left) future state (top).....	93
A.1	MSRC-12 Ordered Human Action List.....	114

LIST OF FIGURES

1.1	An illustration of selected frames from a Kinect 2 video as displayed on the original $75D$ curve, optimally projected into $3D$	1
1.2	Example of a discretely sampled, smooth, noise-less curve (left), and the associated curvature, κ_1 (middle), and torsion, κ_2 (right) profiles.	2
2.1	Example of a regular curve (left) and a non-regular curve (right). The cycloid (right) is smooth everywhere $x \neq 4\pi n, n \in \mathbb{Z}$	7
2.2	Three dimensional curve, $\gamma(t) \subset \mathbb{R}^3$ (black), with the unit tangent (blue), unit normal (red), and unit binormal (green) vector at a single instance, $\gamma(t_0)$ on the curve.	12
2.3	Curve, γ , with curvature defined at the point s_0 to be $\frac{1}{r}$ of the osculating circle. Modified from Image source: [1].	14
3.1	Twisted Cubic example $-1 \leq t \leq 4$. Point $t = 3$ (black dot). First singular vector (red). Second singular vector (magenta). Third singular vector (green).	40
4.1	Illustration of the adaptive windowing algorithm. (Left) To compute the window size to use around point p_i , select a window using an equal and fixed number of points surrounding p_i : $[p_{i-a} \dots p_i \dots p_{i+a}]$. Compute the mean, μ_s of the data in this window. (Middle Left) Compute the two-dimensional subspace of best fit, given by the eigenvectors $\{e_1, e_2\}$ of $[p_{i-a} - \mu_s \dots p_i - \mu_s \dots p_{i+a} - \mu_s]$. (Middle Right) Project data points onto the subspace of best fit on the left side of the curve. Compute the ratio of $(I - \mathbb{P})p_l / (\mathbb{P}p_l + \epsilon_{\text{rat}})$ where ϵ_{rat} is a small offset to keep the denominator from vanishing. Compare this to the cutoff value selected. Do the	

	same for points on the other side of p_i . (Right) Use the results of these ratios and the cutoff value to determine the number of points to include in the computation of generalized curvatures. Note, the mean of this window, μ_m is not necessarily the same as μ_s	53
4.2	Curvature estimation errors. The blue lines show the signed errors in curvature when curvatures are estimated from local derivatives, or Algorithm 5. The red lines show the errors when curvatures are estimated using Algorithm 1. The black lines show the residual errors using Algorithm 4.	58
4.3	Decomposing Algorithm 4. The blue lines show the result of computing curvatures using Algorithm 2. The red lines show the result of adapting the window size to the underlying curvature, or Algorithm 1+3. The combination of these two techniques, shown in black, is Algorithm 4.	59
5.1	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.	61
5.2	3D projection of a 75 dimensional pose stream showing three actions (Push Right, Kick and Push Right). Sections of the trajectory shown in blue represent atomic motions, while sections shown in red represent transitions between motions. As seen here, motion trajectories tend to have low curvature, whereas the trajectories of transitions between motions are highly curved.	68
5.3	Estimated κ_1 curvatures over time as person performs a sequence of three actions with six atomic motions. The figure is organized vertically into five plots, one for each curvature estimation method. The top plot shows the traditional numerical	

derivative method, Algorithm 5. The second plot shows the method described by Algorithm 1. The third plot shows the off-curve extension in Chapter 3, Algorithm 2. The fourth plot shows the Chapter 3 (Algorithm 3) method with adaptive windows. The fifth (bottom) plot shows both the off-curve and adaptive window extensions, Algorithm 4. The horizontal axes are time (at 30fps), while the vertical axes are estimated κ_1 . Vertical red bars indicate the start of a motion, while vertical blue bars indicate the end of a motion. Therefore, sequences from a red bar to a blue one should be low-curvature motions, whereas sequences from a blue bar to a red one should be high-curvature transitions. 70

5.4 κ_1 values computed by all five curvature estimation methods over all ten pose streams. 73

5.5 12 dimensions of curvature (κ_1 through κ_{12}) as estimated using the proposed extensions (Algorithm 4) to the method in Chapter 3 (Algorithm 1). Roughly the first six dimensions of curvature (shown in the top two bars) seem reliable (see Table 5.3 for statistics). Dimensions 7 through 12 still seem to contain some signal, but many transitions no longer show elevated curvatures. (Remember, we expect low curvatures between red bars and blue bars, and higher "spiky" curvatures between blue bars and red bars.) 75

5.6 (Right) 3D Smoothed Projection of 75D curve of two sequential actions; Lift Outstretched Arms followed by Kick. (Left) 1) Raising Arms 2) Transition between Raising Arms and Lowering Arms 3) Lowering Arms 4) Transition between Lift Outstretched Arms and Kick 5) Raising Left Leg 6) Transition between Raising Left Left and Lowering Left Leg 7) Lowering Left Leg 8) Transition between Kick and next action. 78

5.7	Estimated curvatures over time as person performs a sequence of three actions with six atomic motions. The figure is organized vertically into three plots. The top plot shows the traditional numerical derivative method. The second shows the Algorithm 1. The third plot shows the proposed curvature estimation method. The horizontal axes are time (at 30fps), while the vertical axes are estimated curvature magnitudes. Vertical blue bars indicate the start of a motion according to the hand-labeled data, while vertical red bars indicate the end of a motion. Therefore, sequences from a blue bar to a red one are motions, whereas sequences from a red bar to a blue one are transitions between motions.	79
5.8	Accuracy of Algorithm 6 using Algorithm 4 algorithm used to measure motion segmentation on the PALKA dataset as the parameter h (or adaptive cutoff value), used in optimizing the data windows varies.	81
6.1	Dynamic Time Warp counter example. The points on the top signal (blue) are paired with points on the bottom signal (red) by time. This linear based Euclidean distance measure does not align signals based on time and will only compute the distance up to the length of the smallest (in time) curve or time-series.	85
6.2	Dynamic Time Warp example. The points on the top signal (blue) are paired with points on the bottom signal (red). Paired points are described by black lines. This non-linear based Euclidean distance measure aligns signals based on time and provides a better similarity measure.	86
6.3	Agglomerative clustering example. Beginning with each signal, $\{A, B, C, D, E, F\}$ in its own cluster, clusters combine until all elements are in a single cluster.	90
6.4	Probability plot of moving from one state to another in Markov Model Example..	94

6.5	Diagram of Skelton produced by skeletonization algorithm from the Kinect 2 sensor. Labels for all 25 identified body points as defined by the Kinect SDK.	97
6.6	Comparison of various techniques to classify PALKA actions as a function of the number of declared clusters. The blue line shows the accuracy as the original videos are segmented by method described in Chapter 5 using Algorithms 4 and 6 to compute curvature. The green line shows classification accuracy using each individual frame as its own segment. The red line shows the accuracy if each of the original videos are divided into random segments. And the teal line takes each segment as defined by the method described in Chapter 5 using Algorithms 4 and 6 to compute curvature. Then each segment is extended to a fixed length using linear interpolation. An averaged euclidean distance is used instead of a DTW algorithm. Also, not shown is completely random results. As there are 12 action classes in PALKA, perfectly random results are computed as 8.3%.	101
A.1	Diagram of Skelton produced by skeletonization algorithm from the Kinect 2 sensor. Labels for all 25 identified body points as defined by the Kinect SDK.	115
A.2	Action 1,2,3,4 - Lift Outstretched Arms. Action 5,6,7,8 - Action: Kick. Motion 1 - Raising Arms. Motion 2 - Transition between Raising Arms and Lowering Arms. Motion 3 - Lowering Arms. Motion 4 - Transition between Lift Outstretched Arms and Kick. Motion 5 - Raising Left Leg. Motion 6 - Transition between Raising Left Leg and Lowering Left Leg. Motion 7 - Lowering Left Leg. Motion 8 - Transition between Kick and next action.	119
A.3	Example Matlab code to show how to access variables in the provided data files. .	120

CHAPTER 1

INTRODUCTION

The work in this dissertation was originally inspired by a popular problem in computer vision: How can we determine the actions of humans through algorithmic methods? In particular, we have been working with time-series human skeletal data collected from Kinect and Kinect 2 sensors. We observed that the optimal projection of the time-series from an associated continuous human motion may be characterized by a distinguished linear space, e.g., a line or hyperplane. This geometric model suggests human actions exist in contiguous low-dimensional linear spaces, despite numerous degrees of freedom the human body allows. This further suggests that if we had the ability to detect the transition from a n -dimensional space of best fit to a m -dimensional space of best fit, then we could automatically segment continuous human actions without *a priori* information about the number of, or description of, actions performed.

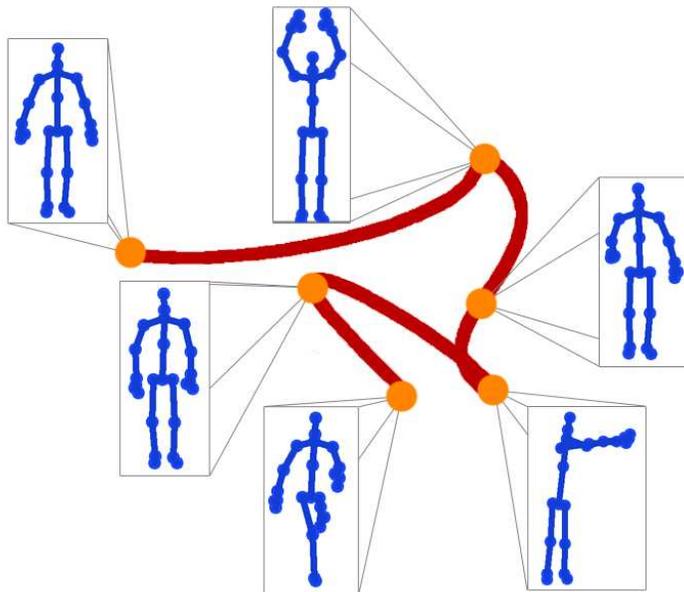


FIGURE 1.1. An illustration of selected frames from a Kinect 2 video as displayed on the original $75D$ curve, optimally projected into $3D$.

One method of determining the region of transition is to look at locations in the time-series which have high or changing generalized curvature values, or curvatures; terms we use interchangeably in this dissertation. A geometric interpretation of curvature in two dimensions is the amount of deviation of a curve from a straight line and torsion is the amount of deviation of a curve from a plane. These concepts have been expanded into n -dimensions by Camille Jordan [2] in 1874. In this dissertation we develop the interpretation of curvature in n -dimensions as a useful method of searching a time-series for points of best fit dimensional transitions.

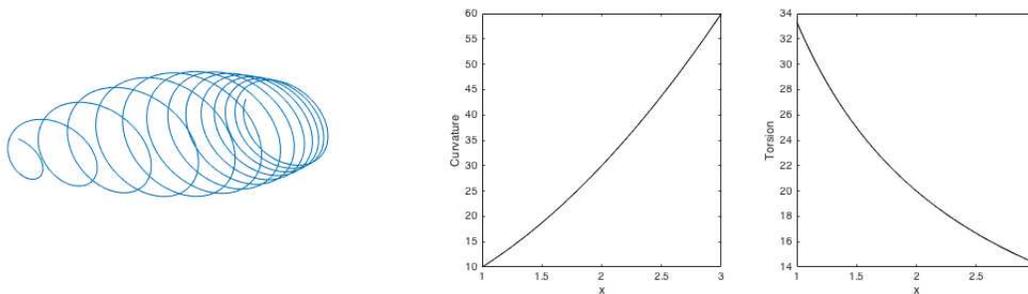


FIGURE 1.2. Example of a discretely sampled, smooth, noise-less curve (left), and the associated curvature, κ_1 (middle), and torsion, κ_2 (right) profiles.

When working with real, possibly noisy sampled data in high-dimensions, curvatures can be challenging to characterize. Traditionally methods of computing generalized curvature values rely on having the analytical version of the curve. Numerical approximations of derivatives grow increasingly inaccurate as the dimension of the curvature increases. Even sophisticated techniques for computing numerical derivatives based on tabulated values, such as Richardson Extrapolation, which are designed to reduce the error term on high order derivatives, have significant limitations to the types of data we are interested in analyzing [3]. In particular, this, and similar methods are prone to large sources of errors when Δt is large, or when the tabular data contains a high amount of noise. In the case of human motion data via the Kinect sensor, this noise is from the skeletonization algorithm. To our

knowledge, prior to this work, there is no numerically robust method of computing high dimensional curvatures on sampled data.

In this dissertation, we explore the connection of the Frenet-Serret Frame (sometimes called the Frenet Frame) to the singular vectors produced from the local singular value decomposition, a common numerically stable numerical analysis technique. From this connection, we have devised a new formula for computing generalized curvatures.

Through an investigation on synthetic data, we found this new formula was excellent for accurately computing curvatures when the data was both noise-free (an exact sampling of a continuous, smooth function) and the curvature values were constant. Deviation from either of these two highly restrictive conditions produces inaccurate curvature estimations. By exploiting some of the mechanics behind this new equation, we created a couple modifications to overcome these limitations.

Armed with these new techniques, we turn our attention back to the motivating problem: given skeletal representations of human actions, can we automate a process to classify the data? The task of action classification has been studied for decades on a variety of data types. In general, the most common approach is to take a video, or video stream, as an input and apply some method to classify pieces of the stream. Based on this labeling, the video is then broken up into the identified segments (a top-down approach).

As we started to approach this problem, our intuition lead us to adopt a bottom-up methodology; given a video, or video stream, can we first segment the data into actions (with no prior knowledge of those actions)? Then, after the videos are segmented can we classify those segments? This approach means we need a very robust method of segmenting the data. As discussed above, an initial observation relating curvature to stages of human

motion, combined with our new techniques of computing curvature on noisy data, gives us a method to start this bottom-up approach.

For this dissertation, we provide accuracies for the task of segmentation on a skeletal based, Kinect 2 dataset we collected. Then we use these segments, combined with some standard (and advanced) techniques used within the computer vision community to obtain preliminary classification results. Our analysis of the data using this approach is strong enough to warrant more investigation.

In Chapter 2, we provide the background information leading up to the formulation of the Frenet-Serret apparatus. This includes starting with the basic definition of a curve, building to the subjects of curvature and torsion, common properties of a curve in \mathbb{R}^3 , and ending with the Frenet-Serret equations. Through starting with examples which can be visualized, and computed, in \mathbb{R}^3 , we give geometric interpretations that will allow us to more easily understand the \mathbb{R}^n case.

In Chapter 3, we start by defining the local singular value decomposition in terms of the *on the curve* covariance matrix. By looking at the Taylor expansion of this problem, we prove the singular vectors are equivalent to the Frenet frame, up to a sign change. For dimensions $n \leq 6$, we then construct a relationship between the local eigenvectors and generalized curvature values via the Frenet Equations. Finally, we make a conjecture, based upon this construction and additional numerical evidence for a new numerically stable formula for computing generalized curvature values.

In Chapter 4, we extend the work in Chapter 3 to the case when the curve is given by discrete data samples. We introduce two additional extensions: a formulation for off-the-curve generalized curvatures, and a method for selecting the appropriate local adaptive ball, i.e., time-window, to perform the SVD calculations.

In Chapter 5, we present applications comparing the various methods of computing generalized curvature. By working with an array of small and large data sets from a Kinect 2 sensor, we explore the computation of generalized curvatures. Using observations about the high-dimensional curve formed by Kinect 2 human activity data and the generalized curvature techniques developed in prior chapters, we give human motion segmentation results.

In Chapter 6, we explore an application of the human motion segmentation to establish preliminary action-based classification results. To do this, we introduce the use of multiple Dynamic Time Warping (DTW) algorithms which gives a similarity measure between two curves that are not necessarily synced temporally. By creating a set of Hidden Markov Models (HMM), we are able to use the DTW distances with the segmented motions to determine which motion segments combine to form individual actions.

Finally, in Appendix A we introduce the Pattern Analysis Laboratory Kontinuous Actions (PALKA) data set, the collection criteria, and hand-labeled ground truth used in the evaluation of Chapters 5 and 6. Other publicly available data sets used in this dissertation are also explained in this appendix. In Appendix B, we provide important pieces of code developed in this thesis including, but not limited to, the practical computation of generalized curvature values evaluated on the curve, and the practical computation of generalized curvature using the mean subtraction and adaptive window size extensions from Chapter 4.

CHAPTER 2

THEORY OF CURVES

In this chapter we introduce basic concepts involving curves, the Frenet Frame, and generalized curvature in a differential geometry context. The material in this chapter follows from several sources [3], [4], [5], and [6]. The rest of this dissertation will assume familiarity with this material as we expand on these concepts.

2.1. INTRODUCTION TO CURVES

To build up the foundation of the work presented in this dissertation, we begin to study the simplest of geometric objects: curves. Our approach for this study will be highly geometric. After a formal definition of a curve, in order to facilitate a visual understanding, we will typically present the material in a low dimensional space, such as \mathbb{R}^3 before generalizing the topics to \mathbb{R}^n .

DEFINITION 2.1.1. *A curve, γ , is defined to be a continuous mapping from an interval $\gamma : [a, b] \subseteq \mathbb{R} \rightarrow \mathbb{R}^n$.*

This is a very basic definition of a curve, and provides us with few mathematical tools. However, by adding the additional constraint of non-vanishing differentiability we can start to analyze curves in many ways.

DEFINITION 2.1.2. *A curve, $\gamma : I \rightarrow \mathbb{R}^n$ is regular if its derivatives up to $\gamma(t)^{(n)} \neq 0$ for all $t \in I$.*

DEFINITION 2.1.3. *A regular curve $\gamma : [a, b] \rightarrow \mathbb{R}^n$ is smooth given that $\gamma'(t)^{(i)} \neq 0$ for all $t \in [a, b]$ and for all $i \leq n \in \mathbb{N}^+$.*

Figure 2.1 shows the canonical examples of regular and non-regular curves, a circle and cycloid respectively.

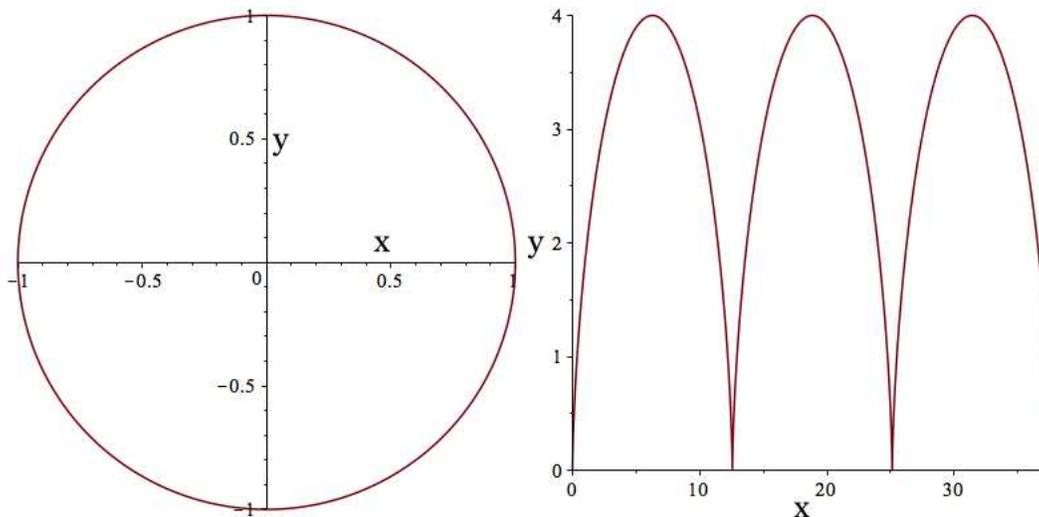


FIGURE 2.1. Example of a regular curve (left) and a non-regular curve (right). The cycloid (right) is smooth everywhere $x \neq 4\pi n$, $n \in \mathbb{Z}$.

DEFINITION 2.1.4. Let $\gamma : (a, b) \rightarrow \mathbb{R}^n$ be a regular curve. Let $h : (c, d) \subset \mathbb{R} \rightarrow (a, b) \subset \mathbb{R}$ be a diffeomorphism, or an invertible function that maps one differential manifold to another such that both the function and the inverse are smooth. Then $\tilde{\gamma} = \gamma \circ h : (c, d) \subset \mathbb{R} \rightarrow \mathbb{R}^n$ is a regular curve, called a reparameterization of γ .

$$\tilde{\gamma}(u) = \gamma \circ h(u) = \gamma(h(u))$$

In other words, start with a curve $\gamma = \gamma(t)$, make a change of parameter $t = h(u)$, and obtain a reparameterized curve $\tilde{\gamma} = \gamma(h(u))$ where u is the new parameter.

With the ability to reparametrize curves, a common reparameterization based on arc length is widely employed. First, we recall the definition of arc length from calculus.

DEFINITION 2.1.5. If $\gamma : [a, b] \rightarrow \mathbb{R}^n$ is a parametrized curve, then for any $a \leq t \leq b$, the arc length from a to t is defined as

$$s(t) = \int_a^t \|\gamma'(u)\| du.$$

Thus, the total arc length of a curve is such that $b = t$.

To elaborate on this definition, the norm of the derivative of the curve over an interval describes the length of that segment of the curve. Since this is a continuous operation, as we vary the length of the segment, the length of the curve (assuming non-zero derivative) varies the arc length in a monotonically increasing manner.

THEOREM 2.1.6. Every regular parametrized curve, γ , can be parametrized by its arc length, $s(t)$ by $\tilde{\gamma} = \gamma(t(s))$ where $t(s)$ is the inverse arc length function.

PROOF. Let $\gamma : [a, b] \rightarrow \mathbb{R}^n$ be a regular parametrized curve and let $s(t)$ be defined as the arc length

$$s(t) = \int_a^t \left\| \frac{d\gamma}{dt} \right\| dt.$$

By definition, since γ is regular,

$$s'(t) = \|\gamma'(t)\| > 0.$$

Then $s(t)$ is a monotonically increasing function. From this, we can establish that $t(s)$, the inverse function of $s(t)$ exists. Thus:

$$\tilde{\gamma}'(s) = \gamma'(t(s))t'(s) = \frac{\gamma'(t(s))}{s'(t(s))} = \frac{\gamma'(t(s))}{\|\gamma'(t(s))\|}$$

Clearly, $\tilde{\gamma}(s)$ is now parametrized by arc length as

$$\|\tilde{\gamma}'(s)\| = \left\| \frac{\gamma'(t(s))}{\|\gamma'(t(s))\|} \right\| = 1$$

□

For the remainder of this chapter, we will use the following notation:

- $\gamma(t)$ denotes an arbitrary regular parametrization
- $\gamma(s)$ denotes an parametrization by arc length
- $\gamma(s_0)$ denotes a point on a curve parametrized by arc length

EXAMPLE 2.1.1. Consider the curve $\gamma(t) = \langle \cos(t), \sin(t), t \rangle$ from $0 \leq t \leq T$. Then

$$\gamma'(t) = \langle -\sin(t), \cos(t), 1 \rangle .$$

The arc length of this curve is then

$$s(t) = \int_0^t \sqrt{\sin^2(u) + \cos^2(u) + 1} dt = \int_0^t \sqrt{2} dt = t\sqrt{2}.$$

Solving for t gives us

$$t = \frac{s}{\sqrt{2}}$$

Now substituting this back into the original curve,

$$\gamma(s) = \left\langle \cos\left(\frac{s}{\sqrt{2}}\right), \sin\left(\frac{s}{\sqrt{2}}\right), \frac{s}{\sqrt{2}} \right\rangle .$$

We can test to make sure this curve has been parametrized by arc length:

$$\|\gamma'(s)\| = \left\| \left\langle -\frac{1}{\sqrt{2}} \sin\left(\frac{s}{\sqrt{2}}\right), \frac{1}{\sqrt{2}} \cos\left(\frac{s}{\sqrt{2}}\right), \frac{1}{\sqrt{2}} \right\rangle \right\|$$

$$\begin{aligned}
&= \sqrt{\frac{1}{2} \sin^2\left(\frac{s}{\sqrt{2}}\right) + \frac{1}{2} \cos^2\left(\frac{s}{\sqrt{2}}\right) + \frac{1}{2}} \\
&= 1
\end{aligned}$$

We also make the note that while we can consider a curve as a single object, it also makes sense to discuss the local properties of a curve. We will use the word “local” to discuss the behavior of a curve around a neighborhood of a point $\gamma(s_0)$. For instance, the local behavior of a curve parametrized by arc length can be studied by means of the Taylor expansion:

$$\gamma(s) = \gamma(s_0) + s\gamma'(s_0) + \frac{s^2}{2}\gamma''(s_0) + \frac{s^3}{6}\gamma'''(s_0) + \mathcal{O}(s^3)$$

of the curve $\gamma(s)$ about the point s_0 .

2.2. INTRODUCTION TO PROPERTIES OF CURVES IN \mathbb{R}^3

Since we now have the tools we need to start to discuss properties of a curve, let us develop the standard nomenclature to describe curves in \mathbb{R}^3 . Curves in this low dimensional space will allow us to visualize concepts important to the Frenet-Serret Frame in Section 2.3.

DEFINITION 2.2.1. *Let $\gamma \subseteq \mathbb{R}^3$ be a regular parametrized curve (not necessarily parametrized by arc length). Then:*

- *T is the unit tangent vector of $\gamma(t)$, points in the direction of motion on the curve, and is defined as*

$$\frac{\gamma'(t)}{\|\gamma'(t)\|}.$$

- N is the unit normal vector of $\gamma(t)$, points in the direction that the tangent vector is changing, and is defined as

$$\frac{T'(t)}{\|T'(t)\|}.$$

- B is the unit binormal vector of $\gamma(t)$, points in the remaining orthogonal direction, or the direction which no motion is occurring, and is defined as

$$T \times N.$$

We can think of T , N , and B as uniquely determined positively oriented orthonormal basis vectors which span \mathbb{R}^3 .

EXAMPLE 2.2.1. Consider the curve $\gamma(t) = \langle \cos(t), \sin(t), t \rangle$. Then

$$\gamma'(t) = \langle -\sin(t), \cos(t), 1 \rangle$$

$$\|\gamma'(t)\| = \sqrt{2}$$

$$T = \frac{1}{\sqrt{2}} \langle -\sin(t), \cos(t), 1 \rangle$$

$$T'(t) = \frac{1}{\sqrt{2}} \langle -\cos(t), -\sin(t), 0 \rangle$$

$$\|T'(t)\| = \frac{1}{\sqrt{2}}$$

$$N(t) = \frac{1}{2} \langle -\cos(t), -\sin(t), 0 \rangle$$

$$B(t) = \left\langle \frac{1}{2\sqrt{2}} \sin(t), -\frac{1}{2\sqrt{2}} \cos(t), \frac{1}{2\sqrt{2}} \right\rangle$$

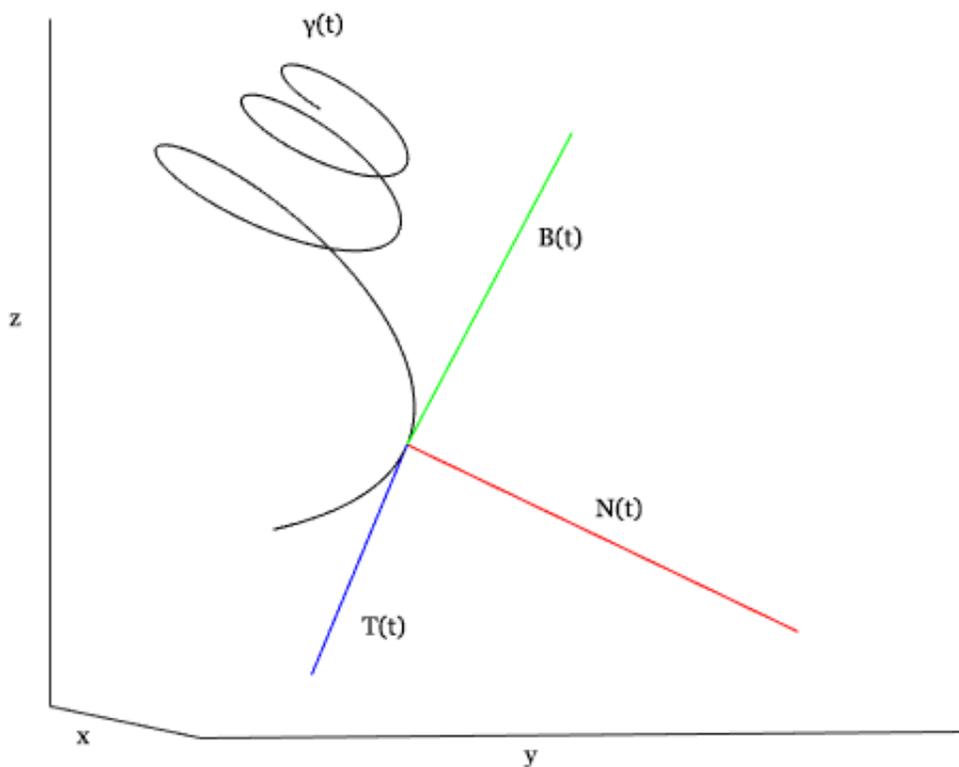


FIGURE 2.2. Three dimensional curve, $\gamma(t) \subset \mathbb{R}^3$ (black), with the unit tangent (blue), unit normal (red), and unit binormal (green) vector at a single instance, $\gamma(t_0)$ on the curve.

2.2.1. CURVATURE. The primary subject of interest in this dissertation involves the derivation, and subsequent use, of curvature and the higher dimensional equivalents (torsion, etc.). These properties of curves are well-known and established for continuous functions. Here, we introduce the subject of curvature using the notation and properties of curves we have just established.

DEFINITION 2.2.2. Let $\gamma \subset \mathbb{R}^n$ be a smooth curve parametrized by arc length, s . The curvature, κ , of γ is defined as

$$\kappa = \left\| \frac{dT}{ds} \right\|$$

where T is the unit tangent vector.

EXAMPLE 2.2.2. The arc length parametrization of a circle (in \mathbb{R}^2) is

$$\gamma(s) = \left\langle a \cos\left(\frac{s}{a}\right), a \sin\left(\frac{s}{a}\right) \right\rangle.$$

It is easy to verify $\|\gamma'(s)\| = 1$ and therefore

$$T(s) = \gamma'(s) = \left\langle -\sin\left(\frac{s}{a}\right), \cos\left(\frac{s}{a}\right) \right\rangle.$$

It follows that

$$\frac{dT}{ds} = \left\langle -\frac{1}{a} \cos\left(\frac{s}{a}\right), -\frac{1}{a} \sin\left(\frac{s}{a}\right) \right\rangle.$$

and therefore

$$\begin{aligned} \kappa &= \left\| \frac{dT}{ds} \right\| \\ &= \frac{1}{a} \end{aligned}$$

In other words, the curvature of a circle is the inverse of the circle's radius.

We can think about the concept of curvature using the same idea: the curvature of a curve, γ , at a point, s_0 , is the inverse radius of the circle of best fit touching the point s_0 . We also make a special note here that since $\kappa = \frac{1}{r}$ where r is the radius of the circle of best fit, this implies $\kappa \geq 0$.

THEOREM 2.2.3. A regular curve in \mathbb{R}^2 has non-zero constant curvature κ if and only if it is part of a circle of radius $\frac{1}{|\kappa|}$.

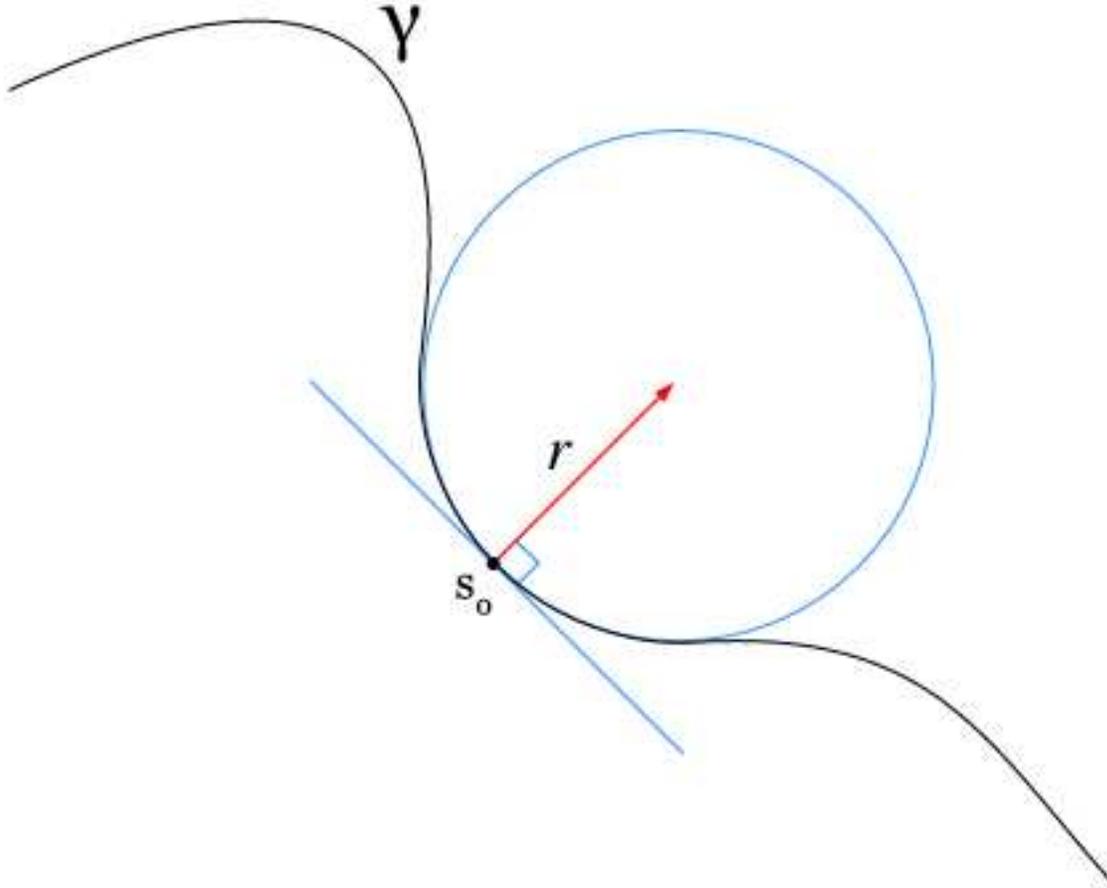


FIGURE 2.3. Curve, γ , with curvature defined at the point s_0 to be $\frac{1}{r}$ of the osculating circle. Modified from Image source: [1]

DEFINITION 2.2.4. *Two functions have a contact of order k at point s_0 if they have the same value and equal derivatives, up to derivative of order k .*

From Theorem 2.2.3, we get the geometric interpretation of curvature. Additionally, we can use this opportunity to define the osculating circle [7]:

DEFINITION 2.2.5. *Given a curve with non-vanishing curvature, the osculating circle of γ at the point $\gamma(s_0)$ is the circle centered at $\gamma(s_0) + \frac{1}{\kappa(s_0)}N(s_0)$.*

We make the note that this circle is uniquely determined by having the property it has contact of order two with the curve. Furthermore, with this geometric interpretation, we can also think of curvature as fitting the equation of the form

$$T(s) = (\cos(\alpha(s)), \sin(\alpha(s)))$$

$$N(s) = (-\sin(\alpha(s)), \cos(\alpha(s)))$$

where $\alpha(s)$ needs to be found. This also implies that curvature can be thought of as the deviation of a curve from a straight line. We will make particular use of this observation later in Section 2.3.

For the sake of completion, we mention one more, traditional, formulation for curvature. This is put in terms of the derivatives of the original curve (assuming the curve is in \mathbb{R}^3):

$$\kappa = \frac{|\gamma' \times \gamma''|}{|\gamma'|^3}.$$

Notice, if the curve is not in \mathbb{R}^3 the cross operation is not defined and this formulation cannot be used. This formula is generalized without the constraint of $\gamma \subset \mathbb{R}^3$ in Section 2.3. However, it does allow us to see that curvature is directly related to the derivatives of the curve.

2.2.2. TORSION. We have seen that curvature describes a curve using a two-dimensional structure (a circle). When $\gamma \subset \mathbb{R}^3$, a one-dimensional structure cannot capture all of the information of the curve. Now we introduce the concept of torsion to help define the curve γ in more dimensions.

DEFINITION 2.2.6. Let $\gamma \subset \mathbb{R}^3$ be a smooth curve parametrized by arc length, s . The torsion, τ , of γ is defined as

$$\tau = -N \cdot B'$$

We can think of torsion as measuring the speed of rotation of the binormal vector at a point, s_0 . If $\tau = 0$ then, γ lies in some plane. From this, we get the geometric interpretation of curvature. Additionally, we can use this opportunity to define the osculating sphere:

DEFINITION 2.2.7. Given a curve with non-vanishing curvature and torsion, the osculating sphere of γ at the point $\gamma(s_0)$ is the sphere centered at $\gamma(s_0) + \frac{1}{\kappa(s_0)}N(s_0) - \frac{\kappa'(s_0)}{\tau(s_0)\kappa^2(s_0)}B(s_0)$.

We make the note that this sphere is uniquely determined by having the property it has contact of order three with the curve. Furthermore, with this geometric interpretation, we can also think of curvature as fitting the helix of the form

$$T(s) = (\cos(\alpha(s)), \sin(\alpha(s)), \beta(s))$$

$$N(s) = (-\sin(\alpha(s)), \cos(\alpha(s)), \beta)$$

$$B(s) = (-\cos(\alpha(s)), -\sin(\alpha(s)), 0)$$

where $\alpha(s)$ and $\beta(s)$ needs to be found. In other words, by ensuring at a point s_0 , the value of the curve is equal to the point defined by the equations above, and the derivatives of the curve are equal to the derivatives of the fitted helix, then we can use that helix as a substitution for the curve at that one point. This will allow us to perform computations on this helix, with known equations, instead of requiring continuous information of the curve. From this,

we can think of torsion at a point as the deviation of a curve from the 2-dimensional plane of best fit. This is another important observation we will make use of later.

Once again, and only for the sake of completion, we can also relate torsion to the derivatives of the curve, as well as curvature in the following formula (assuming the curve is in \mathbb{R}^3):

$$\tau = \frac{\langle \gamma' \times \gamma'', \gamma''' \rangle}{\|\gamma' \times \gamma''\|^2} = \frac{\langle \gamma' \times \gamma'', \gamma''' \rangle}{\kappa^2}.$$

Again, if the curve is not in \mathbb{R}^3 the cross operation is not defined. However, this formula is also generalized in Section 2.3 for the case when $\gamma \subset \mathbb{R}^n$.

2.2.3. FRENET FRAME IN THREE DIMENSIONS. The Frenet Frame is a set of spanning orthonormal basis vectors in Euclidean space and is associated with a point on a curve, $\gamma(t)$. This frame is commonly presented in 3 dimensions as the TNB Frame in \mathbb{R}^3 .

The frame formed by these vectors are used in the Frenet-Serret Equations, a system of differential equations which describe the geometric properties of a curve irrespective of rotation and translation. In particular, for \mathbb{R}^3 , the Frenet-Serret Equations are:

$$\begin{aligned}\frac{dT}{ds} &= \kappa N, \\ \frac{dN}{ds} &= -\kappa T + \tau B, \\ \frac{dB}{ds} &= -\tau N\end{aligned}$$

where $\frac{d}{ds}$ is the derivate with respect to the arc length

$$s(t) = \int_0^t \|r'(\sigma)\| d\sigma,$$

where κ is curvature, and τ is torsion. Given the original curve $\gamma(t)$ we can solve the Frenet-Serret Equations for κ and τ at any point t along the curve.

The 3 dimensional case is a common space to work with these concepts in due to the possibility of visualization. However, it is possible to extend these concepts into higher dimensions (i.e. parameters of the curve which measure the deviation of a curve from the best $3d$ -space, $4d$ -space, ..., nd -space). In 1874, Camille Jordan [2] expanded the Frenet Frame and subsequently, the Frenet-Serret equations into higher dimensional Euclidean spaces.

2.3. THE FRENET-SERRET FRAME AND CURVES IN \mathbb{R}^n

DEFINITION 2.3.1. *Let $\gamma(s)$ be a regular parametrized smoothed curve in \mathbb{R}^n which has been parametrized by arc length. If the derivatives $\gamma^{(i)}$ are linearly independent, the curve, γ , is called a Frenet Curve.*

DEFINITION 2.3.2. *Given a Frenet Curve, γ , the Frenet-Serret frame at any point, s_0 , on the curve is determined computing the Frenet vectors by applying the Gram-Schmidt process to the derivatives of γ :*

$$e_1 = \gamma^{(1)}(s_0)$$

$$e_2 = \frac{\mathbb{P}_1 \gamma^{(2)}(s_0)}{\|\mathbb{P}_1 \gamma^{(2)}(s_0)\|}$$

and

$$e_{i+1} = \frac{\mathbb{P}_i \gamma^{(i+1)}(s_0)}{\|\mathbb{P}_i \gamma^{(i+1)}(s_0)\|}$$

where the projector at each step is

$$\mathbb{P}_i = I - e_i e_i^T$$

Note, from this point on, we will be using the above notation to discuss the Frenet vectors. However, it is worth pointing out that, by definition, $T = e_1$, $N = e_2$, and $B = e_3$.

From these definitions we make the note that every Frenet Curve uniquely induces through its Frenet-Serret n -frame, a curve in the Stiefel manifold of all n -frames in \mathbb{R}^n .

The vectors of the n -dimensional Frenet Frame are related to the generalized curvature values in a similar manner with the 3-dimensional case. It is easier to display this system of differential equations in matrix form:

$$(1) \quad \begin{bmatrix} e'_1(s) \\ \vdots \\ \vdots \\ e'_n(s) \end{bmatrix} = \begin{bmatrix} 0 & \kappa_1(s) & & 0 \\ -\kappa_1(s) & \ddots & \ddots & 0 \\ & \ddots & 0 & \kappa_{n-1}(s) \\ 0 & & -\kappa_{n-1}(s) & 0 \end{bmatrix} \begin{bmatrix} e_1(s) \\ \vdots \\ \vdots \\ e_n(s) \end{bmatrix}$$

where

$$e_1(s) = \gamma'(s)$$

and

$$e_j(s) = \frac{\gamma^{(j)}(s) - \sum_{i=1}^{j-1} \langle \gamma^{(j)}, e_i(s) \rangle e_i(s)}{\|\gamma^{(j)}(s) - \sum_{i=1}^{j-1} \langle \gamma^{(j)}, e_i(s) \rangle e_i(s)\|}$$

Additionally, the real-valued functions for generalized curvature are defined by solving the above system of equations. The closed-form solution for these values are:

$$(2) \quad \kappa_j(s) = \frac{\langle e'_j(s), e_{j+1}(s) \rangle}{\|\gamma'(s)\|}$$

As an interesting and relevant side-note, from this system of equations, if the generalized curvatures are constant then these equations have been shown to have the general solutions

$$(3) \quad \gamma_e(t) = \langle a_1 \cos(\alpha_1 t), a_1 \sin(\alpha_1 t), \dots, a_k \cos(\alpha_k t), a_k \sin(\alpha_k t) \rangle$$

where n is even and

$$(4) \quad \gamma_e(t) = \langle a_1 \cos(\alpha_1 t), a_1 \sin(\alpha_1 t), \dots, a_k \cos(\alpha_k t), a_k \sin(\alpha_k t), bt \rangle$$

where n is odd.

In the same way that we can think of curvature at a point being determined by the size of the circle of best fit, and torsion at a point being determined by the size of the helix of best fit, we can think of the i th generalized curvature as being determined by the size of the generalized circle / generalized helix given by Equations 3 and 4.

Finally, we present the Fundamental Theorem of Curve Theory [8], [5], which allows us to construct a curve from generalized curvature values.

THEOREM 2.3.3. *Let $\kappa_i(s)$, $s = 1, \dots, n-1$, $s \in I$, be smooth functions satisfying $\kappa_j(s) > 0$ for $j = 1, \dots, n-2$. Then there exists a generally curved curve with parameter representation $\gamma(s) \in R^n$ such that s is an arc length parameter and given functions $\kappa_i(s)$ are its curvatures. Two oriented curves in the Euclidean space, R^n , having the same curvature functions are congruent under an orientations preserving motion.*

PROOF. Define $\gamma(s_0) = q_0$, $e_1^{(0)}, \dots, e_n^{(0)}$ as the Frenet n -frame of γ at the point q_0 , and define $F(s) = (e_1(s), \dots, e_n(s))^T$. Then the Frenet equations are equivalent to the matrix equation given by Equation 1 which is a system of linear differential equations of first order. We will refer to Equation 1 in simplified terms: $F(s)' = K(s) \cdot F(s)$.

From the existence and uniqueness theorems for solutions of linear differential equations [6] and given $K(s)$ with an initial condition $F(s_0)$, then $F' = K \cdot F$ has a unique solution, $F(s)$, which is defined for all $s \in I$.

Noting that K is a skew-symmetric matrix, and therefore $0 = K + K^T$, we can take

$$(FF^T)' = F'F^T + F(F^T)' = F'F^T + F(F')^T = KFF^T + FF^TK^T$$

Then, the differential equation

$$(FF^T)' = K(FF^T) + (FF^T)K^T$$

when viewed as a differential equation for the unknown function FF^T has a unique solution [6] for the given initial conditions $F(s_0)(F(s_0))^T = E$ (where E denotes the identity matrix). And, due to the uniqueness of the solution, we must have $FF^T = E$ on the entirety of the interval I . Hence, $F(s)$ is an orthogonal matrix as the continuity of the determinant along the interval I is one.

From this, we establish $F(s)$ determines a unique vector-valued function $e_1(s)$. Using the initial conditions

$$\gamma(s_0) = q_0,$$

we can find a unique curve $c(s)$ with

$$\gamma' = e_1$$

by setting

$$\gamma(s) = q_0 + \int_{s_0}^s e_1(t)dt.$$

From the relation given by the Frenet Equations

$$e_1' = \kappa_1 e_2 \neq 0$$

in Equation 1 and $\kappa_1 > 0$ from the statement of the theorem, we establish e_2 as defined by F , must coincide with the second vector of the Frenet n -frame of γ at event point. This pattern continues for each other e_i . Since $F(s)$ represents the Frenet n -frame of γ at each point, and because $F' = KF$, the given functions κ_i coincide with the Frenet curvatures of γ .

$$\gamma' = e_1$$

$$\gamma'' = \kappa_1 e_2$$

$$\gamma''' = (\kappa_1 e_2)' = (-\kappa_1^2 e_1 + \kappa_1' e_2) + \kappa_1 \kappa_2 e_3$$

$$\vdots$$

$$\gamma^{(i)} = (\text{linear combination of } e_1, \dots, e_{i-1}) + \kappa_1 \cdot \kappa_2 \cdot \dots \cdot \kappa_{i-1} e_i$$

Then, from $\kappa_j(s) > 0$ for $j = 1, \dots, n-2$, the orthogonality of F which we extract the e_j 's, and the construction of the derivatives of γ (which involve the concatenation of orthogonal directions for each successive derivate), we establish that $\gamma', \gamma'', \dots, \gamma^{n-1}$ are linearly independent. We note, that without the constraint, $\kappa_j(s) > 0$, this methodology would still function to produce a linearly independent Frenet Frame, but it would only be unique up to the sign on each basis vector.

□

With this theorem and prerequisite background information on curves, curvatures, and the Frenet-Serret apparatus, we have established most of the necessary tools required to understand the remainder of this thesis.

CHAPTER 3

GENERALIZED CURVATURE ANALYSIS VIA SINGULAR VALUES

3.1. INTRODUCTION

Consider a curve $\gamma(t)$ in \mathbb{R}^n .¹ Recall that if $\gamma(t)$ is parameterized by arc length then $\gamma(t)$ is a solution to the differential equation $E' = EK$. We would like to understand the associated frame $e_1(t), \dots, e_n(t)$ and curvature functions $\kappa_1(t), \dots, \kappa_{n-1}(t)$ from a different point of view. Specifically, consider points on the curve within an ϵ -ball centered at a point $s_0 = \gamma(t_0)$. The tangent line at s_0 is approximated by taking the span of two points on $\gamma(t)$ in an ϵ -ball centered at s_0 while the *osculating* plane at s_0 is approximated by taking the span of three points on $\gamma(t)$ in an ϵ -ball centered at s_0 . However, points on the curve in a small ϵ -ball are nearly linear. The value of $\kappa_1(t_0)$ can be seen as a measure of the failure of the linearity of such points. In a similar manner, the value of the second curvature function, $\kappa_2(t_0)$ is a measure of the failure of planarity of points in an ϵ -ball on the curve. This point of view will be considered more closely in the next section through the local singular value decomposition. In order to make this connection, it is helpful to replace the curve with an idealized version which agrees, to high order, with the curve at $\gamma(t_0)$.

3.1.1. LOCAL APPROXIMATION OF CURVES IN \mathbb{R}^3 AND \mathbb{R}^4 . Consider a curve $\gamma(t)$ in \mathbb{R}^3 . The helix of best fit to γ at $\gamma(t_0)$ is the solution to the differential equation $E' = EK_{t_0}$ where K_{t_0} denotes the curvature matrix K evaluated at t_0 . Thus the curvature functions for the helix will be constants $\kappa_1 = \kappa_1(t_0)$ and $\kappa_2 = \kappa_2(t_0)$. The general solution, $g(t)$, to the differential equation, $E' = EK_{t_0}$, has the form

$$g(t) = (a \cos(\alpha t), a \sin(\alpha t), bt) + Constant.$$

¹Note, the majority of this chapter is taken verbatim from [9].

The helix of best fit to $\gamma(t)$ at $\gamma(t_0)$ is given by

$$h(t) = g(t) - g(t_0) + \gamma(t_0).$$

If $\|\gamma^{(1)}(t_0)\| = 1$ then we get the condition that

$$(5) \quad a^2\alpha^2 + b^2 = 1$$

The relationship between the curvature functions of the helix and the parameters a, b, α is:

$$(6) \quad \kappa_1^2 = a^2\alpha^4$$

$$(7) \quad \kappa_2^2 = b^2\alpha^2$$

Following this pattern, if we solve the differential equation $E' = EK_{t_0}$ for a curve $\gamma(t)$ in \mathbb{R}^4 then we obtain a toroidal curve of best fit at $\gamma(t_0)$ of the form

$$h(t) = g(t) - g(t_0) + \gamma(t_0)$$

where

$$g(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t)) + Constant.$$

We can relate a, b, α, β to the curvature functions as

$$(8) \quad \kappa_1^2 = a^2\alpha^4 + b^2\beta^4$$

$$(9) \quad \kappa_1^2\kappa_2^2 = a^2\alpha^6 + b^2\beta^6 - \kappa_1^4$$

$$(10) \quad \kappa_1^2 \kappa_2^2 \kappa_3^3 = a^2 \alpha^8 + b^2 \beta^8 - \kappa_1^2 (\kappa_1^2 + \kappa_2^2)^2$$

where again we have assumed that the curve is parameterized by arc length so

$$(11) \quad a^2 \alpha^2 + b^2 \beta^2 = 1$$

These equations are derived for $\kappa_1, \kappa_2, \kappa_3$ in [5]. Next we give the corresponding equations for curves in \mathbb{R}^5 and \mathbb{R}^6 . The derivation is straightforward but tedious.

3.1.2. CURVATURE RELATIONS $n = 5$. If we solve the differential equation $E' = EK_{t_0}$ for a curve $\gamma(t)$ in \mathbb{R}^5 then we obtain a curve of best fit at $\gamma(t_0)$ of the form

$$h(t) = g(t) - g(t_0) + \gamma(t_0)$$

where

$$g(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t), ct) + Constant.$$

We can relate a, b, c, α, β to the curvature functions as

$$1 = a^2\alpha^2 + b^2\beta^2 + c^2$$

$$\kappa_1^2 = a^2\alpha^4 + b^2\beta^4$$

$$\kappa_1^2\kappa_2^2 = a^2\alpha^6 + b^2\beta^6 - \kappa_1^4$$

$$\kappa_1^2\kappa_2^2\kappa_3^2 = a^2\alpha^8 + b^2\beta^8 - \kappa_1^2(\kappa_1^2 + \kappa_2^2)^2$$

$$\kappa_1^2\kappa_2^2\kappa_3^2\kappa_4^2 = a^2\alpha^{10} + b^2\beta^{10} - \kappa_1^2((\kappa_1^2 + \kappa_2^2 + \kappa_3^2)(\kappa_2^2 + \kappa_3^2) + \kappa_2^2\kappa_3^4)$$

3.1.3. CURVATURE RELATIONS $n = 6$. If we solve the differential equation $E' = EK_{t_0}$ for a curve $\gamma(t)$ in \mathbb{R}^6 then we obtain a curve of best fit at $\gamma(t_0)$ of the form

$$h(t) = g(t) - g(t_0) + \gamma(t_0)$$

where

$$g(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t), c \cos(\delta t), c \sin(\delta t)) + Constant.$$

Letting $F_k = a^2\alpha^k + b^2\beta^k + c^2\delta^k$, we can relate $a, b, c, \alpha, \beta, \delta$ to the curvature functions as

$$1 = F_2$$

$$\kappa_1^2 = F_4$$

$$\kappa_1^2 \kappa_2^2 = F_6 - \kappa_1^4$$

$$\kappa_1^2 \kappa_2^2 \kappa_3^2 = F_8 - \kappa_1^2 (\kappa_1^2 + \kappa_2^2)^2$$

$$\kappa_1^2 \kappa_2^2 \kappa_3^2 \kappa_4^2 = F_{10} - \kappa_1^2 ((\kappa_1^2 + \kappa_2^2 + \kappa_3^2)(\kappa_2^2 + \kappa_3^2) + \kappa_2^2 \kappa_3^4)$$

$$\kappa_1^2 \kappa_2^2 \kappa_3^2 \kappa_4^2 \kappa_5^2 = F_{12} - F_{10}(\kappa_1^2 + \kappa_2^2 + \kappa_3^2 + \kappa_4^2) + F_8(\kappa_1^2 \kappa_3^2 + \kappa_4^2 \kappa_1^2 + \kappa_4^2 \kappa_2^2)$$

3.2. THE LOCAL SINGULAR VALUE DECOMPOSITION

Recall that at each point $\gamma(t) \in \gamma$, the Frenet-Serret frame is determined by applying the Gram-Schmidt process to the vectors $\gamma^{(1)}(t), \gamma^{(2)}(t), \dots, \gamma^{(n)}(t)$ (where $\gamma^{(k)}(t)$ denotes the k^{th} derivative of γ evaluated at t). We denote this ordered orthonormal basis $e_1(t), \dots, e_n(t)$ and let E denote the orthonormal matrix whose columns are the $e_i(t)$. The main intuition behind a local singular value analysis is to exploit the idea that the Frenet-Serret frame may be viewed as finding the subspace of best fit at a point on the curve. We consider the canonical solution of the Frenet-Serret formula where κ_i is assumed to be constant, i.e., the solutions to $E' = EK$ given by Equations (3) and (4) where K is constant. We use an integral formulation of the singular value decomposition, often referred to as the Karhunen-Loève transformation, at a given point on the curve. We then use a Taylor series approximation for $\gamma(t)$ to determine particular eigenvalues of the Karhunen-Loève transformation in the

ϵ -ball. These relationships can be combined with the relationships between the curvature constants and the curve parameters to determine a formula for computing κ_i locally from the singular values of the Karhunen-Loève transformation.

3.2.1. FORMULATION. Broomhead et al showed that the *local* singular value decomposition could be used to compute the topological dimension of a manifold from sampled points lying on the manifold [10]. This provided a powerful tool for many applications that involved modeling data on manifolds. The original setting of [10] concerned the reconstruction of a manifold, via Takens' theorem, from scalar valued time series statistics of a dynamical system on the manifold. The local singular value decomposition is also useful for applying manifold learning algorithms for geometric data analysis, e.g., local models such as charts [11], or global models based on Whitney's embedding theorem [12]. A more detailed discussion may be found in [13, 14].

Following [15, 10], the *mean centered* covariance matrix of $\gamma(t)$ at t is the matrix

$$\bar{C}_\epsilon(t) = \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} (\gamma(s) - \bar{\gamma}_\epsilon(t))(\gamma(s) - \bar{\gamma}_\epsilon(t))^T ds$$

where

$$\bar{\gamma}_\epsilon(t) = \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} \gamma(s) ds$$

However, we will consider the closely related *on the curve* covariance matrix

$$C_\epsilon(t) = \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} (\gamma(s) - \gamma(t))(\gamma(s) - \gamma(t))^T ds$$

By the singular value decomposition, we have a factorization

$$C_\epsilon(t) = U_\epsilon(t)\Sigma_\epsilon(t)U_\epsilon^T(t)$$

where we assume that the diagonal elements in $\Sigma_\epsilon(t)$ are in monotone decreasing order. We call the columns of $U_\epsilon(t)$ the singular vectors of $C_\epsilon(t)$. Note that such singular vectors are only defined up to a factor of ± 1 . Let $U(t) = \lim_{\epsilon \rightarrow 0} U_\epsilon(t)$. The columns of $U(t)$, written $u_1(t), \dots, u_n(t)$, are called the local singular vectors at $\gamma(t)$. In a similar manner, one can define the local singular vectors $\bar{u}_1(t), \dots, \bar{u}_n(t)$ at $\gamma(t)$ by considering the limiting behavior of the singular vectors in the singular value decomposition of $\bar{C}_\epsilon(t)$ as ϵ tends towards zero.

THEOREM 3.2.1. *Let $\gamma : I \rightarrow \mathbb{R}^n$ be a parametric curve of class C^{n+1} , regular of order n . Let $e_1(t), \dots, e_n(t)$ denote the Frenet-Serret frame at $\gamma(t)$. Let $u_1(t), \dots, u_n(t)$ denote the local singular vectors at $\gamma(t)$. Then for $i = 1, \dots, n$, $e_i(t) = \pm u_i(t)$.*

PROOF. Let $\Gamma(t)$ denote the matrix whose columns are $\gamma^{(1)}(t), \dots, \gamma^{(n)}(t)$. The Frenet-Serret frame, $e_1(t), \dots, e_n(t)$, is obtained by applying the Gram-Schmidt process to the columns of $\Gamma(t)$. Thus $e_i(t)$ is a unit vector orthogonal to the span of $\gamma^{(1)}(t), \dots, \gamma^{(i-1)}(t)$ but lying within the span of $\gamma^{(1)}(t), \dots, \gamma^{(i)}(t)$. Let \mathbf{v} be the $n \times 1$ vector whose k^{th} component is $(s-t)^k/k!$. Then $\Gamma(t)\mathbf{v}$ is the n^{th} order Taylor series expansion for $\gamma(s) - \gamma(t)$ at t . Replacing $\gamma(s) - \gamma(t)$ with its Taylor series expansion leads to the n^{th} order approximation

$$C_\epsilon(t) = \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} (\gamma(s) - \gamma(t))(\gamma(s) - \gamma(t))^T ds \approx \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} (\Gamma(t)\mathbf{v})(\Gamma(t)\mathbf{v})^T ds$$

We rewrite this as

$$\Gamma(t) \frac{1}{2\epsilon} \int_{t-\epsilon}^{t+\epsilon} \mathbf{v}\mathbf{v}^T ds \Gamma(t)^T = \Gamma(t) \mathcal{E} \Gamma(t)^T$$

By the definition of \mathcal{E} , we compute that

$$\mathcal{E}_{i,j} = \frac{\epsilon^{i+j}}{i!j!(i+j+1)} \text{ if } i+j \text{ is even and } \mathcal{E}_{i,j} = 0 \text{ if } i+j \text{ is odd.}$$

We can express $\Gamma(t) \mathcal{E} \Gamma(t)^T$ in terms of the columns of $\Gamma(t)$ and the entries of \mathcal{E} as

$$\frac{\epsilon^2}{3}(c_1 c_1^T) + \frac{\epsilon^4}{5} \left(\frac{1}{6} c_1 c_3^T + \frac{1}{4} c_2 c_2^T + \frac{1}{6} c_3 c_1^T \right) + \cdots + \frac{\epsilon^{2k}}{2k+1} \sum_{i=1}^{2k-1} \frac{1}{i!(2k-i)!} c_i c_{2k-i}^T + \cdots$$

where $c_i = \gamma^{(i)}(t)$. As ϵ tends towards zero, this expression behaves more and more like the rank one matrix $\frac{\epsilon^2}{3} c_1 c_1^T$. Noting that $c_1 = \gamma^{(1)}(t)$, thus is a multiple of $e_1(t)$, we get $u_1(t) = \pm e_1(t)$. Let $P_1 = I - e_1(t)e_1(t)^T$. Pre and post multiplying $\Gamma(t) \mathcal{E} \Gamma(t)^T$ with P_1 deflates away all terms involving c_1 . More precisely,

$$P_1 \Gamma(t) \mathcal{E} \Gamma(t)^T P_1 = \frac{\epsilon^4}{5} \left(\frac{1}{4} P_1 c_2 c_2^T P_1 \right) + \cdots + \frac{\epsilon^{2k}}{2k+1} \sum_{i=2}^{2k-2} \frac{1}{i!(2k-i)!} P_1 c_i c_{2k-i}^T P_1 + \cdots$$

As ϵ tends towards zero, this deflated matrix behaves more and more like the rank one matrix $\frac{\epsilon^4}{5} (\frac{1}{4} P_1 c_2 c_2^T P_1)$. Noting that $P_1 c_2 = P_1 \gamma^{(2)}(t)$, we see that $P_1 c_2$ is orthogonal to $\gamma^{(1)}$ and is in the span of $\gamma^{(1)}, \gamma^{(2)}$ thus is a multiple of $e_2(t)$. This leads to $u_2(t) = \pm e_2(t)$. We now pre and post multiply $P_1 \Gamma(t) \mathcal{E} \Gamma(t)^T P_1$ with $P_2 = I - e_2(t)e_2(t)^T$. Note that since $e_1(t)$ is orthogonal to $e_2(t)$, we have $P_2 P_1 = I - e_1(t)e_1(t)^T - e_2(t)e_2(t)^T$. As ϵ tends towards zero, this doubly deflated matrix behaves more and more like the rank one matrix $\frac{\epsilon^6}{7} (\frac{1}{36} P_2 P_1 c_3 c_3^T P_1 P_2)$. Noting that $P_2 P_1 c_3 = P_2 P_1 \gamma^{(3)}(t)$, we see that $P_2 P_1 c_3$ is orthogonal to the span of $\gamma^{(1)}, \gamma^{(2)}$ but in the span of $\gamma^{(1)}, \gamma^{(2)}, \gamma^{(3)}$ thus is a multiple of $e_3(t)$. This leads to $u_3(t) = \pm e_3(t)$. Continuing to deflate away previously found singular vectors, we obtain the relationship $e_i(t) = \pm u_i(t)$ for all i . Note that for this to work, $\mathcal{E}_{i,i}$ must be non-zero and $P_i P_{i-1} \cdots P_1 \gamma^{(i+1)}(t)$ must be non-zero for each i . These conditions are satisfied since $\mathcal{E}_{i,i} = \frac{\epsilon^{2i}}{(2i+1)i!i!}$ and γ is regular of order n thus $\gamma^{(1)}(t), \dots, \gamma^{(n)}(t)$ are linearly independent. \square

The previous theorem considered the relationship between the local singular vectors of a curve and the Frenet-Serret frame of a curve. We now consider the relationship between the local singular values of a curve and values of the curvature functions. More precisely, in the singular value decomposition

$$C_\epsilon(t) = U_\epsilon(t)\Sigma_\epsilon(t)U_\epsilon^T(t)$$

we considered the limiting behavior of $U_\epsilon(t)$, as ϵ tends towards zero, in order to obtain the local singular vectors. We now consider the limiting behavior of $\Sigma_\epsilon(t)$ as ϵ tends towards zero. Note that the entries of $\Sigma_\epsilon(t)$ are the eigenvalues of $C_\epsilon(t)$ and that they tend towards zero as ϵ tends towards zero. Let $\lambda_{i,\epsilon}(t)$ denote the i^{th} diagonal entry of $\Sigma_\epsilon(t)$. We show that for some constant c_i , we can write

$$\lambda_{i,\epsilon}(t) = c_i\epsilon^{2i} + O(\epsilon^{2i+2})$$

The local singular values of $\gamma(t)$ are then defined as $\sigma_i(t) = \sqrt{c_i}\epsilon^i$.

In Section 2, we have explicitly expressed the curvature, for curves with constant curvature functions, in terms of the parameters of the curves. We now express the leading terms of the eigenvalues $\lambda_{i,\epsilon}(t)$ in terms of the parameters of the curves. This allows us to derive a relationship of the form

$$\kappa_i^2(t) = a_i \lim_{\epsilon \rightarrow 0} \frac{\lambda_{i+1,\epsilon}(t)}{\lambda_{1,\epsilon}(t)\lambda_{i,\epsilon}(t)}$$

where a_i is a constant with known value. From this we obtain

$$\kappa_i(t) = \sqrt{a_i} \frac{\sigma_{i+1}(t)}{\sigma_1(t)\sigma_i(t)}$$

3.2.2. TWO DIMENSIONS. Consider a two dimensional curve with constant curvature $\kappa_1 = 1/a$. This will be a circle of radius a . Up to translation, its parameterized form is $\gamma(s) = (a \cos(\alpha s), a \sin(\alpha s))$. If we assume that the circle is parameterized by arc length then we obtain the constraint $a^2 \alpha^2 = 1$. The components of the covariance matrix $C_\epsilon(0)$ are:

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

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

with

$$C_{12} = C_{21} = \frac{1}{2\epsilon} \int_{-\epsilon}^{\epsilon} (a \cos(\alpha s) - a) \sin(s) ds = 0$$

since the integrand is an odd function.

We follow the usual convention of ordering the eigenvalues by decreasing magnitude so

$$\lambda_{1,\epsilon}(0) = \frac{1}{3} a^2 \alpha^2 \epsilon^2 + O(\epsilon^4)$$

$$\lambda_{2,\epsilon}(0) = \frac{1}{20} a^2 \alpha^4 \epsilon^4 + O(\epsilon^6)$$

$$(12) \quad \lim_{\epsilon \rightarrow 0} \frac{\lambda_{2,\epsilon}(0)}{\lambda_{1,\epsilon}^2(0)} = \frac{9}{20a^2}$$

Given that the curvature $\kappa_1 = 1/a$, we obtain the following expression for κ_1 in terms of the local singular values of the circle:

$$(13) \quad \kappa_1 = \sqrt{\frac{20}{9} \frac{\sigma_2}{\sigma_1^2}} = \frac{\sqrt{20}}{3} \frac{\sigma_2}{\sigma_1^2}$$

3.2.3. THREE DIMENSIONS. Here we consider curves in \mathbb{R}^3 with constant κ_1, κ_2 . Up to translation, such a curve will have the form

$$\gamma(s) = (a \cos(\alpha s), a \sin(\alpha s), bs)$$

Assuming the curve is parameterized by arc length we have $a^2\alpha^2 + b^2 = 1$. The covariance matrix, $C_\epsilon(t)$, is a 3×3 matrix with eigenvalues

$$\lambda_1 = \frac{1}{3}\epsilon^2 + O(\epsilon^4)$$

$$\lambda_2 = \frac{1}{20}a^2\alpha^4\epsilon^4 + O(\epsilon^6)$$

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

Recalling from Section 2 the equations for κ_1, κ_2 in terms of the parameters a, α, b , we obtain

$$(14) \quad \kappa_1^2 = \frac{20}{9} \lim_{\epsilon \rightarrow 0} \frac{\lambda_{2,\epsilon}(t)}{\lambda_{1,\epsilon}^2(t)} \quad \kappa_2^2 = \frac{105}{4} \lim_{\epsilon \rightarrow 0} \frac{\lambda_{3,\epsilon}(t)}{\lambda_{1,\epsilon}(t)\lambda_{2,\epsilon}(t)}$$

This leads to the expression of κ_1, κ_2 in terms of the singular values as:

$$\kappa_1 = \frac{\sqrt{20}}{3} \frac{\sigma_2}{\sigma_1^2} \quad \text{and} \quad \kappa_2 = \frac{\sqrt{105}}{2} \frac{\sigma_3}{\sigma_1\sigma_2}$$

3.2.4. FOUR DIMENSIONS. Here we consider curves in \mathbb{R}^4 with constant $\kappa_1, \kappa_2, \kappa_3$. Up to translation, such a curve will have the form

$$\gamma(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t))$$

This leads to the following formulas:

$$1 = a^2 \alpha^2 + b^2 \beta^2$$

$$\lambda_1 = \frac{1}{3} \epsilon^2 + O(\epsilon^4)$$

$$\lambda_2 = \frac{1}{20} a^2 \alpha^4 + b^2 \beta^4 \epsilon^4 + O(\epsilon^6)$$

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

$$\lambda_4 = \frac{1}{63504} \frac{a^2 b^2 \alpha^4 \beta^4 (\alpha^2 - \beta^2)^2}{a^2 \alpha^4 + b^2 \beta^4} \epsilon^8 + O(\epsilon^{10})$$

Using elimination theory we establish the following representations of the κ_i in terms of the local singular values:

$$\kappa_1 = \frac{\sqrt{20}}{3} \frac{\sigma_2}{\sigma_1^2}, \quad \kappa_2 = \frac{\sqrt{105}}{2} \frac{\sigma_3}{\sigma_1 \sigma_2}, \quad \kappa_3 = \frac{\sqrt{336}}{5} \frac{\sigma_4}{\sigma_1 \sigma_3}$$

3.2.5. PATTERNS IN HIGHER DIMENSIONS. Given that many of the entries of $C_\epsilon(0)$ are odd functions, the covariance matrix has a special structure with many zero entries. For instance, the structure of the covariance matrix for $n = 6$ is

$$\begin{bmatrix} C_{11} & 0 & C_{13} & 0 & C_{15} & 0 \\ 0 & C_{22} & 0 & C_{24} & 0 & C_{26} \\ C_{31} & 0 & C_{33} & 0 & C_{35} & 0 \\ 0 & C_{42} & 0 & C_{44} & 0 & C_{46} \\ C_{51} & 0 & C_{53} & 0 & C_{55} & 0 \\ 0 & C_{62} & 0 & C_{64} & 0 & C_{66} \end{bmatrix}$$

We can permute the columns and rows of this matrix an even number of times to obtain the block matrix

$$\begin{bmatrix} C_{11} & C_{13} & C_{15} & 0 & 0 & 0 \\ C_{31} & C_{33} & C_{35} & 0 & 0 & 0 \\ C_{51} & C_{53} & C_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{22} & C_{24} & C_{26} \\ 0 & 0 & 0 & C_{24} & C_{44} & C_{46} \\ 0 & 0 & 0 & C_{26} & C_{46} & C_{66} \end{bmatrix}$$

Thus we observe the more computationally efficient approach to computing the eigenvalues by computing the eigenvalues of the block submatrices.

3.2.6. FIVE AND SIX DIMENSIONS. First we consider curves in \mathbb{R}^5 with constant $\kappa_1, \kappa_2, \kappa_3, \kappa_4$ which, up to translation, will have the form

$$\gamma(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t), ct)$$

Letting $F_k = a^2\alpha^k + b^2\beta^k$, we obtained the following formulas:

$$1 = a^2\alpha^2 + b^2\beta^2 + c^2$$

$$\lambda_1 = \frac{1}{3}\epsilon^2 + O(\epsilon^4)$$

$$\lambda_2 = \frac{1}{20}F_4\epsilon^4 + O(\epsilon^6)$$

$$\lambda_3 = \frac{1}{1575}(a^2b^2\alpha^2\beta^2(\alpha^2 - \beta^2)^2 + c^2F_6)\epsilon^6 + O(\epsilon^8)$$

$$\lambda_4 = \frac{1}{63504} \frac{a^2b^2\alpha^4\beta^4(\alpha^2 - \beta^2)^2}{F_4} \epsilon^8 + O(\epsilon^{10})$$

$$\lambda_5 = \frac{1}{9823275} \frac{a^2b^2\alpha^6\beta^6(\alpha^2 - \beta^2)^2}{a^2b^2\alpha^2\beta^2(\alpha^2 - \beta^2)^2 + c^2F_6} \epsilon^{10} + O(\epsilon^{12})$$

Using elimination theory, we establish the following representations of the κ_i in terms of the local singular values:

$$\kappa_1 = \frac{\sqrt{20}}{3} \frac{\sigma_2}{\sigma_1^2}, \quad \kappa_2 = \frac{\sqrt{105}}{2} \frac{\sigma_3}{\sigma_1\sigma_2}, \quad \kappa_3 = \frac{\sqrt{336}}{5} \frac{\sigma_4}{\sigma_1\sigma_3}, \quad \kappa_4 = \frac{\sqrt{825}}{4} \frac{\sigma_5}{\sigma_1\sigma_4}$$

In a similar manner, for curves in \mathbb{R}^6 of the form

$$\gamma(t) = (a \cos(\alpha t), a \sin(\alpha t), b \cos(\beta t), b \sin(\beta t), c \cos(\delta t), c \sin(\delta t))$$

we obtain these same expressions for $\kappa_1, \kappa_2, \kappa_3, \kappa_4$ plus the additional relationship

$$\kappa_5 = \frac{\sqrt{1716}}{7} \frac{\sigma_6}{\sigma_1\sigma_5}.$$

Throughout this section, we have assumed the curve to be parameterized with respect to arc length. The local computations can still be made without this assumption. What would change in the formulas in the previous section is that we would replace the assumption that $\|\gamma^{(1)}(t_0)\| = 1$ with $\|\gamma^{(1)}(t_0)\| = r$. We obtain the same connection between the higher curvature functions and *ratios* of singular values. We summarize the results of the previous pages in the following:

THEOREM 3.2.2. *Let $\gamma : I \rightarrow \mathbb{R}^n$ be a parametric curve of class C^{n+1} , regular of order n with $n \leq 6$. Let $\kappa_i(t)$ denote the i^{th} curvature function of γ evaluated at t and let $\sigma_i(t)$ denote the i^{th} local singular value of γ at t . For each $t \in I$ and each $i < n$,*

$$\kappa_i(t) = \sqrt{a_i} \frac{\sigma_{i+1}(t)}{\sigma_1(t)\sigma_i(t)} \quad \text{with} \quad a_1 = \frac{20}{9}, a_2 = \frac{105}{4}, a_3 = \frac{336}{25}, a_4 = \frac{825}{16}, a_5 = \frac{1716}{49}$$

It is easy to check that the formula

$$a_{k-1} = \left(\frac{k}{k + (-1)^k} \right)^2 \frac{4k^2 - 1}{3}$$

is consistent with the first 5 values of a_i given above. Perhaps surprisingly, the numerator of this series arises in the number of Kekulé structures in benzenoid hydrocarbons [16] and the degrees of projections of rank loci [17]. We suspect Theorem 3.2.2 holds more generally. In particular, we make the following conjecture:

CONJECTURE 3.2.3. *Let $\gamma : I \rightarrow \mathbb{R}^n$ be a parametric curve of class C^{n+1} , regular of order n . Then for each $k \leq n$,*

$$(15) \quad \kappa_{k-1}(t) = \frac{k}{k + (-1)^k} \sqrt{\frac{4k^2 - 1}{3}} \frac{\sigma_k(t)}{\sigma_1(t)\sigma_{k-1}(t)}$$

Theorem 3.2.2 shows that the conjecture is true for $\kappa_1, \dots, \kappa_5$. We have numerically verified the conjecture for $\kappa_6, \kappa_7, \kappa_8$. This was done by generating curves with prescribed non-constant curvature and solving the system $E' = EK$ numerically. Then, the local singular values were numerically approximated from the numerically generated curves.

3.3. ALGORITHM: CURVATURE ESTIMATION ON DISCRETELY SAMPLED SMOOTH CURVE

The theoretical contributions of this chapter provide the framework necessary to develop an algorithm for estimating generalized curvature values given a time-series approximation of a curve. Algorithm 1 contains pseudocode related to this algorithm. A full MATLAB implementation is given in Appendix B.

Algorithm 1 Curvature on Discretely Sampled Smooth Curve

```
procedure DOUBLE CURVATURESMOOTH(TIMESERIES TS[1...N], DOUBLE PERCENT-  
AGE)  
  AL =  $\sum_{i=1}^{N-1}$  of d(ts[i],ts[i+1]) // d is the l2-distance measure  
  for int i from 1 to N do  
    w1 = argmax(arlength(ts[i-w1]...ts[i]))  $\leq$  AL*percentage  
    w2 = argmax(arlength(ts[i]...ts[i+w2]))  $\leq$  AL*percentage  
    window = [ts[i-w1]...ts[i]...ts[i+w2]]  
    for int j from 1 to length(window) do  
      window[j] = window[j]-ts[i]  
    end for  
    window = window*(windowT / size(windowT, 1)  
    E = Matlab::flipud(Matlab::eig(window))  
  end for  
  dimensionOfData = size(ts,1)  
  for int i from 1 to dimensionOfData do  
    constantList[i] =  $\frac{i+1}{i+1 * (-1)^i} * \sqrt{\frac{((4(i+1)^2 - 1))}{3}}$   
  end for  
  Initialize GC  
  for int i from 1 to N do  
    for int j from 1 to dimensionOfData-1 do  
      GC[j,i] = constantList[j] *  $\sqrt{\frac{E[j+1,i]}{E[1,i] * E[j,i]}}$   
    end for  
  end for  
  return GC  
end procedure
```

3.4. AN EXAMPLE

We consider the twisted cubic curve in \mathbb{R}^3 given parametrically as $\gamma(t) = [t, t^2, t^3]$. The Frenet-Serret frame can be shown to be:

$$e_1(t) = \begin{bmatrix} \frac{1}{\sqrt{1+4t^2+9t^4}} \\ \frac{2t}{\sqrt{1+4t^2+9t^4}} \\ \frac{3t^2}{\sqrt{1+4t^2+9t^4}} \end{bmatrix} \quad e_2(t) = \begin{bmatrix} \frac{t(2+9t^2)}{\sqrt{1+4t^2+9t^4}\sqrt{1+9t^2+9t^4}} \\ \frac{1-9t^4}{\sqrt{1+4t^2+9t^4}\sqrt{1+9t^2+9t^4}} \\ \frac{3t+6t^3}{\sqrt{1+4t^2+9t^4}\sqrt{1+9t^2+9t^4}} \end{bmatrix} \quad e_3(t) = \begin{bmatrix} \frac{3t^2}{\sqrt{1+9t^2+9t^4}} \\ \frac{-3t}{\sqrt{1+9t^2+9t^4}} \\ \frac{1}{\sqrt{1+9t^2+9t^4}} \end{bmatrix}$$

while the functions $\kappa_1(t), \kappa_2(t)$ can be shown to be

$$\kappa_1(t) = \frac{2\sqrt{1 + 9t^2 + 9t^4}}{(1 + 4t^2 + 9t^4)^{3/2}} \quad \kappa_2(t) = \frac{3}{1 + 9t^2 + 9t^4}$$

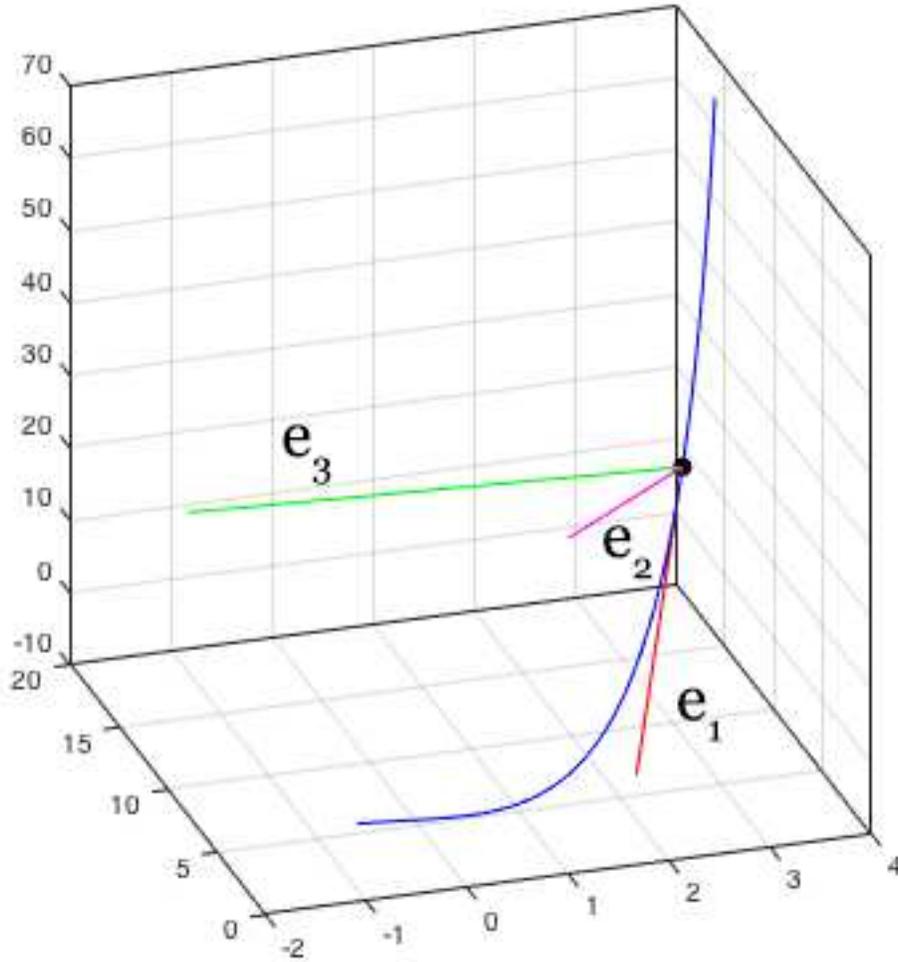


FIGURE 3.1. Twisted Cubic example $-1 \leq t \leq 4$. Point $t = 3$ (black dot). First singular vector (red). Second singular vector (magenta). Third singular vector (green).

Let $\epsilon = .001$ and let $t = 3$. If we consider the singular value decomposition $C_\epsilon(t) = U_\epsilon(t)\Sigma_\epsilon(t)U_\epsilon^T(t)$ for $\gamma(t)$ then we can consider the singular vectors of $C_\epsilon(t)$ as a proxy for the

local singular vectors of $\gamma(t)$ at $t = 3$ and compare to the exact value for $e_i(t)$ at $t = 3$. For instance, comparing the first singular vector to the first frame vector, we get

$$u_{1,\epsilon}(3) = \begin{bmatrix} .036131465 \\ .216788800 \\ .975549656 \end{bmatrix} \quad e_1(3) = \begin{bmatrix} .036131468 \\ .216788812 \\ .975549654 \end{bmatrix}$$

The other singular vectors, $u_{2,\epsilon}(3), u_{3,\epsilon}(3)$ are similarly close to $e_2(3), e_3(3)$. If we consider

$$\sqrt{a_i} \frac{\sqrt{\lambda_{i+1,\epsilon}(t)}}{\sqrt{\lambda_{1,\epsilon}(t)}\sqrt{\lambda_{i,\epsilon}(t)}} \text{ as a proxy for } \kappa_i = \sqrt{a_i} \frac{\sigma_{i+1}(t)}{\sigma_1(t)\sigma_i(t)}$$

then we obtain the following estimates:

$$\kappa_1(3) \approx .0026865640, \quad \kappa_2(3) \approx .0036991369$$

whereas using the exact formulas, we can compare these values to

$$\kappa_1(3) = .0026865644\dots, \quad \kappa_2(3) = .0036991368\dots$$

For these approximations, we used $\epsilon = 10^{-3}$. With a choice of $\epsilon = 10^{-6}$, for this example, we observed about 13 digits of accuracy. This example illustrates how the theorems of the previous section can be used to obtain very good approximations of both the Frenet-Serret frame and values of the curvature functions by considering small values of ϵ .

3.5. CONCLUSION

In this chapter, we established the close connection between the Frenet-Serret apparatus and the local singular value decomposition of regular curves in \mathbb{R}^n . The local singular value

decomposition was defined as the limit of the singular value decomposition of a family of covariance matrices defined on the curve. In particular, we showed in Theorem 3.2.1 that the Frenet-Serret frame and the local singular vectors of regular curves in \mathbb{R}^n agree (up to a factor of ± 1). In addition we showed in Theorem 3.2.2 that values of each of the curvature functions can be expressed in terms of ratios of local singular values for regular curves in \mathbb{R}^n with $n \leq 6$. Conjecture 3.2.3 concerns an extension of Theorem 3.2.2 to arbitrary dimension. We have numerically checked this conjecture for curves lying in \mathbb{R}^n with $n \leq 9$. The techniques also allow for highly accurate approximations for the Frenet-Serret apparatus.

CHAPTER 4

A PRACTICAL METHOD FOR ESTIMATING GENERALIZED CURVATURES FROM NOISY SAMPLE CURVES IN N-DIMENSIONS

4.1. INTRODUCTION

The traditional method for estimating curvature from discrete samples is to compute local derivatives from point differences, and then compute curvature from the derivatives.¹ This method is highly susceptible to noise. Solis was the first to suggest that curvature could be estimated from local singular values [19]. In Chapter 3 we reached a similar conclusion by a different path [20], and our work yielded both a convergence proof and an equation for estimating curvature from local samples. Curvature results in Chapter 3 were theoretical in nature, and not presented as a practical computational method of estimating curvature. It estimates curvature at points known to lie on the curve and assumes the optimal window size is fixed. Neither of these assumptions can be assumed in real-world situations. This chapter presents a practical method for estimating curvature assuming the data points are noisy samples, and for adjusting the scale of the data window to the underlying curvature [21]. Combined, these methods create Algorithm 4, the first SVD-based approach for accurately estimating curvatures from noisy high dimensional points.

The end of this chapter compares Algorithm 4 to other curvature estimation approaches on synthetic data. We show that Algorithm 4 is 10 times more accurate than Algorithm 1, and 40 times more accurate than estimating derivatives with local point differences, Algorithm 5 (which is also described in this chapter).

¹Note, parts of this chapter have been taken verbatim from [18].

4.2. CURVATURE FROM EIGENVALUES

The curvature of any n -dimensional curve may be determined by finding the circle of best fit to the curve in an infinitesimally small region (see Chapter 2). This approximation is known in differential geometry as the *osculating circle*. It resides in the plane of best fit to the data, also referred to as the *osculating plane*. The Karhunen-Loève decomposition allows one to determine an optimal basis where the data correspond to points in a function space. The theory is analogous to Principal Component Analysis for determining best approximations to sampled data; see, e.g., [13] and references therein. In this section we establish a formula for curvature modeling our data as values on a curve $\gamma(t)$ exploring n -dimensional space where t is continuous. We then show how this can be converted into a robust algorithm for discrete data which is optimized using an adaptive window scheme.

4.2.1. FORMULATION. We are motivated by the fact that the osculating circle determines the curvature at a point on a given n -dimensional curve. Our formulation exploits the fact that the circle of interest resides in a plane of best fit to the data locally. This observation permits us to simplify our theoretical considerations to the two-dimensional setting.

The circle is parameterized by t via the function

$$r : [0, 2\pi] \rightarrow \mathbb{R}^2$$

where

$$\gamma(t) = (a \cos(\alpha t), a \sin(\alpha t)).$$

Our analysis starts at the arbitrary point on the circle

$$(a \cos(\alpha t_0), a \sin(\alpha t_0))$$

where it is assumed that the segment of interest is local in the sense that $t \in [t_0 - \epsilon, t_0 + \epsilon]$.

The mean value of any curve in an ϵ -ball about this point is given by

$$\bar{\gamma} = \frac{1}{2\epsilon} \int_{t_0-\epsilon}^{t_0+\epsilon} \gamma(t) dt$$

and hence for the circle

$$\bar{\gamma} = \left(\frac{a \sin(\alpha\epsilon) \cos(\alpha t_0)}{\alpha\epsilon}, 0 \right)$$

The components of the *curve-mean* centered covariance matrix C defined along the interval $t \in [t_0 - \epsilon, t_0 + \epsilon]$ can be written as

$$C_{ij} = \frac{1}{2\epsilon} \int_{t_0-\epsilon}^{t_0+\epsilon} (\gamma_i(t) - \bar{\gamma}_i)(\gamma_j(t) - \bar{\gamma}_j) dt$$

We shall show that evaluating the eigenvalues of C in an ϵ ball on this curve will provide a relationship between the eigenvalues and curvature κ for n -dimensional curves. In this setting the circle of best fit, residing in the plane of best fit, can be used to characterize curvature for a curve in n -dimensional space.

We proceed by computing the mean centered covariance matrix on the osculating circle.

On the diagonal we have

$$C_{11} = \frac{1}{2\epsilon} \int_{t_0-\epsilon}^{t_0+\epsilon} \left(a \cos(\alpha t) - \frac{a \sin(\alpha\epsilon) \cos(\alpha t_0)}{\alpha\epsilon} \right)^2 dt$$

$$C_{22} = \frac{1}{2\epsilon} \int_{t_0-\epsilon}^{t_0+\epsilon} a^2 \sin^2(\alpha t) dt$$

The off diagonal terms are especially simple, i.e.,

$$C_{12} = C_{21} = \frac{1}{2\epsilon} \int_{t_0-\epsilon}^{t_0+\epsilon} \left(a \cos(\alpha t) - \frac{a \sin(\alpha\epsilon) \cos(\alpha t_0)}{\alpha\epsilon} \right) \sin(\alpha t) dt = 0$$

since the integrand is an odd function.

Given that the covariance matrix is diagonal, 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

$$\lambda_1 = \frac{1}{3}a^2\alpha^2\epsilon^2 + O(\epsilon^4)$$

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

$$(16) \quad \lim_{\epsilon \rightarrow 0} \frac{\lambda_2}{\lambda_1^2} = \frac{1}{5} \frac{1}{a^2}$$

Hence, given the curvature κ

$$\kappa = \frac{1}{a}$$

we obtain the expression for curvature in terms of the eigenvalues of the covariance matrix in the limit, i.e.,

$$(17) \quad \kappa^2 = 5 \lim_{\epsilon \rightarrow 0} \frac{\lambda_2}{\lambda_1^2}.$$

In the next section we outline how to adapt this formula to a practical algorithm for determining the curvature of an n -dimensional curve. We forgo the formulation of higher dimensional generalized curvature values since they parallel the formulations in Chapter 3. However, we show results of higher dimensional computations in Section 4.3.

4.2.2. THE CURVATURE ALGORITHM. The formula given in Equation 17 is theoretical. However, it provides the ingredients for a robust algorithm for computing curvature for an

n -dimensional curve. The formula was derived in the setting of a shrinking ϵ -ball about a point t_0 where the domain of the curve is a continuous variable. In practice, we can't actually achieve $\epsilon \rightarrow 0$ on a computer so we have to implement the usual discretization and associate this with the application.

Assume we have P samples of the curve $\gamma(t)$ collected at times $t_i, i = 1, \dots, P$, i.e., $\{\gamma(t_i)\}$. We will assume that the indices of the first and last points in the i th ball are given by l_i and r_i , respectively, corresponding to times t_{l_i} and t_{r_i} . The ball is taken to be centered at the discrete mean of the points in this interval $t_i \in [t_{l_i}, \dots, t_{r_i}]$ denoted by $\bar{\gamma}(t_i)$.

We define the local data matrix as

$$X(t_i) = [\gamma(t_{l_i}) - \bar{\gamma}(t_i) \mid \dots \mid \gamma(t_i) - \bar{\gamma}(t_i) \mid \dots \mid \gamma(t_{r_i}) - \bar{\gamma}(t_i)]$$

Once l_i and r_i have been determined we may compute the eigenvalues of the matrix

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

i.e., the discretization of the equation for the covariance matrix C evaluated via integration in the previous section. The eigenvalues in our formula for curvature then come from the eigenvector problem

$$C(t_i)e = \lambda e.$$

Our formula for curvature κ requires the first two eigenvalues λ_1 and λ_2 , associated with the eigenvectors e_1 and e_2 spanning the *osculating plane*. The question that needs to be answered is how to actually select the window size. We address this below.

4.3. GENERALIZED CURVATURE FROM NOISY POINTS

The main contribution of this section is the extension of the method created in Chapter 3 (Algorithm 1) for the purpose of computing generalized curvature values for noisy data where the underlying curve is not known. To extend this method to the case of noisy data, we introduce the following modification. Assume we are computing the curvature at time t_i . Now we view the Frenet frame as being centered at the point $\bar{\gamma}(t_i)$, the row average of the points $[\gamma(t_l) | \dots | \gamma(t_i) | \dots | \gamma(t_r)]$ sampled at times $t_l, \dots, t_i, \dots, t_r$.

We define the local data matrix as

$$X(t_i) = [\gamma(t_{l_i}) - \bar{\gamma}(t_i) | \dots | \gamma(t_i) - \bar{\gamma}(t_i) | \dots | \gamma(t_{r_i}) - \bar{\gamma}(t_i)]$$

This modification does not change the directions of any of the Frenet bases vectors. However, it changes the constants in front of the ratios of eigenvalues. We can show through analytical methods that the constants for $\kappa_1, \dots, \kappa_5$ are $\sqrt{5}$, $\sqrt{35/3}$, $\sqrt{21}$, $\sqrt{33}$, and $\sqrt{\frac{429}{9}}$, respectively. From the Fundamental Theorem of Curves [22] we know that given a set of curvatures, a unique curve is formed up to rotation and translation. Using this to construct a curve with known generalized curvature values and the numerically computed square root of eigenvalue ratios of this curve, we numerically solved for these constants. Additionally, we are able to establish the numerical constants up to κ_8 , which suggests the following formula for generalized curvature using an average of points:

$$(18) \quad \kappa_{j-1} = \sqrt{\frac{4j^2 - 1}{3}} \sqrt{\frac{\lambda_j}{\lambda_1 \lambda_{j-1}}}$$

Note that this equation is consistent with the analytical values for $\kappa_1, \dots, \kappa_5$. More importantly, by centering the data around the mean of the samples rather than on a point on

the actual curve, this equation makes it possible to compute curvatures at t_i using Equation 18, without knowing the position of the curve at t_i *a-priori*. Section 4.5 will illustrate this method significantly reduces noise in the curvature estimates, since sampled points are rarely exactly on the curve.

4.3.1. **ALGORITHM: ESTIMATING CURVATURE ON NOISY SAMPLED CURVE.** The theoretical contributions of this chapter provide the framework necessary to develop an algorithm for estimating generalized curvature values given a noisy time-series approximation of a curve. Algorithm 2 contains pseudocode related to this algorithm. A full MATLAB implementation is given in Appendix B.

Algorithm 2 Curvature on Discretely Sampled Curve

```
1: procedure DOUBLE_CURVATURENOISY(TIMESERIES TS[1...N], DOUBLE PERCENT-
   AGE)
2:   AL =  $\sum_{i=1}^{N-1}$  of d(ts[i],ts[i+1]) d is the l2-distance measure
3:   for int i from 1 to N do
4:     w1 = argmax(arclength(ts[i-w1]...ts[i]))  $\leq$  AL*percentage
5:     w2 = argmax(arclength(ts[i]...ts[i+w2]))  $\leq$  AL*percentage
6:     window = [ts[i-w1]...ts[i]...ts[i+w2]]
7:     for int j from 1 to length(window) do
8:       window[j] = window[j]-mean(window)
9:     end for
10:    window = window(windowT) / size(windowT, 1)
11:    E = Matlab:flipud(Matlab::eig(window))
12:  end for
13:  dimensionOfData = size(ts,1)
14:  for int i from 1 to dimensionOfData do
15:    constantList[i] =  $\sqrt{\frac{(2(i-1)+3)(2(i-1)+5)}{3}}$ 
16:  end for
17:  Initialize GC
18:  for int i from 1 to N do
19:    for int j from 1 to dimensionOfData-1 do
20:      GC[j,i] = constantList[j] *  $\sqrt{\frac{E[j+1,i]}{E[1,i] * E[j,i]}}$ 
21:    end for
22:  end for
   return GC
23: end procedure
```

4.4. OPTIMIZING DATA WINDOWS

We have outlined a procedure above for determining the curvature of an n -dimensional curve that requires the computation of the eigenvalues associated with data in a local ball. In practice, we anticipate that the optimal size of a discrete data ball will depend on the curvature. If the curvature is small at $\gamma(t_i)$, then we expect to be able to include more data by extending the radius of the ball. In contrast, for large curvatures, the size of the ball will need to be reduced. Experiments suggest that adapting the size of the data ball to reflect curvature leads to better performance of the method.

Hence, instead of using a fixed window size, we present an automated method for computing an adaptive window size at every discrete time t_i . Our goal is to determine the integer values l_i^* and r_i^* which will be used in the formation of a local window, from $[\gamma(t_{l_i^*}) | \dots | \gamma(t_{r_i^*})]$, to be used in the local principal component analysis. For numerical reasons we compute the SVD of X and use the singular values squared, i.e., $\lambda_i = \sigma_i^2$.

A natural basis for determining the window size is given by the first b left singular vectors $e_1(t_i), e_2(t_i), \dots, e_b(t_i)$ of $X(t_i)$, i.e.,

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

The idea is that we are going to let the window size around the point $\gamma(t_i)$ grow as we compare the ratio between the distance of the furthest point(s) in the window from the space spanned by $\{e_1(t_i), e_2(t_i), \dots, e_b(t_i)\}$ and the distance from the projected point. This plane is a best approximation to the *osculating plane* and is illustrated in Figure 4.1.

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

$$p_l = \gamma(t_l) - \bar{\gamma}(t_i)$$

$$p_r = \gamma(t_r) - \bar{\gamma}(t_i)$$

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

Now we construct the projection matrix onto the osculating plane, i.e., the range of the $n \times b$ matrix $E_b = [e_1(t_i) | e_2(t_i) | \dots | e_b(t_i)]$ as $\mathbb{P} = E_b E_b^T$. The desired index values are found

by solving the optimization problems

$$l^* = \arg \min_l \left| \frac{(I - \mathbb{P})p_l}{\mathbb{P}p_l + \epsilon_{\text{rat}}} - h \right|$$

$$r^* = \arg \min_r \left| \frac{(I - \mathbb{P})p_r}{\mathbb{P}p_r + \epsilon_{\text{rat}}} - h \right|$$

where $h > 0$ is a suitably chosen cutoff which ensures that the set of points

$$\{\gamma(t_{l^*}), \dots, \gamma(t_{r^*})\}$$

represents a local region of the curve γ . ϵ_{rat} ensures if the curve is locally linear, we extend beyond that region. Through extensive empirical tests, we have found that any value of γ in the range $0.05 \leq h \leq 0.5$ produces robust bounding intervals and $\epsilon_{\text{rat}} = 10^{-4}$. These optimization problems are solved at each point t_i for l_i^* and r_i^* . Once we have the appropriate window size for each point along the curve the eigenvalues are recomputed and the curvature at time t_i , i.e., $\kappa(t_i)$ is estimated.

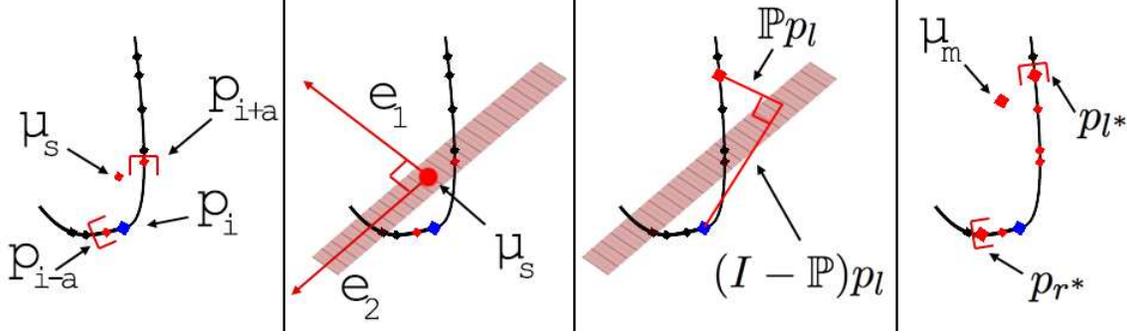


FIGURE 4.1. Illustration of the adaptive windowing algorithm. (Left) To compute the window size to use around point p_i , select a window using an equal and fixed number of points surrounding p_i : $[p_{i-a}|\dots|p_i|\dots p_{i+a}]$. Compute the mean, μ_s of the data in this window. (Middle Left) Compute the two-dimensional subspace of best fit, given by the eigenvectors $\{e_1, e_2\}$ of $[p_{i-a} - \mu_s|\dots|p_i - \mu_s|\dots p_{i+a} - \mu_s]$. (Middle Right) Project data points onto the subspace of best fit on the left side of the curve. Compute the ratio of $(I - \mathbb{P})p_l / (\mathbb{P}p_l + \epsilon_{\text{rat}})$ where ϵ_{rat} is a small offset to keep the denominator from vanishing. Compare this to the cutoff value selected. Do the same for points on the other side of p_i . (Right) Use the results of these ratios and the cutoff value to determine the number of points to include in the computation of generalized curvatures. Note, the mean of this window, μ_m is not necessarily the same as μ_s .

4.4.1. ALGORITHM: ADAPTIVE WINDOWING. Algorithm 3 shows pseudocode for selecting the window size for a single point of the time-series.

Algorithm 3 Adaptive Window Size Selector

```
1: procedure DOUBLE CURVATURESMOOTH(TS[1...N], CUTOFFVALUE, FRAMENUM-  
   BER)  
2:   dimension = size(ts,1)  
3:   Initialize L,R // L contains the beginning frame number and R contains the ending  
   frame number for each generalized curvature dimension  
4:   for int k from 1 to dimension-1 do  
5:     tWindow = ts[frameNumber-ceiling(k/2)]...ts[i+ceiling(k/2)]  
6:     Mean-subtract tWindow  
7:     [U,S,V] = Matlab::svd(tWindow)  
8:     // Search over c. d is point to subspace distance.  
9:     leftP(c) = d(ts[fn-c]-ts[i],U[1:k])  
10:    rightP(c) = d(ts[fn+c]-ts[i],U[1:k])  
11:    L[k] = argmin  $\left( \frac{\text{leftP}}{\sqrt{|\text{leftP}^2 - ||ts[fn - c]^2||} + \epsilon}} - \text{cutoffValue} \right)$   
12:    R[k] = argmin  $\left( \frac{\text{rightP}}{\sqrt{|\text{rightP}^2 - ||ts[fn + c]^2||} + \epsilon}} - \text{cutoffValue} \right)$   
13:   end for  
   return L,R  
14: end procedure
```

4.4.2. ALGORITHM: NOISY SAMPLED ADAPTIVE WINDOWING. Algorithm 4 combines Algorithm 2 and 3 into a complete algorithm for estimating the curvature of noisy time-series approximations of curves. To make this pseudocode easier to read, it is written to estimate curvature at a single location within the time-series. A version of this algorithm, implemented in MATLAB, is found in Appendix B.

Algorithm 4 Curvature Estimation on Noisy Data using Adaptive Windowing

```
1: procedure DOUBLE CURVATURE(TS[1...N], CUTOFFVALUE, FN) fn is an integer referring to the frame number
2:   dimension = size(ts,1)
3:   for int k from 1 to dimension-1 do
4:     tWindow = ts[fn-ceiling(k/2)]...ts[i+ceiling(k/2)]
5:     Mean-subtract tWindow
6:     [U,S,V] = Matlab::svd(tWindow)
7:     // Search over c. d is point to subspace distance.
8:     leftP(c) = d(ts[fn-c]-ts[i],U[1:k])
9:     rightP(c) = d(ts[fn+c]-ts[i],U[1:k])
10:    L[k] = argmin  $\left( \frac{\text{leftP}}{\sqrt{|\text{leftP}^2 - ||ts[fn - c]^2||} + \epsilon}} - \text{cutoffValue} \right)$ 
11:    R[k] = argmin  $\left( \frac{\text{rightP}}{\sqrt{|\text{rightP}^2 - ||ts[fn + c]^2||} + \epsilon}} - \text{cutoffValue} \right)$ 
12:    window(k) = [ts[fn-L[k]] | ... | ts[fn+R[k]]]
13:    Mean subtract window
14:    window(k) = window(k)*(window(k)T) / size(window(k)T, 1)
15:    E = Matlab::flipud(Matlab::eig(window(k)))
16:  end for
17:  for int j from 1 to dimension-1 do
18:    GC[j] =  $\sqrt{\frac{(2(j-1)+3)(2(j-1)+5)}{3}}$  *  $\sqrt{\frac{E[j+1]}{E[1] * E[j]}}$ 
19:  end for
20:  return GC
21: end procedure
```

4.4.3. ALGORITHM: NUMERICAL DERIVATIVE CURVATURE ESTIMATION. Algorithm 5 presents a basic algorithm to estimate curvature based on the Frenet-Serret equations (see Chapter 2, Equation 2). Since the algebraic curve is not known, we must approximate the derivatives. For the purposes of this dissertation, we consider the simple technique of subtracting neighboring frames to use as these approximations.

Algorithm 5 Numerical Derivative

```
1: procedure DOUBLE NUMERICALDERIVATIVE(TS[1...N], FN) fn is an integer referring  
   to the frame number  
2:   dimension = size(ts,1)  
3:   Initialize nd  
4:   for int j from 2 to dimension+1 do  
5:     for int i from 2 to N-1 do  
6:       nd[:,i,j] = nd[:,i,j-1] - nd[:,i+1,j-1]  
7:     end for  
8:   end for  
9:   Initialize evs  
10:  for int i from 1 to N do  
11:    [Q,R] = Matlab::qr(nd[:,i,:])  
12:    evs[:,i,:] = Q[:,1:dimension+1]  
13:  end for  
14:  Initialize GC  
15:  for int j from 1 to dimension+1 do  
16:    for int i from 1 to N-1 do  
17:      GC[j,i] = | (evs[:,i,j] - evs[:,i+1,j]) · evs[:,i,j+1] | / ||nd[:,i,1]||  
18:    end for  
19:  end for  
   return GC  
20: end procedure
```

4.5. SYNTHETIC EXAMPLE

To compare curvature estimation algorithms, we generated a 3D synthetic curve with curvature and torsion defined by the equations

$$(19) \quad \kappa(t) = 5t^2 + 5t, \quad \tau(t) = \frac{100}{2t + 1}$$

We then sampled 1000 points in the range from $t = 1$ to $t = 3$, and added 0.00015 uniform noise to each sample. We estimated the curvature at each point using Algorithms 1, 2, 1+3, 4, and 5.

The results are shown in Figure 4.2. The blue lines shows the signed error at every data point when the curvature is estimated using local derivatives. Even though the added noise

is very small (0.00015), the error in the estimated curvatures can be as large as 187. The average error magnitude is 25.2. None of this is surprising, given that estimating derivatives by subtracting noisy samples is known to be error prone. The red lines show the signed error at every data point when Algorithm 1 is used to estimate curvature. The average magnitude of the error is much smaller (9.6 vs. 25.2). Nonetheless, 9.6 remains a significant error, particularly considering how little noise was added. Finally, the black lines show the signed error when curvatures are estimated using Algorithm 4. The error is now much smaller, with an average magnitude of 0.8. In fact, the black lines almost look like we just drew a thick horizontal axis.

Algorithm 4 is defined by Equation 18 applied to adaptive data windows, as described in Section 4.4. Algorithm 4 advances the previous state of the art, namely Algorithm 1 (the algorithm derived in Chapter 3), in two ways. The first is Equation 18, which estimates the curvature of the principal curve. By way of contrast, Algorithm 1 computed the curvature for a data point assumed to be on the true curve. Any noise in the data point therefore contributes noise to curvature estimate. The second advance is the method for adaptively selecting the window size based on the underlying curvature, as opposed to Algorithm 1 and 2 which uses a fixed scale. Figure 4.3 decomposes the impact of these two contributions. The blue curve shows the result of applying Equation 18 to estimate curvature assuming a fixed scale window. As a result of Equation 18, the variance in the error becomes very small, meaning that there is very little difference in error between one point and its neighbor. This shows that the assumption that data samples lie on the curve adds a large but unbiased error to Algorithm 1 that is fixed by using Equation 18 instead of Equation 15. What remains is bias caused by the fixed window size. The error represented by the blue lines is small at the beginning of the sampled curve, where t is near 1 and the curve is nearly linear. As t

gets larger and the curvature increases, however, the estimates from Equation 18 begin to systematically underestimate the curvature. The bias grows to 34.2 when the value of t is approximately 2.5. Different fixed window sizes yield different versions of the blue curve, but the problem of bias always remains.

The red lines in Figure 4.3 show the result of estimating curvature using Equation 15 but adding an adaptive window as suggested in Section 4.4. The adaptive window removes the bias that is evident in the blue lines. Since it assumes data points are on the curve, we still see the random errors from Algorithm 1. As before, the black line represents Algorithm 4, which is the combination of adaptive windows and Equation 18.

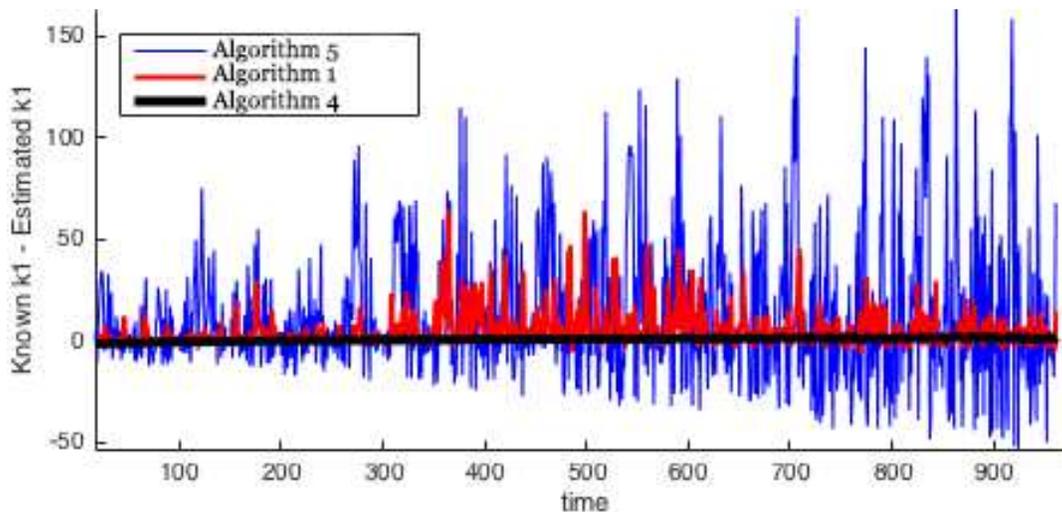


FIGURE 4.2. Curvature estimation errors. The blue lines show the signed errors in curvature when curvatures are estimated from local derivatives, or Algorithm 5. The red lines show the errors when curvatures are estimated using Algorithm 1. The black lines show the residual errors using Algorithm 4.

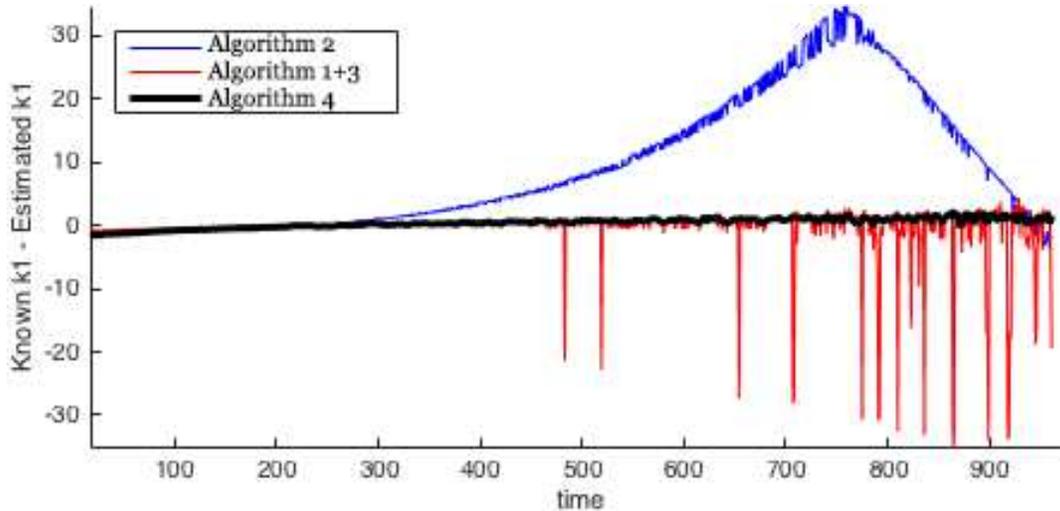


FIGURE 4.3. Decomposing Algorithm 4. The blue lines show the result of computing curvatures using Algorithm 2. The red lines show the result of adapting the window size to the underlying curvature, or Algorithm 1+3. The combination of these two techniques, shown in black, is Algorithm 4.

4.6. CONCLUSION

In this chapter, we established two algorithms designed to more accurately compute generalized curvature values on noisy, discrete, time-series data. Expanding on the theory presented in the previous chapter, we began by establishing a relationship between the local singular value decomposition of regular, mean-subtracted curves in \mathbb{R}^n and the Frenet-Serret apparatus. When using the SVD, a commonly employed technique is to mean-subtract the data in order to remove bias from the coordinate system. This approach differs from the method established in Chapter 3 and as a result, a new equation for computing generalized curvature as a function of the singular values was established. The other major contribution from this chapter is an algorithmic approach to selecting the optimal window size to use for the local singular value decomposition. In the example provided, we show evidence that a small window size is needed when the generalized curvature value is large. Conversely, a large window size better approximates the generalized curvature value when the generalized

curvature value is small. The combination of these two proposed algorithms, referred to as Algorithm 4, provides much more accurate and precise estimates of generalize curvature than the method in Chapter 3 or other numerical techniques.

CHAPTER 5

HUMAN MOTION SEGMENTATION VIA GENERALIZED CURVATURE

5.1. 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 example is the Microsoft Kinect [23]. The Kinect II outputs (x, y, z) coordinates for 25 body parts at approximately 30 frames per second. The Asus Xtion Pro Live [24] also produces real-time body poses, while the LeapMotion [25] and Intel RealSense [26] produce detailed hand poses. Figure 5.1 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 actions, while hand poses are analyzed for gestures. In both cases, the goal is to determine *when* motions occur and

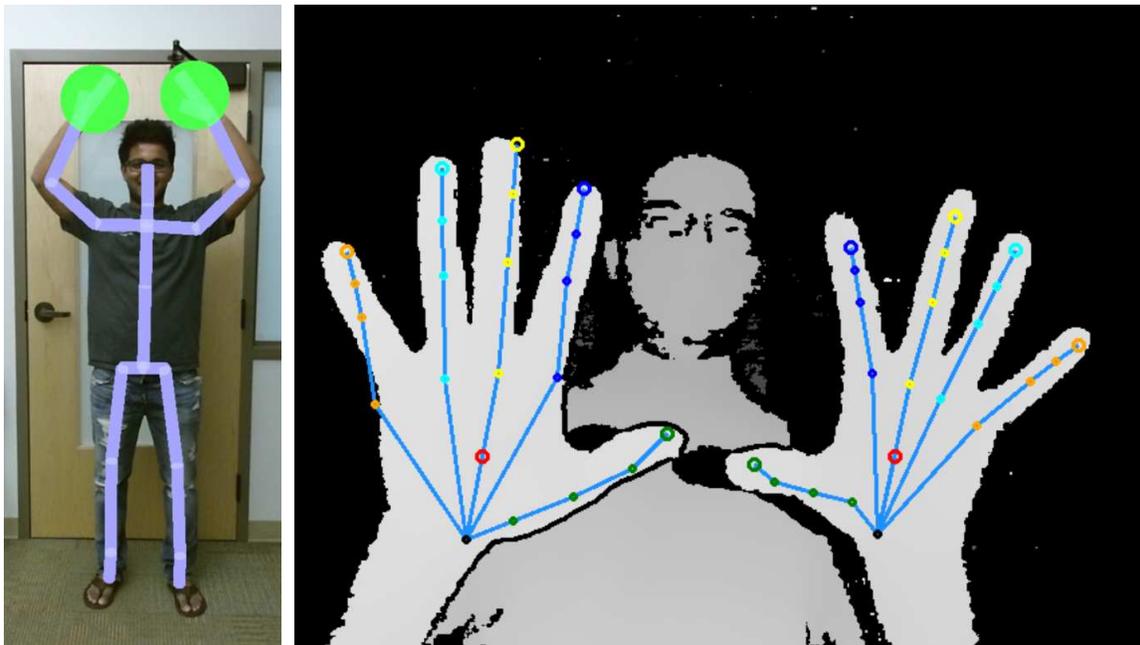


FIGURE 5.1. Example poses. The left side shows 25 body pose points extracted by a Microsoft Kinect II. The right side shows 44 hand points (22 per hand) extracted by the Intel RealSense.

what motions occur. This requires segmenting pose streams into motions and classifying the motions. When the set of possible actions is known a-priori, segmentation and classification can be solved jointly, as in [27–30]. However, there are applications where the actions are not known in advance. Examples include labeling tools, where the goal is to segment a stream prior to labeling, and some healthcare applications, where the goal is to measure the frequency, duration and magnitude of motions rather than to identify actions. In these cases, pose streams must be segmented into sets of unknown motions.

This chapter presents the first practical algorithm for segmenting unconstrained pose streams into arbitrary motions. It is based on two fundamental observations. The first 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 bodies trace out continuous curves in pose space over time, and 3D sensors sample points along these curves. Together, these observations suggest that pose streams can be segmented by separating the low-curvature transport phases of motions from the high-curvature transitions (initializations and conclusions) between motions. Moreover, this approach should work even when subjects do not pause or slow down between actions.

To go into more detail, the body parts tracked by a sensor like the Microsoft Kinect define a *pose space*. For example, the Kinect II tracks 25 points in 3 dimensions, so each pose is a point in a 75 dimensional space. Over time, a person’s body traces out a smooth, continuous curve in pose space, and the body positions detected by 3D sensors are noisy discrete samples of this curve.

By using the curvature estimation methods developed in Chapters 3 and 4, specifically, Algorithm 4, we will develop within Algorithm 6 a method to segment these pose streams.

When the curvature can be approximated to a high degree of accuracy, we will be able to identify key structures within the generalized curvature profiles for each video. This dimensionality reduction provides new techniques for extracting continuous, non-smooth motions from skeletal data.

To test the idea of using curvatures to segment Kinect 2 pose streams, we apply Algorithm 4 to a set of continuous action data streams. We show that Algorithm 4 estimates curvatures which when combined with Algorithm 6 partitions frames into motion frames and transition frames with 83.8% accuracy and recognizes 90% of all transitions. For comparison, curvatures based on Algorithm 1 and Algorithm 5 yields accuracies of 69.8% and 63.6% respectively, and recognize 35.1% and 3.1% of transitions.

5.2. RELATED WORK

This section touches on kinesiology, pose stream segmentation, computer vision, and computational geometry. We briefly discuss human motion below, with the goal of clarifying terminology. We then discuss different approaches to temporal segmentation, the role of differential curves in computer vision, and curvature estimation techniques from computational geometry.

5.2.1. HUMAN MOTION. As stated above, human motions can be divided into three phases. The initial phase recruits muscles to overcome the inertia of the previous state, whether the previous state was at rest or the remnants of a previous motion. Once the motion is initiated, the majority of the movement is relatively smooth. Finally, there is a conclusion to the motion where the body either stops or transitions to the next motion. Although these three basic phases apply to all human motions, the phases go by different names. Biomechanical engineers may call them *acceleration*, *motion* and *deceleration*, or

sometimes *take-off*, *motion* and *rest*. Kinesiology textbooks use *initialization*, *preparation* or even *anticipation* for the first phase, and *action*, *execution* or *transport* for the second. *Conclusion*, *completion* or *termination* are acceptable for the final phase. Where possible, we adopt the most common kinesiology terms, most notably *initialization* for the first phase and *conclusion* for the last. To avoid confusions, however, we use *transport* instead of the more common *action* or *execution* to describe the middle part of a motion.

5.2.2. SEGMENTING POSE STREAMS. Many applications require segmenting pose streams and labeling the resulting segments. When the set of actions is finite and known in advance, segmentation and labeling can be solved jointly, as in [27–30]. When the actions or 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 [31, 32]. The other is to segment the stream into fragments that cluster, on the theory that motions tend to repeat [33–38]. 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 [39]. Zhou et al 2013 extend this work to hierarchical decompositions at multiple scales [40]. Kruger et al. propose unsupervised segmentation based on self-similar structures using neighbourhood graphs [41, 42]. Koppula et al. use the sum of Euclidean distances between skeleton joints as edge weights for graph-based segmentation [43].

Temporal segmentation techniques that rely on kinetic changes are assuming that subjects pause or at least slow down between motions. This may not always be true, particularly for motions that are parts of familiar actions. Techniques that rely on clustering work well for repeated, rhythmic motions such as walking, but may not work as well for less regular motions. The approach proposed here is based on curvature, and separates motions even in the absence of pauses or when motions do not repeat themselves.

5.2.3. DIFFERENTIAL CURVES IN COMPUTER VISION. Differentiable curves have a long history in computer vision. Generally, 3D differentiable curves 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 [44], and Faugeras further developed this analysis [4]. Zucker gives the most thorough explication of the role of differential geometry in computer vision [45], including the differential geometric description of curves in more than 3 dimensions. Generalized cylinders were one popular representation defined in terms of smooth differentiable curves, for example Pegna [46], Bronsvort & Klok [47], and Zerroug & Navatia [48]. Wagner & Ravani described rational generalized cylinder models as Frenet curves [49]. Differential curves have also been used to describe the motion of cameras through stationary environments [50], the motion of tools as seen from stationary cameras [51], and the motion of moving cameras in complex domains [52]. More recently, Kim et al. analyzed space-time curves in terms of curvatures and torsions [53]. Differentiable curves have also been used to compare trajectories. Chern [54] and Qu [55] solved kinematics using Frenet frames. Wang et al. [56] and Vochten et al. [57] propose invariant trajectory descriptors based on Frenet-Serret formulae.

5.3. EXPERIMENT: PRELIMINARY HUMAN MOTION

Poses over time trace out a curve in pose space, and Chapter 4 describes new techniques for robustly estimating its multi-dimensional curvature at any discrete sample time t_i . The derivation of this technique was strictly mathematical, however, so the goal of this section is to evaluate it on real data, in particular data from the Microsoft Kinect II sensor.

There is a challenge to evaluating this technique experimentally, namely the lack of ground truth data. The purpose of this algorithm is to robustly estimate curvatures by

overcoming the noise in the Kinect and related sensors. Therefore we cannot use data from these sensors to evaluate the results. Instead, we rely on indirect measures such as the smoothness of estimated curvatures, as described below.

5.3.1. EVALUATION METHODOLOGY. The goal of the algorithm presented above is to robustly extract $n - 1$ dimensional curvatures from noisy data samples. Unfortunately, since all pose sensors have at least some noise, there is no "ground truth" data to compare results to. Instead, we have to rely on indirect measures of quality based on the mechanics of human motion.

We define a human motion (as opposed to an activity) as a movement during which no joint changes direction. Consider, for example, the motion "duck down". It involves almost every major joint in the body. To duck, people bend at the hips, knees and ankles, while curving their spine and lowering their head. They also bend their elbows while raising their arms to keep their balance. Nonetheless, ducking down is a single motion because no joint changes the angle of its ego-centric trajectory, although different joints accelerate or decelerate at different times during the motion. Rising up again is then another motion.

Motions have the property that they create smooth trajectories in pose space with relatively low curvatures, since muscles don't start pulling in different directions mid-motion. Transitions between motions, on the other hand, produce sharp curves. When a person ducks down and then comes back up, for example, their curvatures in pose space spike at the transition, as almost every joint reverses direction. Transitions between less-related motions also produce curvature spikes. If a person rises up and then pushes to the right, for example, their shoulders and arms change directions while their knees and hips come to an approximate stop. As a result, we expect methods of estimating curvature to produce small and smooth curvature values during motions with sudden spikes in curvature between them.

We therefore collected ten Kinect II pose trajectories of people performing sequences of three actions from the Microsoft gestural challenge [58], and hand-labeled the data at the level of motions rather than actions. For example, the action "duck" is split into two motions, one for ducking down and one for rising up. Figure 5.6 shows a 3D projection of one of the 75 dimensional pose trajectories, with motions shown in blue and the transitions between motions shown in red. The idea is that the curvature estimates for the blue segments should be small, although possibly noisy, while the red segments should contain curvature spikes. In the experiments below, we therefore compare the ratio of κ_1 curvatures during transitions to the κ_1 curvatures during motions, with the idea that good curvature estimations should yield high ratio values.

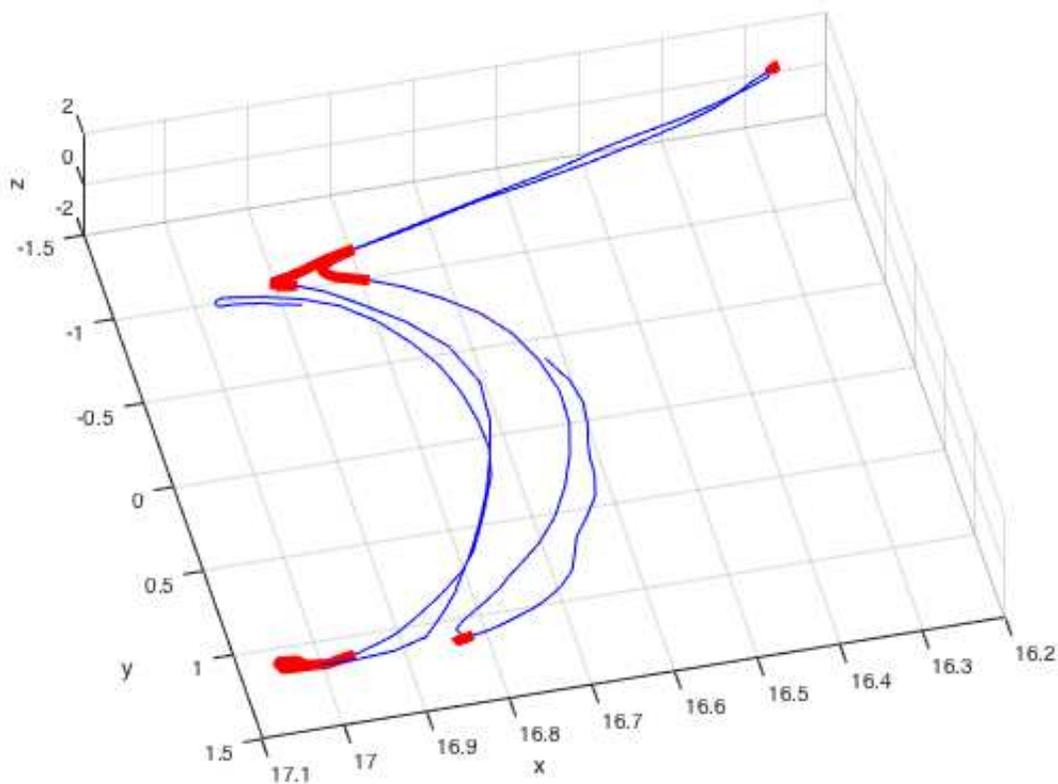


FIGURE 5.2. 3D projection of a 75 dimensional pose stream showing three actions (Push Right, Kick and Push Right). Sections of the trajectory shown in blue represent atomic motions, while sections shown in red represent transitions between motions. As seen here, motion trajectories tend to have low curvature, whereas the trajectories of transitions between motions are highly curved.

5.3.2. EXPERIMENT DESIGN. Using the evaluation methodology above, we compare five curvature estimation techniques. The first is the traditional method of estimating numerical derivatives from data samples and then using those derivatives to compute curvatures as described in [22]. This represents the state of the art prior to the technique presented in Chapter 3, and we refer to it as Algorithm 5. The second method is the technique in Chapter 3, as described in Algorithm 1. The third method extends the method in Chapter 3 by centering the data and altering the curvature formula as proposed in Algorithm 2. This

algorithm estimates curvatures without knowing the exact position of the curve. The fourth method adds adaptive data window sizes as described by Algorithm 3. Finally the fifth technique applies both extensions to the results in Chapter 3, Algorithm 4.

Figure 5.7 shows the κ_1 curvature estimates of the five techniques over the course of a single pose stream (video). The red vertical lines mark the beginning of a motion, while the blue lines mark the end of a motion. Thus motion sequences begin at a red line and end at a blue one, while transition sequences begin at blue line and end at a red one. As predicted, the κ_1 estimates of our method tend to be relatively lower than the κ_1 estimates of the other techniques during smooth motions, and higher than the estimates of other techniques during transitions between motions. Note also the changes in scale on the vertical axes in Figure 5.7; the ratios between the estimated κ_1 values during transitions compared to motions becomes much stronger for the more refined estimation methods.

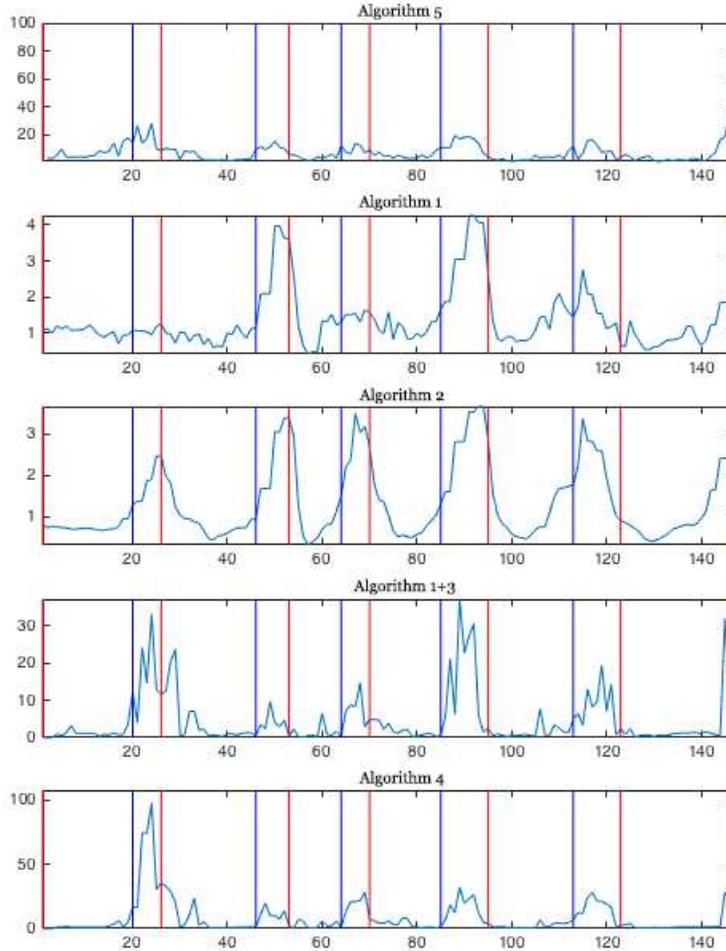


FIGURE 5.3. Estimated κ_1 curvatures over time as person performs a sequence of three actions with six atomic motions. The figure is organized vertically into five plots, one for each curvature estimation method. The top plot shows the traditional numerical derivative method, Algorithm 5. The second plot shows the method described by Algorithm 1. The third plot shows the off-curve extension in Chapter 3, Algorithm 2. The fourth plot shows the Chapter 3 (Algorithm 3) method with adaptive windows. The fifth (bottom) plot shows both the off-curve and adaptive window extensions, Algorithm 4. The horizontal axes are time (at 30fps), while the vertical axes are estimated κ_1 . Vertical red bars indicate the start of a motion, while vertical blue bars indicate the end of a motion. Therefore, sequences from a red bar to a blue one should be low-curvature motions, whereas sequences from a blue bar to a red one should be high-curvature transitions.

To quantify this observation, we measured the mean and median κ_1 values over the course of a motion for each technique, and we did the same over the course of the transitions that followed. We then computed the ratios of these values (transition over motion), and

computed the means and medians of these ratios for the 59 motion/transition pairs in the ten pose streams (videos). The results are shown in Table 5.1. Our extension of mean centering the data to avoid needing to know the position of the curve (and subsequently replacing Equation 15 with Equation 18) improves performance over the method in Algorithm 1. This is true whether you look at the mean of mean ratios, or the median of the median ratios. Using an adaptive window size, however, is the more significant extension. Together, these two refinements produce the best curve estimates.

TABLE 5.1. Comparison of in-motion to in-transition curvatures estimated by five techniques. The technique names (algorithms) are described in the first paragraph of Section 5.3.2. The measures are ratios of estimated curvatures during transitions between motions over curvatures within motions. Since curvatures should be high during transitions and low during motions, high values are better.

Method	Mean of means	Median of medians	Mean of max's	Mean of st. dev.'s
Algorithm 5	2.76	2.88	2.84	3.19
Algorithm 1	2.04	1.53	1.90	2.72
Algorithm 2	2.70	2.41	2.38	3.46
Algorithm 3	6.11	4.65	5.43	6.81
Algorithm 4	7.43	5.62	6.55	9.68

Table 5.1 also shows the mean of the ratios of standard deviations. Whereas the mean of ratio means and median of ratio medians show that the estimated curvatures are higher during transitions than motions, the mean of the ratio standard deviations shows that the estimated curvatures are “spikier” (less uniform) during transitions and smoother during motions. Again, the results are as predicted, with both extensions improving on the method in Chapter 3 Algorithm 1, and their combination performing the best. Finally, Table 5.1 shows the mean of the ratio of maximum values between the transition and motion segments. This number is particularly interesting if the goal is to segment pose streams based on peaks in the estimated curvatures.

Table 5.1 shows a very strong trend, but there is some question as to whether the mean of ratio means (or median or ratio medians) is the appropriate statistic. The problem is that while all motions have low curvature, some have slightly higher true curvatures than others. Similarly, while all transitions have high curvatures, some may be higher than others. As a result, although high ratios are better than low ratios, every motion/transition pair has a different "true" ratio. As an alternative statistic, we look at every motion/transition pair and ask which technique produced the highest ratio. As shown in Table 5.2, once again the versions with both extensions outperforms the alternatives. Interestingly, the numerical derivative is very sensitive to noise so it does poorly on average (see Table 5.1) but it occasionally succeeds (see Table 5.2).

TABLE 5.2. Counts of motion/transition sequence pairs for which each technique had the highest (best) ratio. The first set of comparisons are among all five techniques: numerical derivatives (Algorithm 5), method in Chapter 3 (Algorithm 1), extended off-curve (Algorithm 2), extended with adaptive windows (Algorithm 3), and with both extensions (Algorithm 4). The second set of comparison compares numerical derivatives (the previous state of the art), to the method proposed in Chapter 3 (Algorithm 1) for on-the-curve and off-the-curve proposed here with and without adaptive window size. The values being compared are ratios of means, medians or maximums.

Method	Mean	Median	Max
Algorithm 5	6	7	11
Algorithm 1	2	2	4
Algorithm 2	6	10	6
Algorithm 3	12	16	12
Algorithm 4	32	23	25
Algorithm 5	8	8	13
Algorithm 1	7	9	7
Algorithm 4	43	41	36

Finally, we were interested in whether the grand means and grand medians hid significant differences among the pose streams. Figure 5.4 shows the median of ratio medians over

each of the ten streams. There are significant variations based on the specific motions and transitions, but the relative ordering of quality among the techniques is fairly stable.

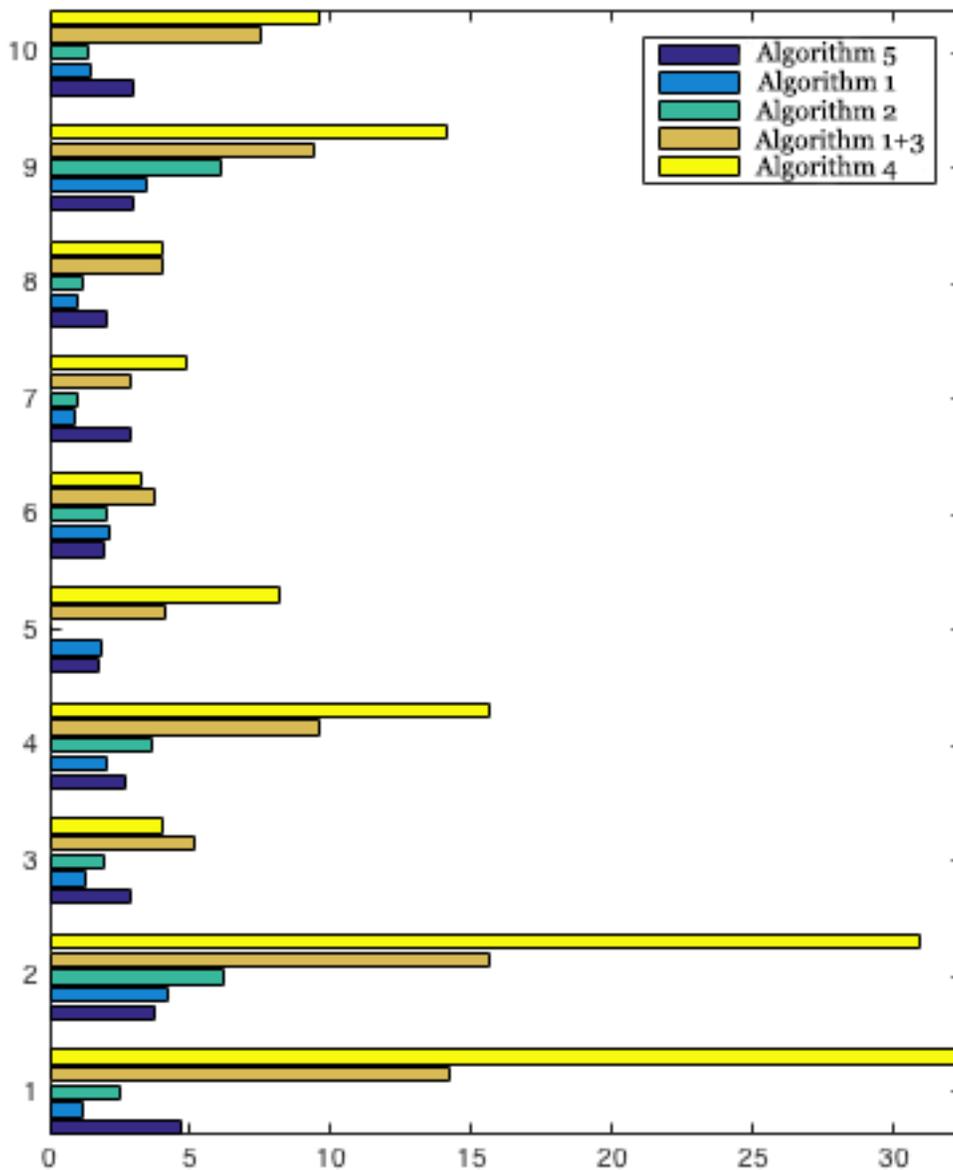


FIGURE 5.4. κ_1 values computed by all five curvature estimation methods over all ten pose streams.

5.3.3. PRELIMINARY HIGHER-DIMENSIONAL CURVATURES. So far, we have evaluated the quality of curvature estimation in terms of κ_1 , the first direction of curvature. Curves

in high-dimensional spaces have many dimensions of curvature, however, and one of the advantages of the framework in Chapter 3 is that it can be used to calculate curvatures in n dimensions. Having extended in Chapter 3 to handle noisy data, we are interested in how many dimensions of curvature might be extracted from a real-world signal. Figure 5.5 shows the estimates of κ_1 through κ_{12} for a single pose stream. Not surprisingly, the curvature estimates are highly correlated across dimensions. The argument that motions have low curvature while transitions have high curvature holds across dimensions. Also not surprisingly, the signal to noise ratio decreases as the index of the curvature dimension increases. In particular, many of the transition segments lose their higher curvature estimates. This could be because the curvature estimates beyond six dimensions are no longer reliable, or alternatively it could be because the transitions between atomic motions are rarely more than six dimensional.

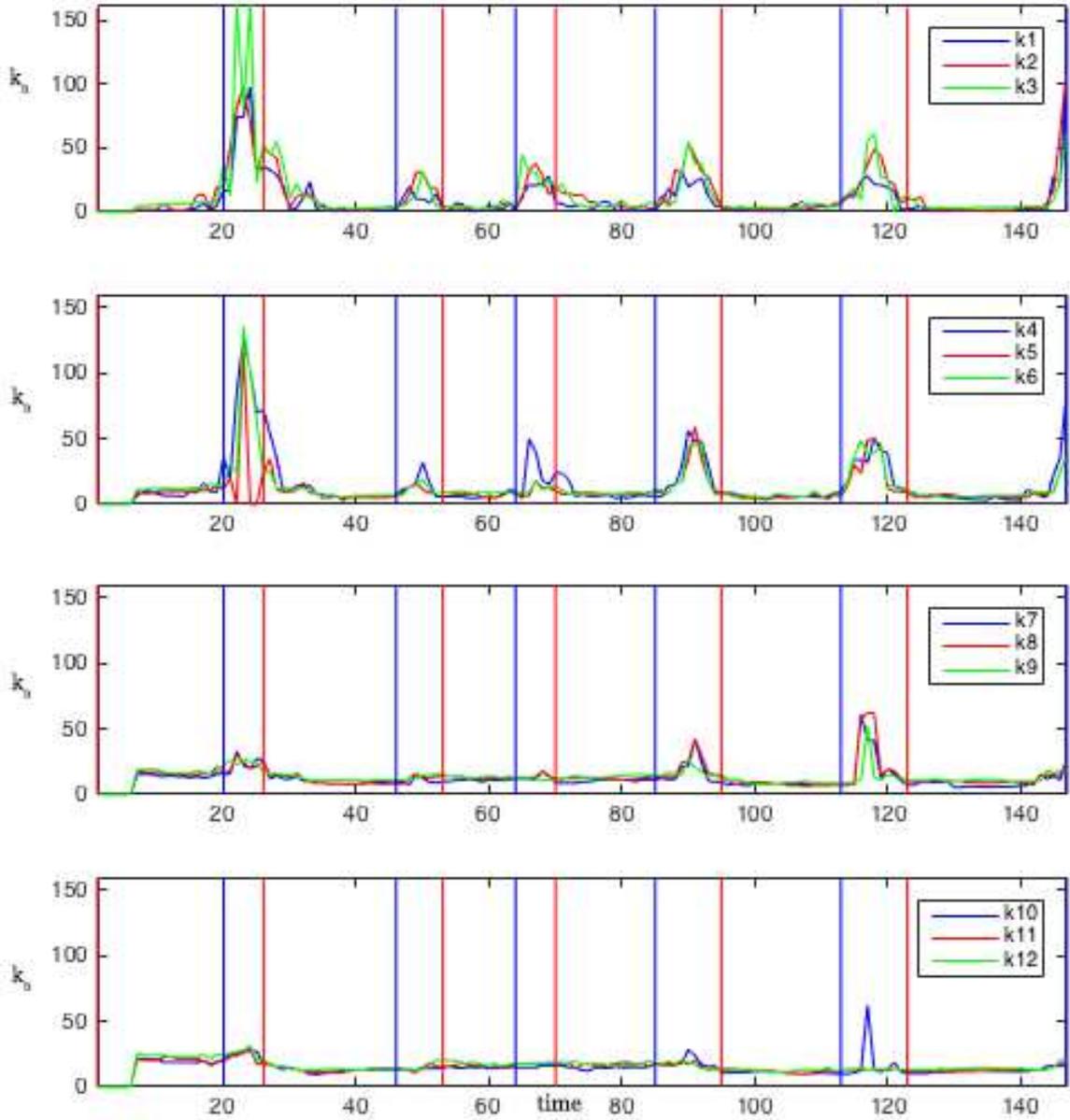


FIGURE 5.5. 12 dimensions of curvature (κ_1 through κ_{12}) as estimated using the proposed extensions (Algorithm 4) to the method in Chapter 3 (Algorithm 1). Roughly the first six dimensions of curvature (shown in the top two bars) seem reliable (see Table 5.3 for statistics). Dimensions 7 through 12 still seem to contain some signal, but many transitions no longer show elevated curvatures. (Remember, we expect low curvatures between red bars and blue bars, and higher “spiky” curvatures between blue bars and red bars.)

Table 5.3 shows the same quality measures as in the previous sections, except this time we compare the quality of κ_1 to κ_2 to ... to κ_{12} . The quality of the curvature estimates is highest

with κ_1 but slowly degrades. When the median of median ratios drops below 2, we discard the data as being no longer generally useful (although this depends on the application). We suggest that our method extracts approximately 6 useful dimensions of curvature from a Kinect II pose stream. Whether this limit is because of the curvature estimation technique or the degrees of freedom in transitions between human motions remains to be determined.

TABLE 5.3. Comparisons of quality measures for κ_1 through κ_{12} , as computed using both proposed extensions to the method in Chapter 3.

N	Mean of means	Median of medians	Mean of st. dev.'s
κ_1	7.41	6.34	9.80
κ_2	6.64	5.70	6.07
κ_3	5.12	4.10	7.92
κ_4	4.55	3.52	7.79
κ_5	3.62	1.95	8.79
κ_6	3.90	2.03	9.90
κ_7	3.33	1.48	13.34
κ_8	3.21	1.30	12.47
κ_9	2.93	1.20	20.60
κ_{10}	3.07	1.18	23.62
κ_{11}	2.77	1.07	28.91
κ_{12}	2.74	1.06	26.55

5.4. EXPERIMENT: FULL PALKA DATASET

The 10 videos used for the prior set of experiments illustrate the usefulness of the Algorithm 4, the algorithm proposed in Chapter 4 to the problem of human action segmentation. We showed the validity of our assumption: human motions, in their native high-dimensional pose space, have low generalized curvature values during smooth continuous motions and high generalized curvature values during motion transitions. Furthermore, from the experiment in Section 5.3.3 we conclude that κ_1 values will give us a feature with the highest level of contrast between motions and transitions. This section expands on those ideas. We consider an entire dataset (PALKA) and use the curvature algorithms proposed in Chapters

3 and 4 (Algorithms 1-4) along with rudimentary curvature estimates to segment human action skeleton videos into individual motions and transitions.

5.4.1. ALGORITHM: TEMPORAL SEGMENTATION. Human body positions trace out smooth curves in pose space over time, and sensors like the Kinect 2 sample this curve. Our goal is to segment pose curves into individual motions, without knowing the set of possible motions in advance or requiring that actions are repeated. Our approach is to segment pose streams into sequences of smooth motions separated by high-curvature transitions using the simple algorithm shown in Algorithm 6.

Algorithm 6 A simple segmentation algorithm based on curvature.

```

1: procedure SEGMENT(VIDEO X, MINPEAKHEIGHT, MINSIZEDETECTIONS)
2:   //  $D[i] == 0$  indicates frame  $i$  is motion;  $D[i] == 1$  indicates transition
3:    $k[] = \text{Curvature}(X)$  // any discrete curvature estimation method
4:    $\text{peaks} = \text{Matlab}::\text{findpeaks}(k, \text{minPeakHeight})$ 
5:   Initialize D to array of zeros of length (X)
6:   For every peak
7:     assign  $D[\text{peak} - \text{peak.width}]$  through  $D[\text{peak} + \text{peak.width}]$  the value 1.
8:   Set isolated motion frames to be transitions // (i.e.  $D[i] = 1$ )
9:   Remove detections that are not part of a run of length minSizeDetections.
10:  Output D // D is a vector indicating transition frames.
11: end procedure

```

We tested the algorithm above on the PALKA data set, which contains pose streams from 234 videos recorded by a Kinect 2 sensor. Every video shows a person performing three actions drawn from the MSCR-12 [58] 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. Additional information about this dataset has been included in Appendix A.

Figure 5.6 illustrates part of one PALKA video. This example contains two MSRC-12 actions, but four basic motions. The first MSRC-12 action, wave arms, is two motions: the subject raises their arms, and then lowers them. Similarly, the second MSRC-12 action, kick,

is two motions: kicking out, and then bringing the leg back. The right side of Figure 5.6 shows the 75-dimensional pose curve projected onto its first two eigenvectors. Low curvature sections of the curve (as estimated by Algorithm 4) are colored blue, while high curvature sections are colored red. The left side of the figure illustrates selected poses along the curve, including poses during transitions.

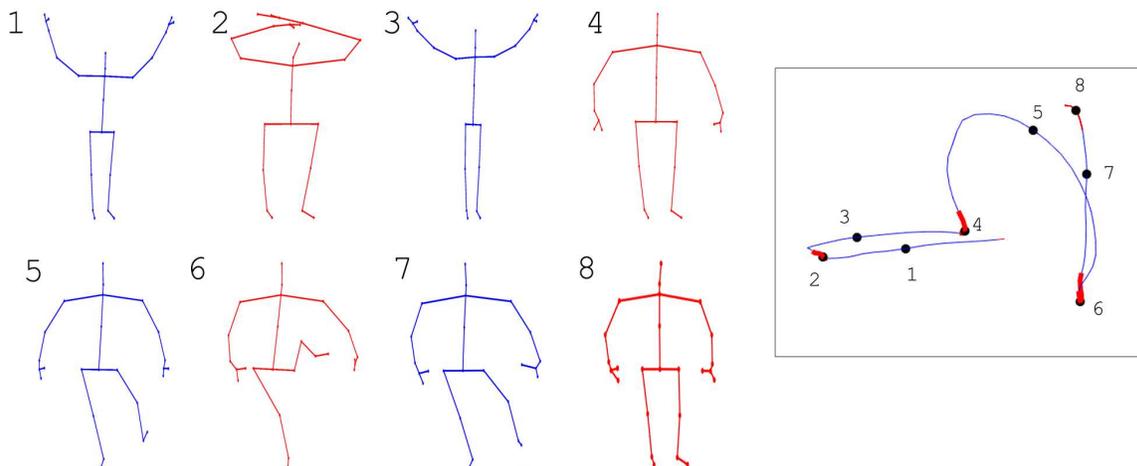


FIGURE 5.6. (Right) 3D Smoothed Projection of 75D curve of two sequential actions; Lift Outstretched Arms followed by Kick. (Left) 1) Raising Arms 2) Transition between Raising Arms and Lowering Arms 3) Lowering Arms 4) Transition between Lift Outstretched Arms and Kick 5) Raising Left Leg 6) Transition between Raising Left Left and Lowering Left Leg 7) Lowering Left Leg 8) Transition between Kick and next action.

Figure 5.7 shows the estimated curvatures for a complete PALKA video, computed using local derivatives, Algorithm 5 (top), Algorithm 1 (middle), and Algorithm 4 (bottom). Because the extreme curvature at the end of the video distorts the scale of the vertical axis, part of the video is broken out in the box to the left. There are five transitions in this data (not counting the end of the video), and curvatures computed from local derivatives clearly identify two of them (the third and fifth). There are also high curvatures around the first transition, but they are not well localized. Algorithm 1 does better, finding an increase in

curvature near all transitions, but none of the transitions are well localized. Algorithm 4, on the other hand, clearly predicts all five transitions with no false positives.

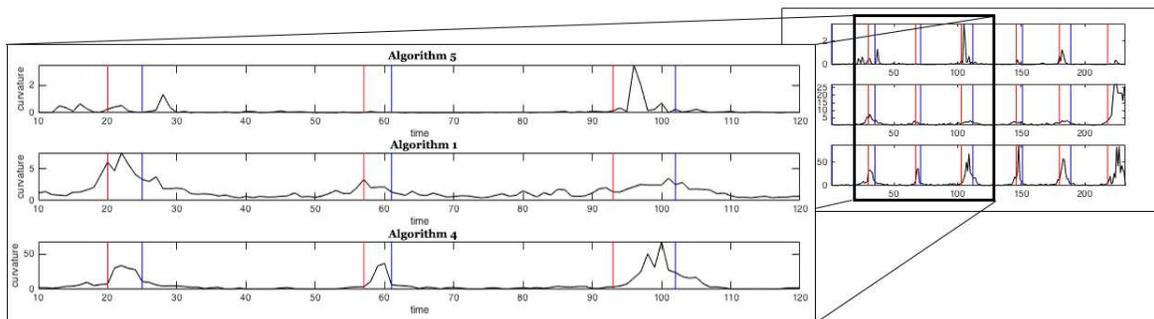


FIGURE 5.7. Estimated curvatures over time as person performs a sequence of three actions with six atomic motions. The figure is organized vertically into three plots. The top plot shows the traditional numerical derivative method. The second shows the Algorithm 1. The third plot shows the proposed curvature estimation method. The horizontal axes are time (at 30fps), while the vertical axes are estimated curvature magnitudes. Vertical blue bars indicate the start of a motion according to the hand-labeled data, while vertical red bars indicate the end of a motion. Therefore, sequences from a blue bar to a red one are motions, whereas sequences from a red bar to a blue one are transitions between motions.

Table 5.4 summarizes performance across the entire data set. We tested curvature-based temporal segmentation (Algorithm 1) with all three curvature estimation techniques. The segmentation algorithm takes two parameters (`minPeakHeight` and `minSizeDetection`), and Algorithm 4 requires one more (the adaptive cutoff threshold). We therefore divide the 234 videos into a training set (156 videos) and a test set (78 videos), making sure that no subject appears in both the training and test sets. For every curvature estimation technique, we exhaustively searched for the best parameters over the training set, and then used those parameters when the technique was applied to the test set. Table 5.4 also includes a baseline algorithm that labels all frames as motion frames. While the baseline is useless as a segmentation algorithm, it provides a basis for analyzing frame-based percentages.

TABLE 5.4. Segmentation results on PALKA dataset.

	Motion Frame Accuracy	Transition Frame Accuracy	Total Accuracy	% of Transitions Detected
Algorithm 4	91.4%	70.5%	83.8%	90.0%
Algorithm 1	91.8%	31.8%	69.8%	35.1%
Algorithm 5	99.9%	1.2%	63.6%	3.1%
Baseline	100%	0%	63.2%	0%

As shown in Table 5.4, the baseline algorithm labels 63.2% of the frames correctly, since 63.2% of all frames in the test set are motion frames. Curvature-based segmentation using numerical derivatives, Algorithm 5, to estimate curvature is only slightly better, at 63.6% correct. When Algorithm 1 is used to estimate curvatures, 69.8% of frames are correctly labeled. When Algorithm 4 is used to estimate curvature, however, 83.8% of all frames are labeled correctly.

Looking at the accuracies broken out by motion frames and transition frames, we see that all curvature estimation techniques do a good job identifying motion frames. In essence, it is rare for any technique to overestimate curvature. Numerical derivatives (Algorithm 5), however, rarely identify segments of high curvature. Algorithm 1 does better, but only Algorithm 4 finds the majority of transition frames. The last column in Table 5.4 shows the percent of transitions, as indicated by the hand-labeled ground truth data, that were detected automatically. (For this table, a ground-truth transition is considered detected if it overlaps an automatically-detected transition.) Algorithm 4 detects 90% of transitions bottom-up, whereas Algorithm 1 and Algorithm 5 detect 35.1% and 3.1% respectively. False positive transitions are not reported, because they did not occur in practice.

5.4.2. ADAPTIVE PARAMETER ON SEGMENTATION RESULTS. The algorithm for optimizing the data windows (Algorithm 3) to use in Algorithm 4 (see Section 4.4) is a function of the parameter, h . When presenting that algorithm, we noted that through empirical tests,

using a h value in the range $0.05 \leq h \leq 0.5$ produces robust bounding intervals. That qualitative result was from observations made on experiments using synthetic data. To strengthen the validity of those claims, now that we have a quantitative method for testing the use of the curvature algorithm (for the task of motion segmentation as described in the prior section), we vary h and observe the change of segmentation accuracy of Algorithm 6 using Algorithm 4.

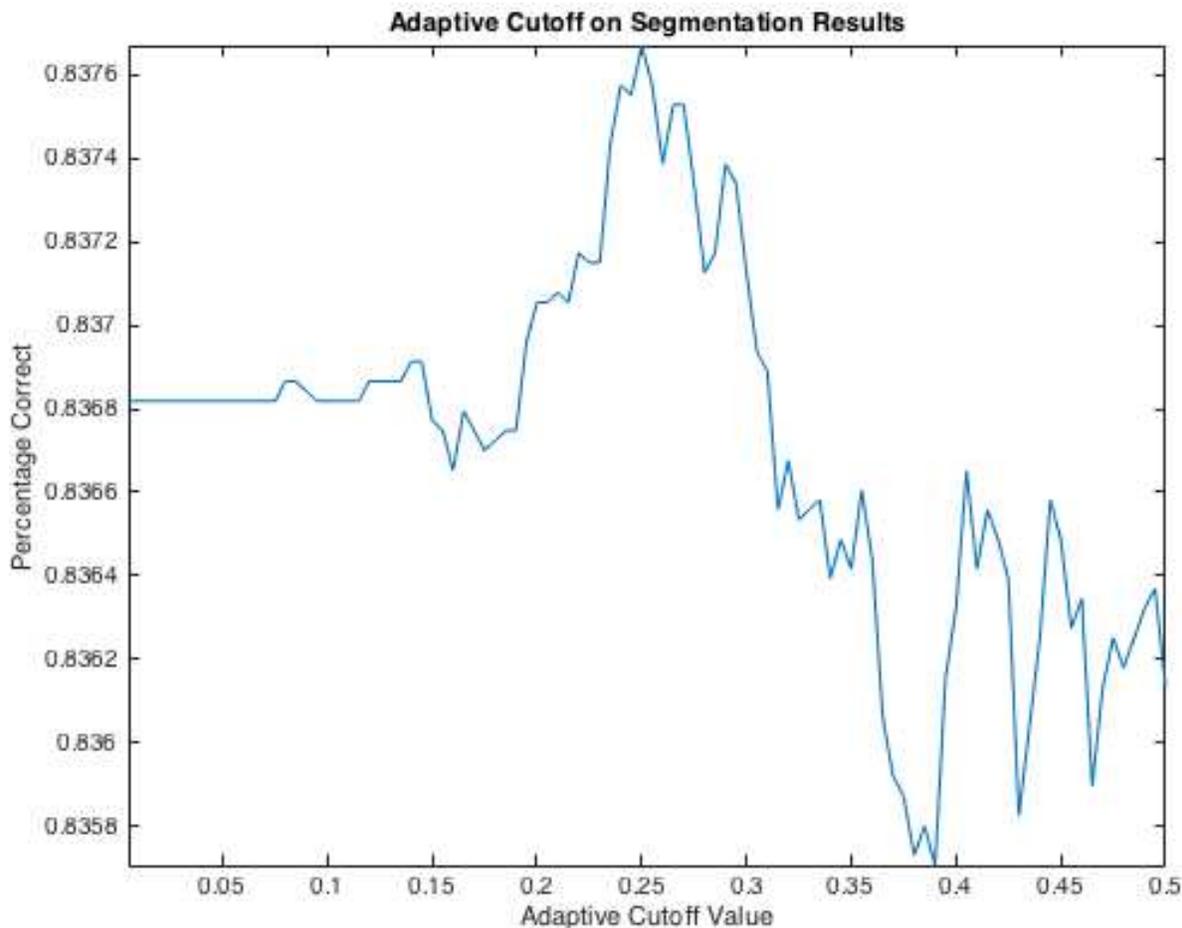


FIGURE 5.8. Accuracy of Algorithm 6 using Algorithm 4 algorithm used to measure motion segmentation on the PALKA dataset as the parameter h (or adaptive cutoff value), used in optimizing the data windows varies.

From Figure 5.8 we note that as h , the Adaptive Cutoff Value varies, the change in accuracy of the segmentation algorithm is less than 0.2%. The best accuracy on the PALKA

dataset is 83.8% which is achieved when $h = 0.25$. This segmentation accuracy is the value reported in Section 5.4.1, Table 5.4.

5.5. CONCLUSION

This chapter shows the extensions of the theoretical results in Chapters 3 and 4 for the use in the applied problem of motion segmentation by estimating generalized curvatures from discretely sampled noisy data. At any discretely sampled time t_i , the approach proposed here estimates the $n - 1$ generalized curvatures of a pose trajectory in n dimensions. Our indirect, preliminary experiments, measures indicate that Algorithm 4 produces more reliable curvature estimates than Algorithm 1, and much more reliable curvature estimates than traditional techniques based on estimating numerical derivatives (Algorithm 5). Experiments also suggest that meaningful curvatures can be estimated for up to 6 dimensions from Kinect II pose streams.

Accurate curvature estimates in turn allow us to segment pose streams without knowing the set of action or motions in advance. The beginnings and endings of human motions are marked by high curvatures in pose space, while the body of the motion – the so-called transport phase – is characterized by low curvature. We therefore present a simple, curvature-based temporal segmentation algorithm that divides pose streams into motions with intervening transitions, without assuming that subject pause between motions or that all motions rhythmically repeat. We use these techniques to robustly segment pose streams into atomic motions for use in subsequent action analysis.

CHAPTER 6

HUMAN ACTION CLASSIFICATION

6.1. INTRODUCTION

In this final content chapter, we conclude with the exploration of our initial motivation task of action classification.¹ This widely studied task has been performed on a wide variety of data types such as RGB, RGB-D , skeleton data, point cloud, etc. [59] [60] [61] [62] [63]. The overall goal for this line of research is to develop a robust method to identify all human activities in real-time. Accomplishing this task has obvious security applications. However, these systems are often explored for use within medical establishments.

Building on the best set of segments obtained in Chapter 5 (which were segmented using Algorithms 4 and 6) we will construct a system to label these segments. To do this, we use a common computer vision techniques of separating the data into a training and testing datasets. Within the training data, we use the ground truth labels to establish a baseline of what each action class “looks” like. The testing data is then compared to the labeled training data. Based upon the similarities, the testing data is labeled and the accuracy of the labels are compared with the ground truth data.

The system we showcase in this dissertation is built on four major components. The first component is segmenting the original videos (as described in previous chapters). The remaining components are well-established techniques combined with new work in Hidden Markov Models. First, we compare the similarities of segments using a dynamic time warping algorithm. Based on these distances, we use an agglomerative clustering algorithm to group

¹Part of this work is in collaboration with Pradyumna Kumar.

similar motions together. Finally these groups are pieced together into complete actions using Hidden Markov Model techniques developed by Pradyumna Kumar [64].

Applying all these techniques on the PALKA dataset, a dataset collected for this project, yields an 82% classification rate. To determine the quality of this classification accuracy, since there are no other classification results on this dataset, we ran multiple experiments to determine the effects of our segmentation algorithm, the distant measures, and the classification techniques have on the final accuracy results.

6.2. ALGORITHM: DYNAMIC TIME WARPING

Dynamic Time Warping (DTW) is an algorithm designed to measure the similarity between two curves, or time-series, which are not necessarily synced in velocity or time. A common example looks at the walking patterns to two individuals. While these two samples may accelerate or decelerate at different times, DTW can determine if (and how much) similarity there is between the two walking patterns.

There are many versions of this technique as it has been the source of many studies [65] [66] [67] [68] and is commonly used in the fields of time-series analysis [69] [70], speech recognition [71] [72] [73], and more recently image- and video-based computer vision [74] [72].

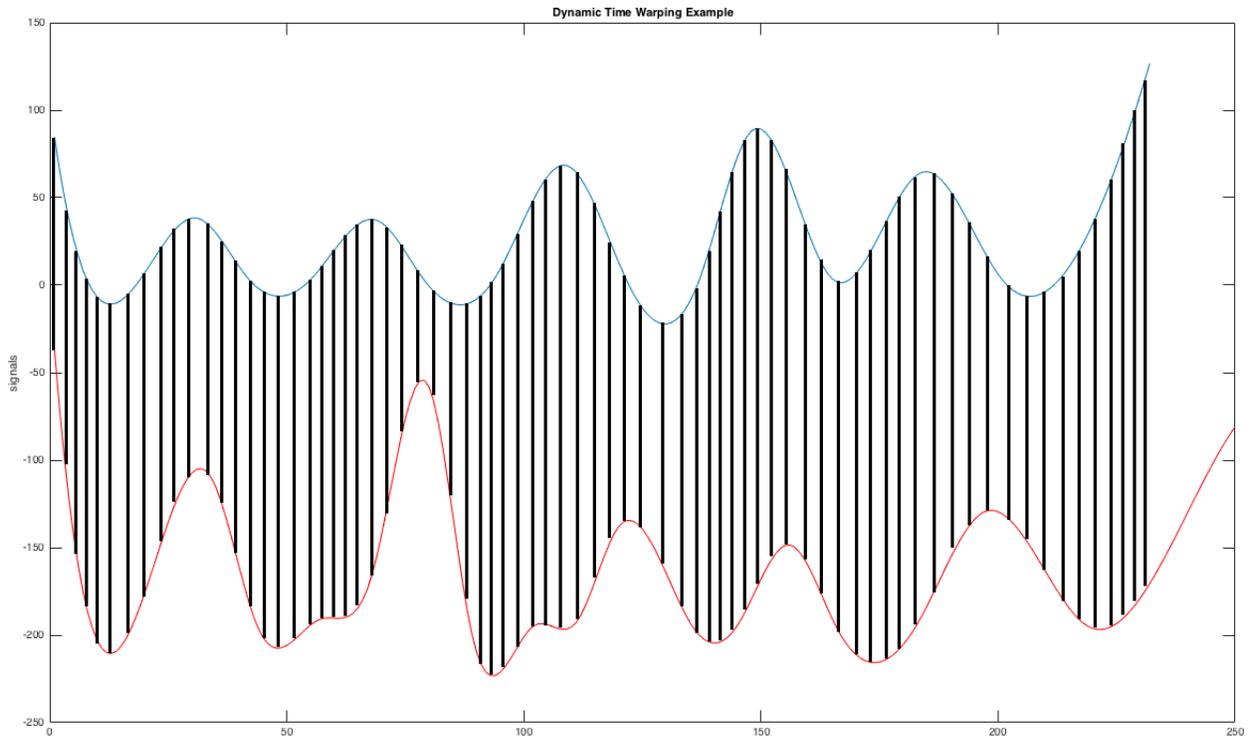


FIGURE 6.1. Dynamic Time Warp counter example. The points on the top signal (blue) are paired with points on the bottom signal (red) by time. This linear based Euclidean distance measure does not align signals based on time and will only compute the distance up to the length of the smallest (in time) curve or time-series. .

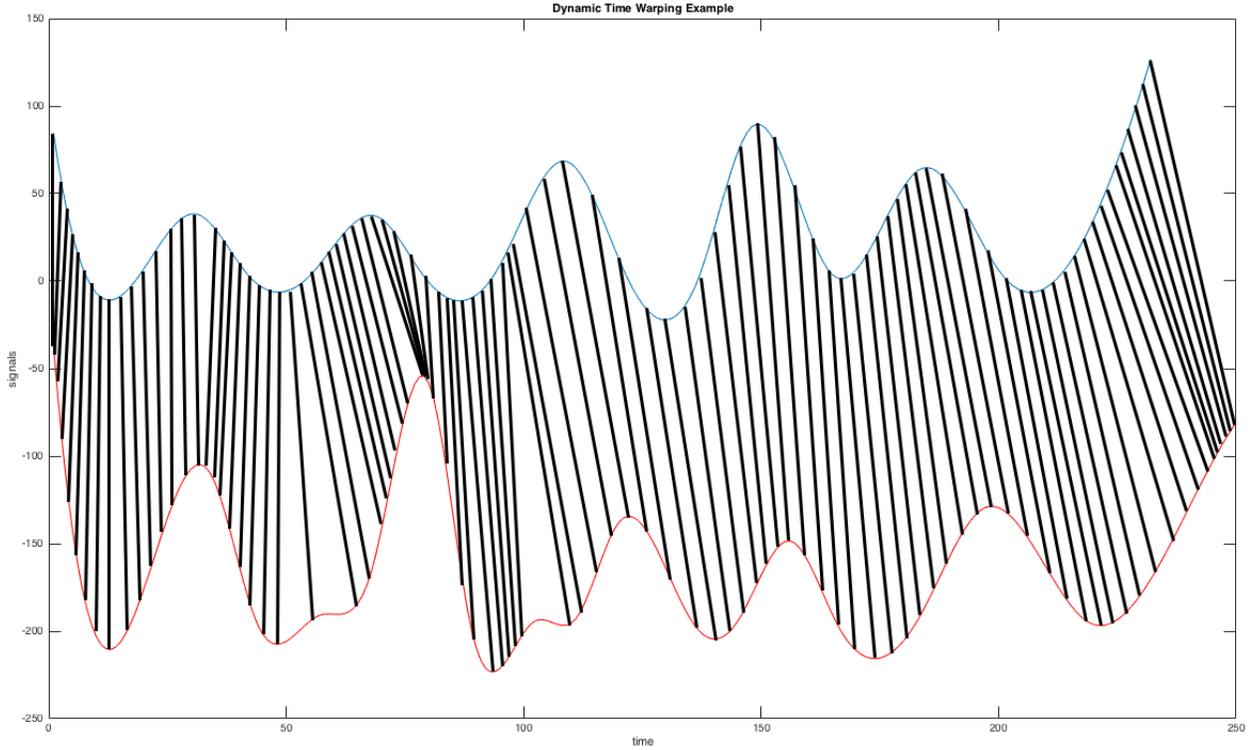


FIGURE 6.2. Dynamic Time Warp example. The points on the top signal (blue) are paired with points on the bottom signal (red). Paired points are described by black lines. This non-linear based Euclidean distance measure aligns signals based on time and provides a better similarity measure.

Consider two signals, $X = \{x_i\}_{i=1}^n$ and $Y = \{y_j\}_{j=1}^m$. Construct a n-by-m matrix, D such that

$$D(i, j) = d(x_i, y_j)$$

where

$$d(x_i, y_j) = (x_i - y_j)^2$$

(the standard l_2 distance). We will define a warping path

$$W = \{w_k\}_{k=1}^K$$

where

$$\max(m, n) \leq K < m + n - 1.$$

The k th element of W is defined as $w_k = (i, j)_k$. This warping path will take the elements of signal X and match them with elements of signal Y . By imposing additional constraints, we can optimize the path taken such that the euclidean distance between all matched pairs are minimized while maintaining monotonically increasing indices from X and Y . In particular, we note several common constraints that are used in the remainder of this chapter:

- **Boundary Conditions:** Define $w_1 = (1, 1)$ and $w_K = (n, m)$.
- **Continuity:** If $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a - a' \leq 1$ and $b - b' \leq 1$.
- **Monotonicity:** If $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$.

Finally, to find the desired path, we wish to solve the optimization problem:

$$\text{DTW}(X, Y) = \min \left(\frac{\sqrt{\sum_{k=1}^K w_k}}{K} \right)$$

An efficient method of solving this optimization problem is achieved through dynamic programming. Lying at the core of the algorithm is finding the cumulative distance

$$\gamma(i, j) = d(x_i, y_j) + \min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\}.$$

Pseudocode for this procedure is found in Algorithm 6.2. We take note that this algorithm runs in $\mathcal{O}(nm)$.

Algorithm 7 Dynamic Time Warping Algorithm without Locality Constraint

```
1: procedure DOUBLE DTW(ARRAY1[1..N], ARRAY2[1..M]
2:   Initialize Dmatrix
3:   for int i from 1 to n do
4:     Dmatrix[i,0] = Inf
5:   end for
6:   for int i from 1 to m do
7:     Dmatrix[0,i] = Inf
8:   end for
9:   Dmatrix[0,0] = 0
10:  for int i from 1 to n do
11:    for int j from 1 to m do
12:      cost = d(array1[i], array2[j]) // where d(x1, x2) is a distance measure
13:      Dmatrix[i,j] = cost + minimum(Dmatrix[i-1,j], insertion
14:                                   Dmatrix[i,j-1], // deletion
15:                                   Dmatrix[i-1,j-1]) // match
16:    end for
17:  end for
18:  return DTW(n,m)
19: end procedure
```

By including additional constraints, we see a special case of this algorithm may be used to compute the distance between two signals, X and Y , where a non-linear warping path is not allowed. In particular, if $n = m$ and w is constrained as

$$w_k = (i, j)_k \text{ where } i = j = k,$$

then we achieve a distance, $DTW(X, Y)$, as shown in Figure 6.1. Note, for testing purposes, if $n \neq m$, we perform a simple linear interpolation of the shorter signal until the condition $n = m$ is met.

With limited *a priori* knowledge of the data, it might be necessary to impose a locality constraint to the algorithm. In particular, if the two signals are very large in length ($n, m \gg \gg 0$) and we wish to determine if the signals have a similar amplitude within small

regions, then the modification presented in Algorithm 8 can be used with greater effectiveness than the algorithm (Algorithm 7) presented above. When computation time is of great importance, the use of this locality constraint can greatly speed up the run time if $n > m$ and $m \gg \gg window$.

Algorithm 8 Dynamic Time Warping Algorithm with Locality Constraint

```

1: procedure DOUBLE DTW(ARRAY1[1..N], ARRAY2[1..M], INT WINDOW
2:   Initialize Dmatrix
3:   window = max(window, abs(n-m))
4:   for int i from 1 to n do
5:     for int j from 1 to m do
6:       Dmatrix[i,j] = Inf
7:     end for
8:   end for
9:   Dmatrix[0,0] = 0
10:  for int i from 1 to n do
11:    for int j from max(1,i-window) to min(m,i+window) do
12:      cost = d(array1[i], array2[j]) // where d(x1, x2) is a distance measure
13:      Dmatrix[i,j] = cost + minimum(Dmatrix[i-1,j], insertion
14:                                   Dmatrix[i,j-1], // deletion
15:                                   Dmatrix[i-1,j-1]) // match
16:    end for
17:  end for
18:  return DTW(n,m)
19: end procedure

```

6.3. CLUSTERING

For the past several decades, clustering has been a commonly employed technique for computer scientists [75], [76]. In essence, clustering is the process of labeling samples such that the number of labels is less than the number of samples where similar samples have the same label.

From Chapter 5 we have small segments extracted from larger videos. Each segment contains a collection of sequential frames that describe the human structure and the movement

of that structure through some time interval. We desired to cluster these segments in such a manner that similar segments (in pose space) are grouped together.

The primary method we use to determine which segments are similar is the dynamic time warping algorithm presented in Section 6.2. Starting with each segment labeled as its own cluster, an agglomerative clustering algorithm - a type of bottom-up clustering mechanic - is used to group segments depicting similar skeleton representation sequences. Figure 6.3 outlines this bottom-up approach.

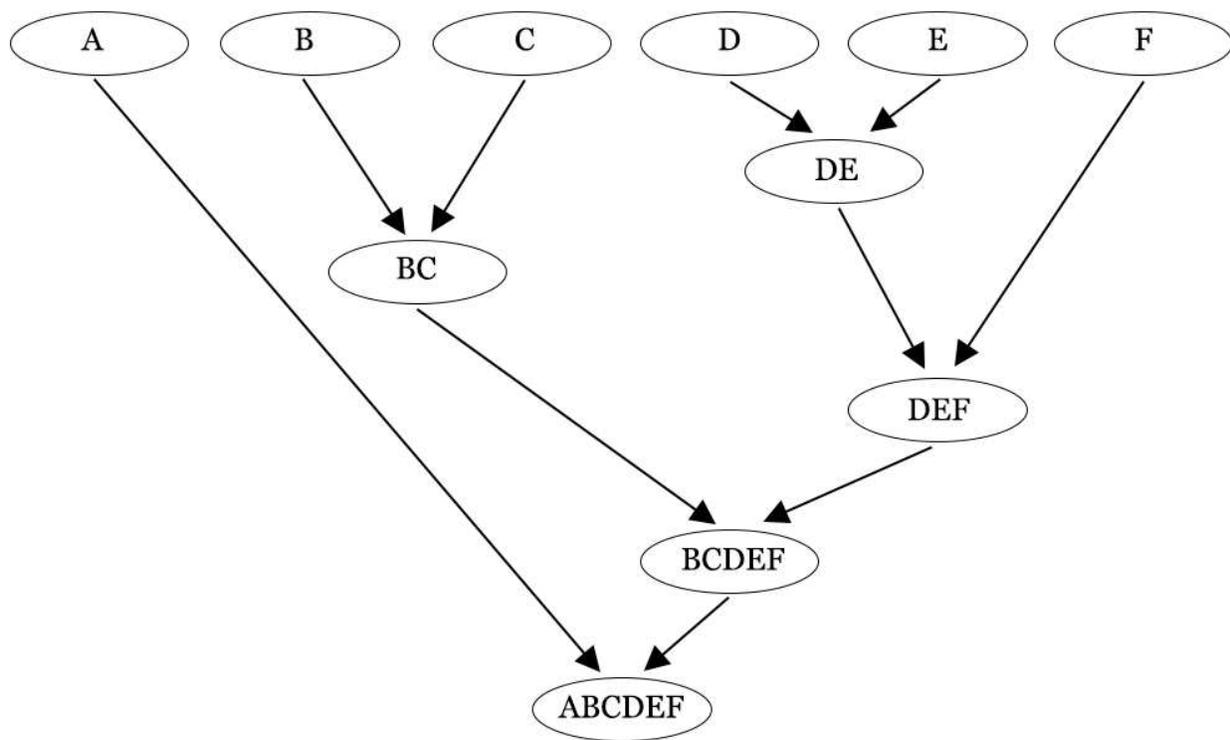


FIGURE 6.3. Agglomerative clustering example. Beginning with each signal, $\{A, B, C, D, E, F\}$ in its own cluster, clusters combine until all elements are in a single cluster.

To describe this process, we begin by constructing a distance matrix between all pairs of elements x_1, \dots, x_n . In order to obtain a desired number of clusters, we will merge the two closest elements together and recompute the distance matrix. Obviously, depending on the data type of each element, we can compute the distance matrix using any appropriate norm,

$d(x, y)$, given each element is a single signal (which we can consider to be a cluster of one element). However, once we start merging data, there are a number of common techniques for computing the distance between clusters, \mathcal{A} and other clusters \mathcal{B} . In particular:

- **K-medoids:** Use a k-means algorithm to estimate the center of the cluster. Use these centers as the new positions.
- **Complete-linkage Clustering:** Use the maximum distance between elements in the clusters

$$\max\{d(x, y) : x \in \mathcal{A}, y \in \mathcal{B}\}.$$

- **Single-linkage Clustering:** Use the minimum distance between elements in the clusters

$$\min\{d(x, y) : x \in \mathcal{A}, y \in \mathcal{B}\}.$$

There are, of course, additional methods to compute the distance between clusters. However, for this dissertation, we only consider these basic techniques for our application of action classification.

Combining clusters one at a time, we continue with the same chosen method until we arrive at the desired number of clusters. In situations that we are training data, the cluster will be labeled as the mode of the labels of the individual segments in each cluster.

Algorithm 9 Agglomerative Clustering

```
1: procedure DOUBLE AGGLOMERATIVECLUSTERING(SAMPLES[1...N], DESIREDNUM-
   CLUSTERS)
2:   while N  $\neq$  desiredNumClusters do
3:     Initialize Distances = Inf
4:     for int i from 1 to N do
5:       for int j from i+1 to N do
6:         measure = d(samples[i], samples[j]) where d is a distance measure
7:         if measure < Distances then
8:           Distances = measure
9:           index1 = i
10:          index2 = j
11:         end if
12:       end for
13:     end for
14:     Update samples such that samples[index1] and samples[index2] are treated as a
   single sample (K-medoids, Complete-linkage, Single-linkage, etc.)
15:   end while
16: end procedure
```

6.3.1. ALGORITHM: AGGLOMERATIVE CLUSTERING.

6.4. HIDDEN MARKOV MODELS

The description in Chapter 5 illustrates the observation that each human action is comprised of one or more atomic motions. Each action class may contain a variable number of segments depending on how the action is performed, the curvature estimates from Algorithm 4 of Chapter 4, and the segmentation results of Chapter 5.

From the clustering algorithm (Algorithm 9), we have separated each segment into a predetermined number of clusters. Each cluster contains segments which are similar, as determined by our dynamic time warping algorithm. The final step to classify these segments is to determine which cluster to combine, and in what order to form conjoined cluster which are pieces of the original actions. The method we have chosen to explore is a conditional probabilistic approach known as Hidden Markov Models (HMMs).

A simple understanding and example of Hidden Markov Models is as follows. Assume for some set of data, there are a total of N possible states, s_1, s_2, \dots, s_N . During every t th time-step, a piece of data is identified as one of these N th states. Given a sample is in State j at time-step t , we know the next state is randomly chosen. The current state j will determine the probability of entering each other state at time $t + 1$. However, it is only the current state that impacts the probabilities of entering other states. Another way to say this is that given $q_t \in \{s_1, \dots, s_N\}$, q_{t+1} is conditionally independent of $\{q_{t-1}, q_{t-2}, \dots, q_1\}$.

6.4.1. A BASIC HMM EXAMPLE. Consider the following situation: while working on a project, you find your productivity increases when sampling beverages. The possible choices are beer, wine, or whiskey. While working 18 hour days, you will understandably consume more than one beverage. However, there are some conditions (based on personal preferences) placed on the order $\{w_1, w_2, \dots, w_t\}$ in which you enjoy these drinks. The following probability table (Table 6.4.1) describes your preferences.

TABLE 6.1. Probabilities between all possible states in a Markov Model Example. Current state (left) future state (top).

	Beer	Wine	Whiskey
Beer	0.5	0.1	0.4
Wine	0.05	0.8	0.15
Whiskey	0.15	0.15	0.7

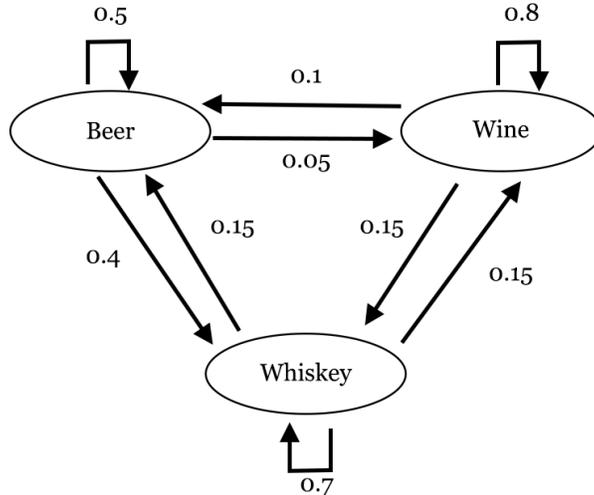


FIGURE 6.4. Probability plot of moving from one state to another in Markov Model Example.

From this we can determine the probability that if you are currently drinking whiskey, you will next drink beer, followed by another whiskey as

$$\begin{aligned}
 &P(w_2 = \text{beer}, w_3 = \text{whiskey} | w_1 = \text{whiskey}) \\
 &= P(w_3 = \text{whiskey} | w_2 = \text{beer}, w_1 = \text{whiskey}) * P(w_2 = \text{beer} | w_1 = \text{whiskey}) \\
 &= P(w_3 = \text{whiskey} | w_2 = \text{beer}) * P(w_2 = \text{beer} | w_1 = \text{whiskey}) \\
 &= (0.4)(0.15) \\
 &= 0.06
 \end{aligned}$$

This process describes a Markov Model. To extend this example to an example of a Hidden Markov Model, start by assuming that the type of drink you are enjoying is unknown to you (for one reason or another). However, by observing the color of the liquid, you can assign probabilities based solely upon this color (for example, a light brown liquid is 60% likely to be beer, 30% whiskey, and 10% wine). Using these probabilities and without knowing

the type of current liquid, we can still construct the probabilities of any sequence of drinks. Since the current state is unknown (but based on probabilities of an outside observation) our model is now considered to be an HMM.

6.4.2. HMMS ON PALKA. In order to use HMMs to classify unknown segments, we need to first build a set of HMMs for each action class. Using training data (where the labels are known) we can determine which cluster segments are identified with each action class. Given the similarities between different clusters (due to the large number of clusters and the fact that cluster segments may be used for multiple actions) we can determine the probability of going from one cluster to another using the ground truth labeling.

For example, consider the actions of ‘Lift Outstretched Arms’ and ‘Had Enough’. Let an instance of lift outstretched arms be segmented into 4 pieces. The first piece starts from the base state and contains segments that show the arms rising followed by the second state of the hands meeting above the hear. The first piece of goggles also shows the arms rising followed by the second state of the hands meeting on top of the head while the head is extended forward. The cluster containing similar segments of the arms raising can then be followed by either a cluster containing the hands meeting above the head or by the hands meeting on the head while the head lowers. By knowing the true labels in each cluster, we can determine the number of times those labels go into the first cluster or the second. This creates a probability distribution based on only the initial ‘raising arms’ cluster.

Once we build probabilities from each state (or cluster) to every other state based upon ground truth labeling we can then bring in the data from the testing set. By clustering these testing videos in the same manner, we use the probabilities we determined between cluster to apply labels obtained from the training probability distribution web. Finally, since the

purpose is to determine the quality of these labels, we use ground truth on the testing data to see if the assigned labels match with the ‘true’ or known labels.

6.5. CLASSIFYING PALKA HUMAN ACTIONS

Tying together the entire of work presented in this dissertation, we finally perform the task of action classification using the full PALKA dataset (see Appendix: A). Given several of the algorithms presented in this body of work depend on sets of parameters, for our final classification results, we choose the parameters found to give the best classification results.

Figure 6.6 displays classification accuracies using several different methods of segmenting the original PALKA videos as well as different methods of computing clusters. For all of these techniques, we first establish a set of preprocessing steps. In each frame of every video in 3-dimensional pose space...

- (1) ...the skeleton is translated such that base_spine joint is centered at $(0, 0, 0)$.
- (2) ...the skeleton is rotated so the vector formed by the base_spine joint and mid_spine joint points in the direction of the vector $\langle 0, 1, 0 \rangle$.
- (3) ...the skeleton is rotated such that the vector normal from the plane formed by the base_spine, right_shoulder, and left_shoulder is pointed in the direction of the vector $\langle 1, 0, 0 \rangle$.
- (4) ...we ensure the skeleton is facing the correct direction by testing to see if the z -coordinate of the left_hip is always smaller than the z -coordinate of the right_hip. If it is not, the skeleton is rotated along the appropriate axis by π .

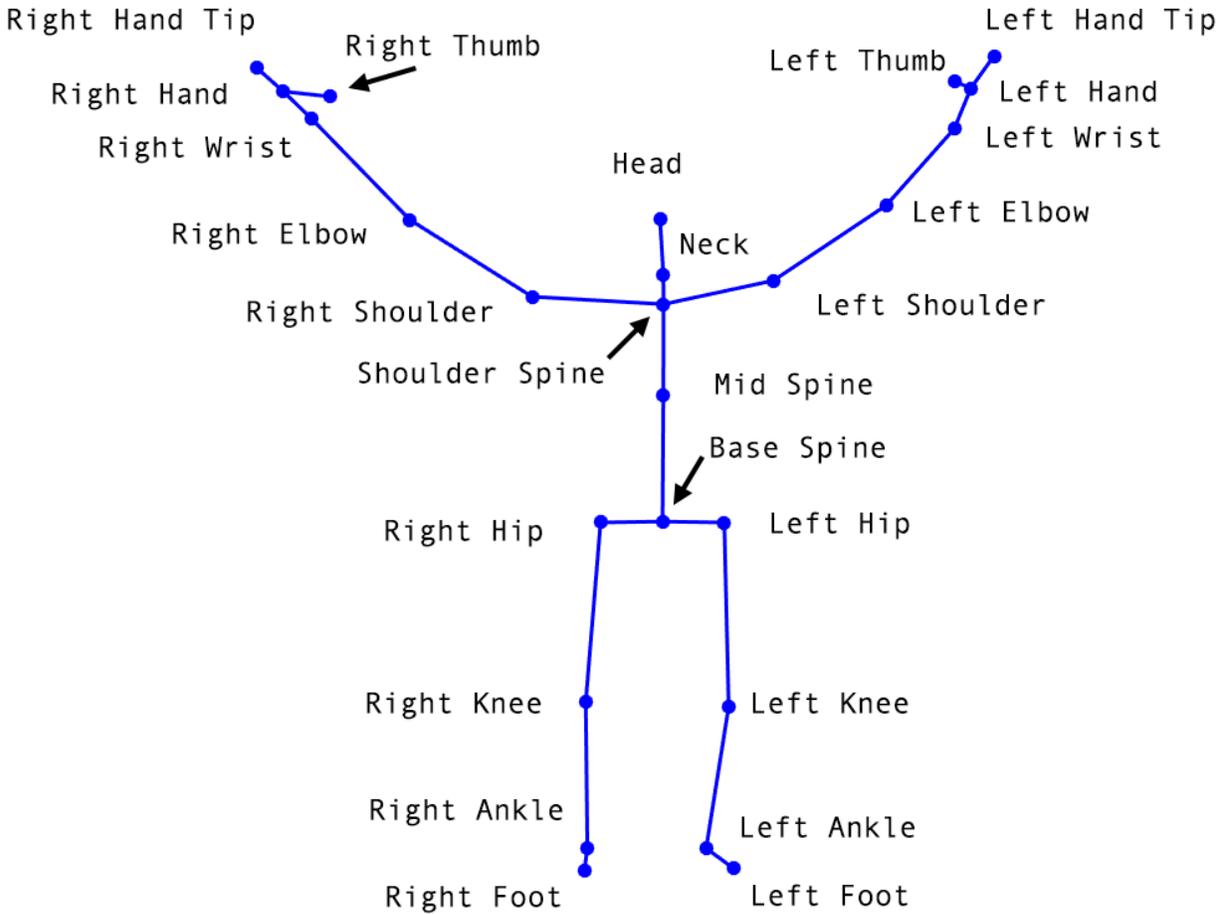


FIGURE 6.5. Diagram of Skelton produced by skeletonization algorithm from the Kinect 2 sensor. Labels for all 25 identified body points as defined by the Kinect SDK.

From these initial pre-processing steps, we then start to perform the computations leading to classification results (see Figure 6.6).

6.5.1. CURVATURE CLASSIFICATION. Curvature classification uses the curvature algorithm (Algorithm 4) to help define motion segments (Algorithm 6) which are then clustered and assigned an action label.

- (1) Starting with the preprocessing steps above, we use the ACE-PC curvature algorithm with the adaptive cutoff value set to $h = 0.25$. These curvature profiles are then used to segment the videos.

- (2) All peaks above a threshold value of 19 are found. The motion transitions are determined to be the location of the peaks \pm the width of the peaks. All isolated motion-declared frames are set as transitions. If there are any transition sequences (including sequences of length 1) less than 4 frames long, these sequences are redefined as motions. Segmentation occurs on the boundaries between declared motions and transitions.
- (3) Using $d(x, y) = (x - y)^2$, a DTW score is computed for all pairs of segments.
- (4) These distances are clustered using a k-medoids agglomerative clustering technique with a number of different output clusters $\{50, 100, 150, 200, 250\}$. Since the k-medoids technique relies on randomly generated initial conditions, for each set of declared output clusters, experiments were performed 10 times and the results were averaged together.
- (5) Hidden Markov Models were trained for all action classes within the PALKA dataset using a leave-one-person-out scheme. The left-out person was then tested on these models. All combinations of one-person-left-out were tested and the accuracy of the classifications were averaged as a function of the number of declared clusters from the prior step.

6.5.2. FRAME CLASSIFICATION. Frame Classification ignores the use of curvature computations on the data. Instead, segmentation is performed by setting each individual frame as its own segment. Steps 3 – 5 from Subsection 6.5.1 were performed on these single-frame segments. Noting here, due to each segment containing only one frame, the DTW score for each pairwise segment is the standard euclidean distance.

6.5.3. **RANDOM LENGTH CLASSIFICATION.** Random Length Classification also ignores curvature results for segmentation purposes. Instead, for each video, the number of segments given by motion-based ground-truth is computed. The video is then segmented into that many segments of random length. Steps 3 – 5 from Subsection 6.5.1 were performed on these random length segments.

6.5.4. **INTERPOLATION CLASSIFICATION.** Interpolation Classification starts by using ACE-PC found curvatures to segment each video as described in Steps 1 – 2 from Subsection 6.5.1. We eliminate the use of a dynamic time warping algorithm by performing a linear interpolation of each segment to achieve a uniform segment length of 150 frames. The distance matrix used by the clustering algorithm is computed as the special case identified in the Section 6.2 where a non-linear warping path is not allowed. Steps 4 – 5 from Subsection 6.5.1 were performed using this averaged, direct frame euclidean distance matrix.

6.5.5. **RANDOM RESULTS.** For completion, and to compare these various methods, we state that given the PALKA dataset contains 12 action classes, a completely random classification assignment would yield an accuracy of

$$\frac{100}{12} = 8.\bar{3}\%.$$

6.5.6. **CLASSIFICATION RESULTS.** The methods shown in Figure 6.6 were chosen such that we could perform an analysis of the numerous algorithms that were used to classify human actions from the skeleton data of the PALKA dataset.

An examination of the 5 methods presented above describe the various effect of the contributions to this dissertation. Starting with the most basic approach, if we ignore any logical efforts to correctly classify data, we mathematically end up with a 8.3% chance of

correctly classifying the data. It is easy to see that using data containing a greater number of action classes will decrease random guessing in a predictable manner.

The driving force behind our theories of using proper segmentation as a precursor to action classification is demonstrated by comparing the frame, random length, interpolation, and curvature results. By segmenting the videos into pieces of random length, the only tools used for classification are pre-existing, and well-established, techniques common to the field of computer vision. The 72% accuracy results seen from this method, when compared to high accuracies of curvature segmented videos, illustrates that proper segmentation can be used to great affect for this task. We note the 76% and 82% accuracies of interpolation and curvature resp. both use curvature based segmentation.

The difference between interpolated (76%) and curvature (82%) are all based on the method used to compute distances between segments. By interpolating each segment to a prescribed number of frames, we over emphasize the temporal importance of the curvature based segmentation.

For example, consider two instances of the action ‘Wind It Up’. This action is performed by “With initial motion to the back, swing the arm in three full circles without stopping”. Due to inherent errors in the Kinect SDK skeletonization algorithm, one instance of this action may have 2 spots of high curvature (where part of the arm is obstructed from the sensor). In the second instance of this action, the person is at a slight angle to the sensor and therefore the arm is not obstructed. This results in a constant curvature for the entire action. Segments in the first case would natively be about 40 frames where in the second case, the segment is around 120 frames. Despite the highly similar nature between the two instances of the action, by interpolating the segments to equal length, the linear warping map computes a euclidean distance between the two segments that is extremely high. On

the other hand, using a dynamic time warping algorithm on the two segments, especially if the boundary conditions are not enforced, the result would be a near perfect match (small distance).

Finally, we consider the frame-by-frame method. As a counter to the pattern formed by the other method, this appears to excel when coarsely clustered. A more detailed investigation is needed to understand this approach. However, a working theory is the action performed in the PALKA dataset can largely be defined by unique poses. By segmenting each video into individual frames, we have created a pose detector. Given a dataset with more similar actions, we expect the accuracy of this method to decrease. We do note that the accuracy from the curvature experiments (82%) is still higher than the maximum obtained by the frame method (80%).

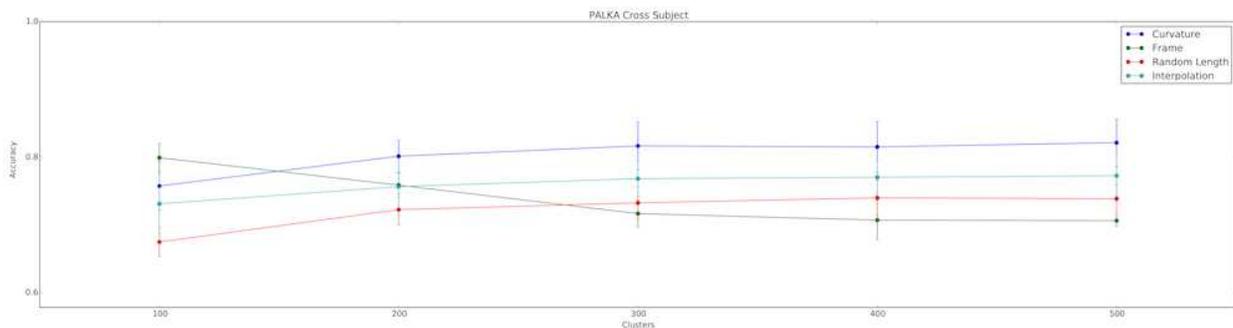


FIGURE 6.6. Comparison of various techniques to classify PALKA actions as a function of the number of declared clusters. The blue line shows the accuracy as the original videos are segmented by method described in Chapter 5 using Algorithms 4 and 6 to compute curvature. The green line shows classification accuracy using each individual frame as its own segment. The red line shows the accuracy if each of the original videos are divided into random segments. And the teal line takes each segment as defined by the method described in Chapter 5 using Algorithms 4 and 6 to compute curvature. Then each segment is extended to a fixed length using linear interpolation. An averaged euclidean distance is used instead of a DTW algorithm. Also, not shown is completely random results. As there are 12 action classes in PALKA, perfectly random results are computed as $8.\bar{3}\%$.

6.6. CONCLUSION

This chapter of this dissertation was designed to show the completion of our initial motivating task: action classification using a bottom-up curvature-based approach. Starting with curvature segmented motions, we employed a common computer vision techniques of creating a testing and training data set. Then, we used curvature estimations to segment large videos, curve-based distance measure (DTW), agglomerative clustering techniques and finally Hidden Markov Models to automate the assignment of action labels to individual motions.

Our preliminary results on the PALKA dataset, a Kinect 2 dataset which only contain skeleton data describing a highly restrictive list of human actions, is very promising (82% accuracy). Through a detailed study on this dataset, we identified the importance of using properly segmented data. We also identified the role the pure classification techniques (DTW and clustering) are to the final accuracy results. From our final experiments, we conclude the methodology established in this dissertation for the task of motion segmentation and action classification is worthy of future attention and exploration.

CHAPTER 7

CONCLUSIONS

This dissertation presents the research and necessary background information related to the formulation and use of a numerically stable method of computing, and using, generalized curvature values based upon the singular value decomposition.

In Chapter 2 we built up the background concepts required to understand the properties of curves relating to the rest of the dissertation. While the concepts of curvature, and by extension, generalized curvatures, have been well-known for over 150 years, these formulations require having an analytical, smooth function to work on.

Chapter 3 introduced the idea of estimating generalized curvature values when an analytical, smooth function is not known. Starting with the continuous case, we established a method using the KL-transform to compute generalized curvature values using a closed-form equation using scaled eigenvalues. The discrete form of the KL-transform is the Singular Value Decomposition (SVD). Using this algorithm, we transformed the closed-form equation using scaled eigenvalues to a closed-form equation using local singular values. Finally, by generating a time-series by discretely sampling a known curve with known generalized curvature values, we provided an example showcasing the accuracy of this method where the signal has a lack of noise.

However, when the discrete approximation to a curve contains noise, the estimates produced by the algorithm in Chapter 3 are both noisy and inaccurate. By varying the window size used in the local SVD, we discovered a significant increase in accuracy by reducing the number of points used by the SVD in segments of the time-series characterized by high generalized curvature values. We also found a significant increase in accuracy by increasing

the number of points used by the SVD in segments of the time-series characterized by small generalized curvature values. Chapter 4 contains the formulation of two modifications to the algorithm proposed in Chapter 3. One modification is the development and inclusion of an adaptive window size selector to increase accuracy of the curvature estimations as just described. The second is a modification to the generalized curvature algorithm which assumes the underlying curve is not known. By changing the properties of the data fed into the local SVD, we are able to increase the precision of the estimates. Using a toy problem, we showcase the error in estimates as produced by the method described in Chapter 3 alongside the errors from both proposed modifications in Chapter 4, as well as the prior SOA technique of computing derivatives numerically. Here, we show our numerical techniques for computing generalized curvature values to be several orders of magnitude more precise and accurate than previous methods.

The initial goal of this project was to create techniques to be used for the task of human action segmentation; a non-traditional first step for the application of human action classification/detection. Using human action data collected from a Kinect 2 sensor, we applied the mathematical algorithms developed in Chapters 3 and 4. In Chapter 5, we explore various uses of curvature in human activity data. While some results of the simulations we ran, such as the use of higher dimensional curvatures have not yet been fully understood, we found compelling evidence to suggest we can segment data streams of human activity to the human motion level. After creating our own, challenging, dataset (as described in Appendix A), we used our curvature algorithms, in part, to segment the data with a 83.8% accuracy.

We recognize the computer vision community has not shown any interest in motion segmentation or motion recognition. A common problem of interest within the community, instead, is action recognition/classification. In Chapter 6, we started with the results of the

human motion segmentation of Chapter 5 and used a Dynamic Time Warping algorithm to compare motion segments. A clustering algorithm was used on the distance matrix from comparing all pairs of motion segments. Then these clusters were used to develop a set of Hidden Markov Models. This allowed us to group temporally adjacent motions together into actions. It also allowed us to create a training/testing system which will classify any human activity data streams (collected by the same sensor) into recognized human actions. While work on this task is still fairly new, and only preliminary results have been reported, the output shows enough promise to continue pursuing this goal in future work.

This dissertation describes a lot of new and exciting techniques which we have already introduced to computer vision applications. However, in creating the new mathematical techniques with the desire to use them in this project, there are still many unanswered questions suitable for future research topics. In particular: can the Frenet Frame generated by the subspace of best fit in a local region provide any useful information? can these subspaces, or generalized curvature values be of use to areas outside of the computer vision community (such as an analysis of time-series on Grassmannians)? can the conjecture in Chapter 3 be proven? and when looking at human activity data, why are higher dimensional curvature estimations so highly correlated? These are examples of questions posed by this dissertation which call for a further study.

BIBLIOGRAPHY

- [1] Cepheus, “Osculating circle for curvature.” Public Domain Image, November 2006.
- [2] C. Jordan, “Sur la théorie des courbes dans l’espace à n dimensions,” *Comptes Rendus*, 1874.
- [3] C. F. Gerald and P. O. Wheatley, *Applied Numerical Analysis*. Pearson, 2004.
- [4] 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.
- [5] W. Kühnel, *Differential geometry: curves-surfaces-manifolds*, vol. 16. American Mathematical Soc., 2006.
- [6] S. Lang, *Undergraduate Analysis*. Springer, 2nd ed. ed., 1997.
- [7] M. Spivak, “Differential geometry, volume 1. publish or perish,” *Inc.*, 1979.
- [8] R. Sulanke, “The fundamental theorem for curves in the n -dimensional euclidean space.” library.wolfram.com, 2009.
- [9] R. Arn, B. Draper, M. Kirby, and C. Peterson, “The frenet-serret apparatus and local singular value decomposition of curves in \mathbb{R}^n ,” *arXiv preprint arXiv:1511.05008*, 2015.
- [10] D. Broomhead, R. Indik, A. Newell, and D. Rand, “Local adaptive Galerkin bases for large-dimensional dynamical systems,” *Nonlinearity*, vol. 4, no. 2, p. 159, 1991.
- [11] D. Hundley, M. Kirby, and R. Miranda, “Empirical dynamical system reduction II: Neural charts,” in *Semi-analytic methods for the Navier–Stokes equations (Montreal, 1995)* (K. Coughlin, ed.), vol. 20 of *CRM Proc. Lecture Notes*, (Providence, RI), pp. 65–83, Amer. Math. Soc., 1999.
- [12] D. Broomhead and M. Kirby, “A new approach for dimensionality reduction: Theory and algorithms,” *SIAM J. of Applied Mathematics*, vol. 60, no. 6, pp. 2114–2142, 2000.

- [13] M. Kirby, *Geometric data analysis: an empirical approach to dimensionality reduction and the study of patterns*. John Wiley & Sons, Inc., 2000.
- [14] A. V. Little, J. Lee, Y.-M. Jung, and M. Maggioni, “Estimation of intrinsic dimensionality of samples from noisy low-dimensional manifolds in high dimensions with multiscale SVD,” in *Statistical Signal Processing, 2009. SSP’09. IEEE/SP 15th Workshop on*, pp. 85–88, IEEE, 2009.
- [15] D. Broomhead, R. Jones, and G. P. King, “Topological dimension and local coordinates from time series data,” *Journal of Physics A: Mathematical and General*, vol. 20, no. 9, p. L563, 1987.
- [16] S. J. Cyvin and I. Gutman, *Kekulé structures in benzenoid hydrocarbons*, vol. 46. Springer Science & Business Media, 2013.
- [17] P. Aluffi, “Degrees of projections of rank loci,” *arXiv preprint arXiv:1408.1702*, 2014.
- [18] R. Arn, B. Draper, M. Kirby, and C. Peterson, “Curvature estimation for temporal segmentation.” Submitted to ECCV 2016, March 2016.
- [19] F. J. Solis, “Geometry of local adaptive galerkin bases,” *Applied Mathematics and Optimization*, vol. 41, pp. 331–342, 2000.
- [20] R. Arn, B. Draper, M. Kirby, and C. Peterson, “The frenet-serret apparatus and local singular value decompositions,” *submitted to the Advances in Geometry*, 2016.
- [21] T. Hastie and W. Stuetzle, “Principal curves,” *Journal of the American Statistical Association*, vol. 84, pp. 502–526, 1989.
- [22] W. Kühnel, *Differential geometry: curves-surfaces-manifolds*. American Mathematical Society, 2006.
- [23] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE MulitMedia*, vol. 19, pp. 4–10, 2012.

- [24] *ASUS Xtion PRO LIVE*. accessed 2/23/2016.
- [25] *Leap Motion*. accessed 2/23/2016.
- [26] *Intel RealSense Technology*. accessed 2/23/2016.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] G. Yu, Z. Liu, and J. Yuan, “Discriminative orderlet mining for real-time recognition of human-object interaction,” in *Computer Vision–ACCV 2014*, pp. 50–65, Springer, 2014.
- [31] H. Shuzi, Y. Jing, and C. Huan, “Human actions segmentation and matching based on 3d skeleton model,” in *Control Conference (CCC), 2013 32nd Chinese*, pp. 5877–5882, IEEE, 2013.
- [32] 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*, pp. 69–75, IEEE, 2014.
- [33] 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*, pp. 20–27, IEEE, 2012.

- [34] 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.
- [35] 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.
- [36] 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*, pp. 812–819, 2014.
- [37] 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.
- [38] 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*, pp. 147–156, ACM, 2011.
- [39] 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*, pp. 1–7, IEEE, 2008.
- [40] 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.
- [41] 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.

- [42] A. Vögele, B. Krüger, and R. Klein, “Efficient unsupervised temporal segmentation of human motion,” in *Symposium on Computer Animation*, pp. 167–176, Citeseer, 2014.
- [43] 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.
- [44] J. J. Koenderink, *Solid Shape*. Cambridge, MA: MIT Press, 1990.
- [45] S. W. Zucker, “Differential geometry from the frenet point of view: Boundary detection, stereo, texture and color,” in *Handbook of Mathematical Models in Computer Vision*, pp. 357–373, Springer US, 2006.
- [46] J. Pegna, *Variable sweep geometric modeling*. UMI, 1988.
- [47] W. F. Bronsvoort and F. Klok, “Ray tracing generalized cylinders,” *ACM Transactions on Graphics (TOG)*, vol. 4, no. 4, pp. 291–303, 1985.
- [48] 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*, pp. 96–103, IEEE, 1993.
- [49] M. G. Wagner and B. Ravani, “Curves with rational frenet-serret motion,” *Computer Aided Geometric Design*, vol. 15, no. 1, pp. 79–101, 1997.
- [50] 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.
- [51] Z. Duric, E. Rivlin, and A. Rosenfeld, “Understanding object motion,” *Image and vision computing*, vol. 16, no. 11, pp. 785–797, 1998.
- [52] 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.

- [53] 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.
- [54] S. Chern, “Moving frames,” *The Mathematical Heritage of Élie Cartan (Lyon, 1984), Astérisque*, vol. 1985, pp. 67–77, 1985.
- [55] C. Qu, “Invariant geometric motions of space curves,” in *Computer Algebra and Geometric Algebra with Applications*, pp. 139–151, Springer, 2005.
- [56] 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 (IS-CAS), 2015 IEEE International Symposium on*, pp. 1198–1201, IEEE, 2015.
- [57] 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.
- [58] 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.), pp. 1737–1746, ACM, 2012.
- [59] A. Taha, H. H. Zayed, M. E. Khalifa, and E.-S. M. El-Horbaty, “Skeleton-based human activity recognition for video surveillance,” *International Journal of Scientific and Engineering Research*, vol. 6, no. 1, 2015.
- [60] H. Koppula and A. Saxena, “Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation,” *Proceedings of the 30th International Conference on Machine Learning*, 2013.
- [61] J. Wang, “Mining actionlet ensemble for action recognition with depth cameras,” *Computer Vision and Pattern Recognition*, 2012.

- [62] R. Poppe, “A survey on vision-based human action recognition,” *Image and vision computing*, vol. 28, no. 6, 2010.
- [63] V. Parameswaran and R. Chellappa, “View invariance for human action recognition,” *International Journal of Computer Vision*, vol. 66, no. 1, 2006.
- [64] P. Kumar, “Consistent hidden markov models,” Master’s thesis, Colorado State University, 2014.
- [65] C. A. Ratanamahatana and E. Keogh, “Everything you know about dynamic time warping is wrong,” *Third Workshop on Mining Temporal and Sequential Data*, 2004.
- [66] M. Muller, “Dynamic time warping,” *Information retrieval for music and motion*, 2007.
- [67] C. A. Ratanamahatana and E. Keogh, “Three myths about dynamic time warping data mining,” *Proceedings of SIAM International Conference on Data Mining*, 2005.
- [68] P. Senin, “Dynamic time warping algorithm review,” *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 2008.
- [69] E. J. Keogh and M. J. Pazzani, “Scaling up dynamic time warping to massive datasets,” *Principles of Data Mining and Knowledge Discovery*, vol. 1, no. 11, 1999.
- [70] S. Salvador and P. Chan, “Toward accurate dynamic time warping in linear time and space,” *Intelligent Data Analysis*, vol. 11, no. 5, 2007.
- [71] L. Muda, M. Begam, and I. Elamvazuthi, “Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques.” arXiv preprint arXiv:1003.4083, 2010.
- [72] C. S. Myers and L. R. Rabiner, “A comparative study of several dynamic time-warping algorithms for connected-word recognition,” *Bell System Technical Journal*, vol. 60, no. 7, 1981.

- [73] A. Kassidas, J. F. MacGregor, and P. A. Taylor, "Synchronization of batch trajectories using dynamic time warping," *AiChE Journal*, vol. 44, no. 4, 1998.
- [74] Kovacs-Vajna and Z. Miklos, "A fingerprint verification system based on triangular matching and dynamic time warping," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2000.
- [75] K. C. Gowda and G. Krishna, "Agglomerative clustering using the concept of mutual nearest neighbourhood," *Pattern recognition*, vol. 10, no. 2, 1978.
- [76] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000.

APPENDIX A

DATASETS

A.1. MSRC-12

In 2012, Microsoft issued an action recognition challenge using a dataset called MicroSoft Research Challenge 2012 (MSRC-12) [58]. This dataset is comprised of body-joint coordinate videos of people performing one of twelve actions. The list of these actions are found in Table A.1.

TABLE A.1. MSRC-12 Ordered Human Action List

1. Lift Outstretched Arms	5. Wind it Up	9. Had enough
2. Duck	6. Shoot	10 Change Weapon
3. Push Right	7. Bow	11. Beat Both
4. Goggles	8. Throw	12. Kick

Each video of the MSRC-12 dataset contained a person performing one of these action 8 – 10 times each, with a rest, or pause, in-between each action. With this dataset, the problem of segmenting the data, so that each instance of an action is its own video, is trivial. When using a simple computation of the l_2 distance between neighboring frames (to approximate the velocity of the actor), it is obvious, based upon a threshold of that velocity, where each action begins and ends. However, this technique cannot be used with as much success when actions do not have a substantial pause separating actions.

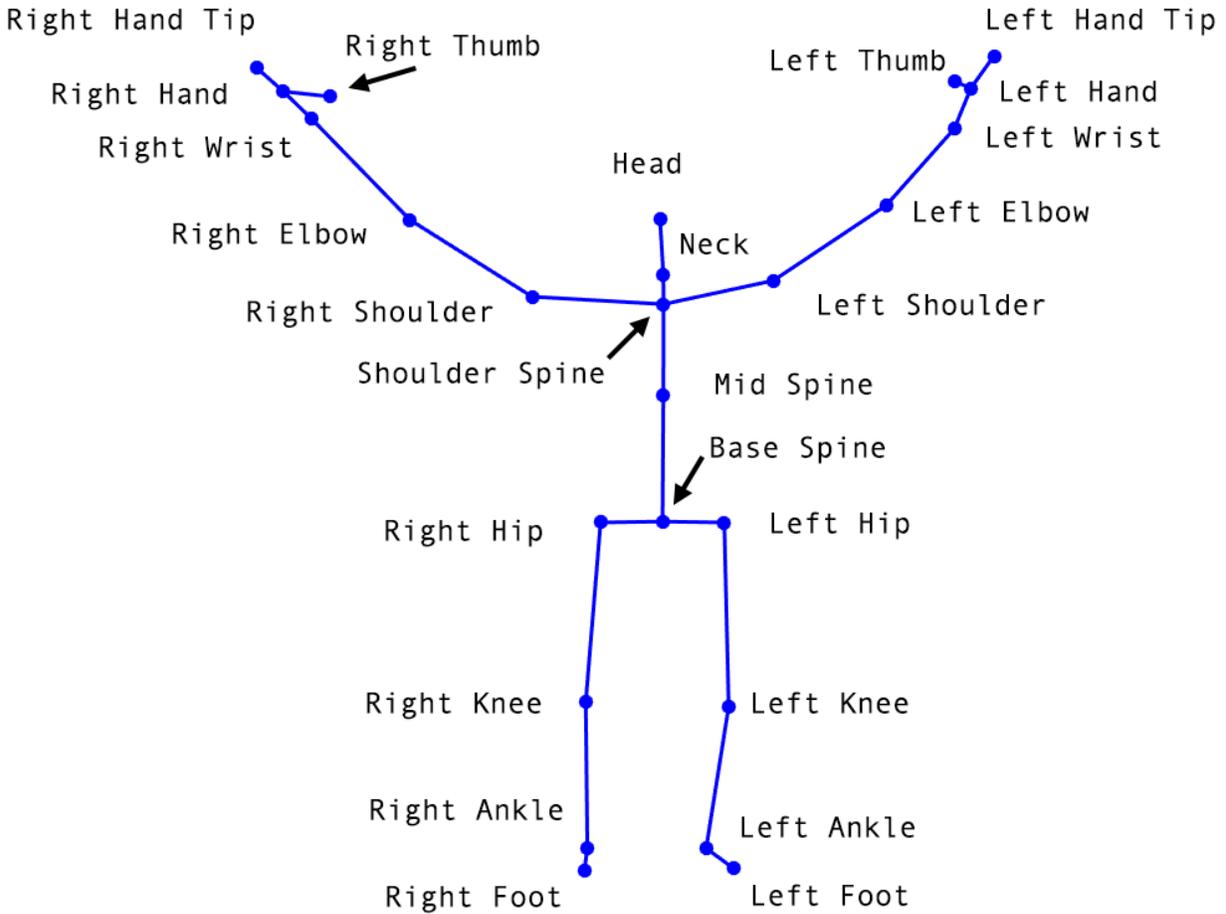


FIGURE A.1. Diagram of Skelton produced by skeletonization algorithm from the Kinect 2 sensor. Labels for all 25 identified body points as defined by the Kinect SDK.

A.2. PATTERN ANALYSIS LABORATORY KONTINUOUS ACTIONS (PALKA)

The Pattern Analysis Laboratory Kontinuous Actions (PALKA) dataset was created to test new algorithms in motion/action segmentation and classification. Common publicly available dataset at the time of creation (late-2014) were not suitable for this task due to a lack of appropriate ground truth. This dataset contains two forms of ground truth: action- and motion- based. Each video contains multiple actions. The action-based ground truth gives the frame numbers which describes a temporal location where the entirety of an action begins and/or ends. These locations are labeled using the 12 action classes described by the

MSRC-12 dataset. The motion-based ground truth splits the video into “inAction” segments or “inTransition” segments. The “inAction” segment are the frames where there is significant motion related to an action. The “inTransition” segments describes temporal locations where the actor is changing between atomic motions. The “inAction” and “inTransition” segments span the entirety of each video.

A.2.1. COLLECTION DETAILS. This dataset contains a total of 47,644 frames in 234 videos. Videos contain between 102 and 382 frames each. Each video contains 3 actions (as described in the Data Action Descriptions section) in the form Action A > Action B > Action A. For example, video 4 has the actor performing Lift Outstretched Arms, followed by Duck, followed by Lift Outstretched Arms. Each set of A>B>A were performed by 3 different actors. Before performing these actions, actors were verbally instructed which actions to perform and each actor saw a visual demonstration of the actions in order to minimize variability. In order to make the action segmentation problem hard, actors were instructed not to pause between individual motions or actions. The actions chosen for this dataset are uniformly described versions of the actions shown in the MSRC-12 dataset.

Data was collected using a Microsoft Kinect 2 device. The skeletons were extracted from RGB-D data using the Kinect for Windows SDK 2.0.

Ground truth was labeled in two separate ways. The traditional method of labeling human activity dataset, that is, to label each frame by the action, has been performed. However, we have also labeled the data at the motion level as well. This involved the author spending 28 long, tedious hours hand labeling the 47,644 frames contained in the 234 videos. As described above, motion-based ground truth marks the temporal locations of the start of motions and the start of transitions between motions. However, only the action-class

was provided for these motion and transition segments. Motion-class information was not collected.

A.2.2. DATA ACTION DESCRIPTIONS. Neutral State: Relaxed standing position with arms by the side facing forward.

- Lift Outstretched Arms: Abduct both arms out to the side bringing both the hands overhead. Reverse the process to return to a neutral state.
- Duck: Flex the knees and hips, lowering the body. Slightly flex the shoulder forward to keep balance. Then immediately extend the knees and hips while extending the shoulders to return to the neutral state.
- Push Right: Raise the right arm, bringing it as far across the front of the body near shoulder level while keeping the elbow extended. Reverse the process to return to a neutral state.
- Goggles: Simultaneously flex both shoulders and elbows to raise the hands to the eyes. Reverse the process to return to a neutral state.
- Wind it Up: With initial motion to the back, swing the arm in three full circles without stopping.
- Shoot: Raising both arms sim. while keeping the elbows extended. Bring the hands together, mimicking the shooting of a pistol with kickback, producing slight elbow flexion. Return to a neutral state.
- Bow: Keeping the legs extended, flex the hips and spine, lowering the upper body forward to a bowed position. Reverse the process to return to a neutral state.
- Throw: Using only the left arm, raise the hand above and behind the left shoulder with a flexed elbow. Then extend the elbow and shoulder to move the hand as far forward as possible. Return to a neutral state.

- Had Enough: Simultaneously flex both shoulders and elbows, raising the hand to a bowed forehead. Reverse the process to return to a neutral state.
- Change Weapon: Flexing the shoulder and elbow, raise the left arm bringing the hand over the opposite shoulder. Mimic grabbing a weapon from the back. Next simultaneously move the right hand to the mid-chest in the front by extending the elbow while raising the left hand to the same forward position. Once both hands have joined, mimic attaching the weapon then return to a neutral state.
- Beat Both: Bring both hands to mid-upper chest by raising arms to the front. Lower both hands at the same speed about 6 inches, beating an imaginary drum. Reverse the process to return to a neutral state.
- Kick: Flexing the left hip, raise the left leg towards the front. Reverse the process to return to a neutral state.

A.2.3. SKELETON DATA EXAMPLE. The file PALKAdata.mat is a Matlab file which contains the following variables:

- actionGT: This is ground truth information which describes where each action begins and ends. This is given as a cell array where the number of each cell corresponds with the same cell number of masterData, masterListOfActions, and motionGT.
- masterData: This is a cell array where each cell contains a single video in matrix form. The matrix in each cell has 75 rows and N columns (where N is the number of frames in the video).
- masterListOfActions: This is a cell array where each cell contains a matrix with 1 row and 3 columns. The entries of the matrix are integer values between 1 and 12 which correspond to actions in the variable stringActionList.

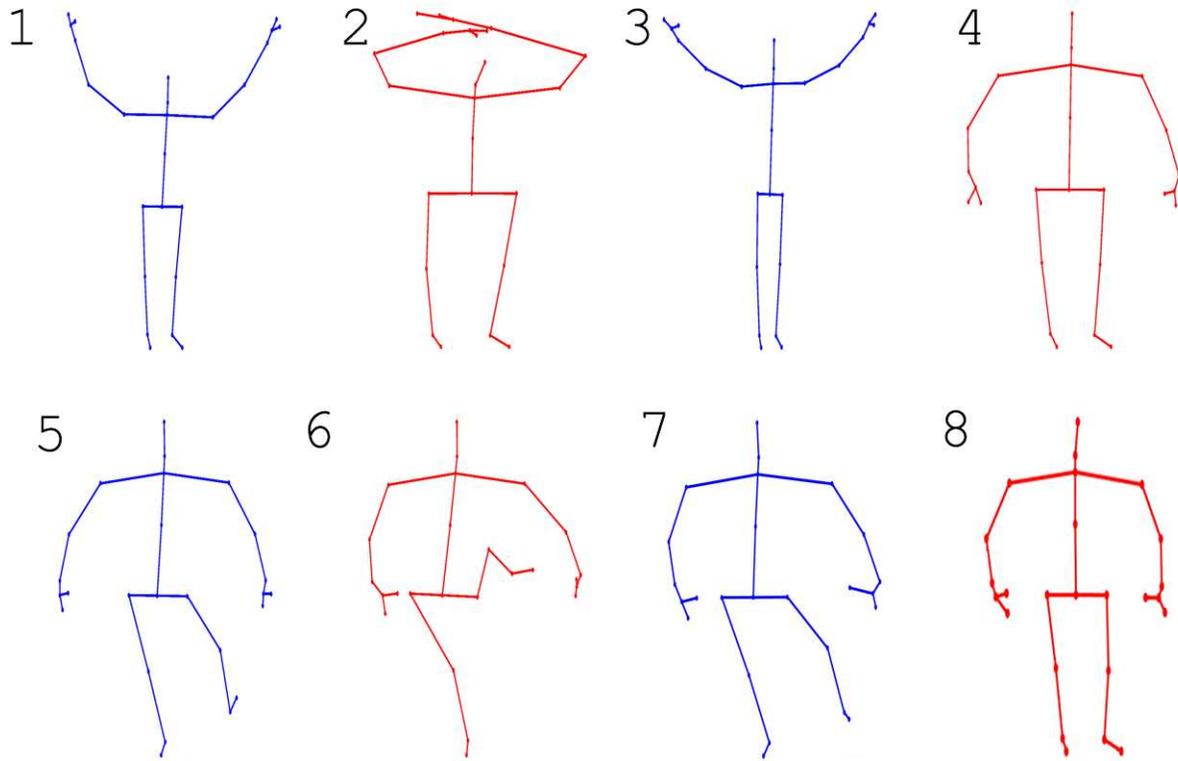


FIGURE A.2. Action 1,2,3,4 - Lift Outstretched Arms. Action 5,6,7,8 - Action: Kick. Motion 1 - Raising Arms. Motion 2 - Transition between Raising Arms and Lowering Arms. Motion 3 - Lowering Arms. Motion 4 - Transition between Lift Outstretched Arms and Kick. Motion 5 - Raising Left Leg. Motion 6 - Transition between Raising Left Leg and Lowering Left Leg. Motion 7 - Lowering Left Leg. Motion 8 - Transition between Kick and next action.

- `motionGT`: This is a cell array. In each cell is a $[2 \ 1]$ cell array. `motionGT{\#}\{1\}` contains a $[1 \ M]$ matrix. Each entry in the matrix defines the beginning frame of a motion. `motionGT{\#}\{2\}` contains a $[1 \ M]$ matrix. Each entry in the matrix defines the beginning frame of a transition.
- `stringActionList`: This is a cell array with 12 rows and 1 column. In each cell is a string which labels action performed.

```

>> load("filename.mat")

>> masterListOfActions{64}
ans =
2 12 2

>> stringActionList{2}
ans =
Duck

>> stringActionList{12}
ans =
Kick

>> size(masterData{64})
ans =
75 173

>> actionGT{64}
ans =
2 44 98 149

>> motionGT{64}
ans =
[1x8 double]
[1x8 double]

>> motionGT{64}{1}
ans =
1 29 46 65 76 83 101 125

>> motionGT{64}{2}
ans =
24 44 58 70 80 98 119 147

```

FIGURE A.3. Example Matlab code to show how to access variables in the provided data files.

The sample code in Figure A.2.3 gives an example of how to examine the content of the 64th video. After loading the data, by calling `masterListOfActions`, we see the 64th video contains actions 2, 11, and 2.

Calling `stringActionList` tells us that action number 2 is Duck and action number 12 is Kick. Hence, in video 64, the actor performs Duck, followed by Kick, followed by Duck.

Looking at `size(masterData{64})`, we get there are 173 frames in this video, and the video (like all the videos in this data set) have 75 dimensions.

By calling `actionGT{64}`, we see where each action begins and ends. For example, the first action begins on frame number 2 and ends on frame 44. The second action begins on frame number 44 and ends on frame 98. The final action begins on frame number 98 and ends on frame 149.

The 64th cell in `motionGT` contains two cells (all videos contain two cells). The variable `motionGT{64}{1}` contains the frame numbers of each beginning motion. The variable `motionGT{64}{2}` contains the frame numbers of each beginning transition. For example, frame 83 starts the beginning of a motion while frame 98 starts the beginning of a transition. The length of `motionGT{X}{1}` is always the same length as `motionGT{X}{2}`.

APPENDIX B

CODE

B.1. SVD-BASED GENERALIZED CURVATURE USING ON THE CURVE SUBSPACES WITH A FIXED WINDOW SIZE

The code presented here will compute generalized curvature values using the method described in Chapter 3.

```
function [generalizedCurvature] = curvaturesFixedOn(points, ...
                                                    eballPercentage)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:
% curvaturesFixedOn.m
%
% INPUT:
% points - a matrix where each column is a point
%
% OUTPUT:
% generalizedCurvature - a matrix where each row is a particular gc
% value (i.e. curvature, torsion, etc.). Each column corresponds to
% the center of the window based on the input points.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

%close all

curveOriginal = points;

sizeCurveOriginal = size(curveOriginal);

dimensionOfData = sizeCurveOriginal(1,1);

%Init the singular value matrix

eigenvalues = zeros(sizeCurveOriginal(1,1), sizeCurveOriginal(1,2));

%Compute the total arc length of the curve

distanceBetweenPoints = zeros(1, sizeCurveOriginal(1,2));

for i = 1:sizeCurveOriginal(1,2)-1
    distanceBetweenPoints(1,i) = norm(curveOriginal(:,i) - ...
                                     curveOriginal(:,i+1));
end

totalArcLength = sum(distanceBetweenPoints);

%Determine desiredDiameter

desiredDiameter = totalArcLength*eballPercentage;

for i = 1:sizeCurveOriginal(1,2)

    beginningFrame = i;

    lastFrame = i;

    stillLookBefore = 1;

```

```

stillLookAfter = 1;

beforeDistance = 0;
afterDistance = 0;

beforeCounter = 0;
while(stillLookBefore == 1)
    if(i-beforeCounter < 1)
        beginningFrame = 1;
        break
    end

    newBeforeDistance = beforeDistance + ...
                        distanceBetweenPoints(1,i-beforeCounter);

    if(newBeforeDistance >= desiredDiameter/2)
        stillLookBefore = 0;
        beginningFrame = i-beforeCounter;
    else
        beforeDistance = newBeforeDistance;
    end

    beforeCounter = beforeCounter + 1;
end

afterCounter = 0;
while(stillLookAfter == 1)
    if(i+afterCounter > sizeCurveOriginal(1,2))
        lastFrame = sizeCurveOriginal(1,2);
        break
    end
end

```

```

end

newAfterDistance = afterDistance + ...
                    distanceBetweenPoints(1,i+afterCounter);

if(newAfterDistance >= desiredDiameter/2)

    stillLookAfter = 0;

    lastFrame = i+afterCounter;

else

    afterDistance = newAfterDistance;

end

afterCounter = afterCounter + 1;

end

windowData = curveOriginal(:, beginningFrame:lastFrame);

sizeWindowData = size(windowData);

val = ceil((sizeWindowData(1,2)-1)/2);

if(val == 0)

    val = 1;

end

%Use the point on the curve as the "mean" data point

meanData = windowData(:,val);

for j = 1:sizeWindowData(1,2)

    windowData(:,j) = windowData(:,j) - meanData;

end

```

```

windowData = windowData';
windowData = windowData'*windowData/size(windowData,1);

%Compute the eigenvalues
evs = eig(windowData);
eigenvalues(:,i) = flipud(evs);
end

%If on the curve
constantList = ones(1, dimensionOfData);

for i = 1:dimensionOfData
    constantList(i) = ((i+1)/(i+1-(-1)^i) ) * sqrt( (4*(i+1)^2 - 1) / 3);
end

generalizedCurvature = zeros(sizeCurveOriginal(1,1)-1, ...
                               sizeCurveOriginal(1,2));

for i = 1:sizeCurveOriginal(1,2)
    for j = 1:sizeCurveOriginal(1,1)-1
        generalizedCurvature(j,i) = constantList(j)*...
            sqrt(eigenvalues(j+1,i)/...
                (eigenvalues(1,i)*eigenvalues(j,i)));
    end
end
end

```

B.2. SVD-BASED GENERALIZED CURVATURE USING OFF THE CURVE SUBSPACES WITH ADAPTIVE WINDOW SIZES

The code presented here will compute generalized curvature values using the methods described in Chapter 4.

```
function [generalizedCurvature, fullEigenvectors]
    = curvaturesAdaptiveOff(data, lastCurvatureDimension)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:
% curvaturesAdaptiveOff.m
%
% INPUT:
% data - [m n] matrix where m is the dimension of the data and n is
% the number of frames of the time-series.
%
% lastCurvatureDimension - integer value, must be less than m. This
% will be the last generalized curvature value computed and the last
% set of eigenvectors returned.
%
% OUTPUT:
% generalizedCurvature - [lastCurvatureDimension n] matrix. Each row
% represents a generalized curvature value (i.e. the first row is
% curvature, the second is torsion, etc.)
%
% fullEigenvectors - [lastCurvatureDimension 1] cell array. The first
```

```

% cell contains, for each point along the curve, 1 m-dimensional
% vector, representing the 1st eigenvector of a local svd. The second
% cell contains, for each point along the curve, 2 m-dimensional
% vectors, representing the first 2 eigenvectors of a local svd, etc.
%
% DESCRIPTION:
% Given a video in time-series format, this will compute all
% generalized curvature values up to lastCurvatureDimension.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define the cutoff value. Initial tests show cutoffValue = 0.005
% produces good results.
cutoffValue = 0.5;

% Compute the dimension of the data
sizeVideo = size(data);
dataDimension = sizeVideo(1,1);
dataFrames = sizeVideo(1,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Edit this area with constants %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

constantList = zeros(1, lastCurvatureDimension);

```

```

for i = 1:lastCurvatureDimension
    constantList(i) = sqrt(((2*(i-1) + 3)*(2*(i-1) + 5) / 3));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Edit this area with constants %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Init the generalizedCurvature and fullEigenvectors, and eigenvalues
% structures
generalizedCurvature = zeros(lastCurvatureDimension, dataFrames);
eigenvalues = zeros(dataDimension, dataFrames);
fullEigenvectors = cell(lastCurvatureDimension,1);

% We are only going to attempt the generalized curvature computations on
% frames that have enough points on each side to allow us to compute the
% necessary eigenvectors (i.e. to compute curvature, we need the tangent
% vector, this requires at least 3 points, so the computation will start
% on frame number 2).

% This for loop works over each frame of the original data.
for i = ceil(lastCurvatureDimension/2)+1:dataFrames- ...
                                ceil(lastCurvatureDimension/2)

```

```

% This for loop works over each generalized curvature dimension we
% care about.
for k = 1:lastCurvatureDimension

    %Create a small window around the ith frame which will be used to
    %compute the smallest amount of data to get reliable singular
    %vectors.
    tDistance = ceil(k/2);
    tWindow = data(:,i-tDistance:i+tDistance);

    %Subtract the center point (so the ith frame is on the origin).
    meanT = mean(tWindow,2);
    sizeTwindow = size(tWindow);
    for j = 1:sizeTwindow(1,2)
        tWindow(:,j) = tWindow(:,j) - meanT;
    end

    %Take the SVD of the windowed data and store the singular vectors
    [U,~,~] = svd(tWindow);

    %Create an epsilon ball around frame i. Grow the ball
    %asymmetrically with respect to the number of points (so it is
    %not really a ball) until the ratio between the B and A side of a
    %right triangle first gets larger than the cutoff value.

    %stillLoop will control the while loop.

```

```

stillLoop = 1;

%radiusCounter will be used to modify the size of the "ball"
%within the while loop
radiusCounter = 1;

%Init the variables useableLeft and useableRight which stores the
%final solution for this particular frame (i) and generalized
%curvature value (k)
useableLeft = [];
useableRight = [];

while(stillLoop == 1)

    %Define the left and right most frames to test around point
    %(i).
    firstFrame = i-radiusCounter;
    lastFrame = i+radiusCounter;

    %If the declared frames to test are outside of the range of
    %the data, reset the values to either the first frame or
    %last frame
    if(firstFrame < 1)
        firstFrame = 1;
    end
    if(lastFrame > dataFrames)
        lastFrame = dataFrames;

```

```

end

%Create a small window (containing only 2 points - the left
%and right most points as defined by firstFrame and
%lastFrame). Then, in order to compare these points with the
%singular vectors in U, subtract the (i)th point from this
%small window of 2 points.
testingWindow = data(:,firstFrame) - data(:,i);
testingWindow = cat(2,testingWindow, data(:,lastFrame) ...
                    - data(:,i));

%Given a right triangle, A and B are the sides of the
%triangle making a right angle and C is the hypotenus of the
%right triangle.

%Compute side B of a right triangle
leftB = distanceBetweenPointAndSubspace(...
                    testingWindow(:,1), U(:,1:k));
rightB =distanceBetweenPointAndSubspace(...
                    testingWindow(:,2), U(:,1:k));

%Compute side C of a right triangle
leftC = norm(testingWindow(:,1) );
rightC = norm(testingWindow(:,2) );

%Compute side A of a right triangle
leftA = sqrt(abs(leftB^2 -leftC^2));

```

```

rightA =sqrt(abs(rightB^2-rightC^2));

%Ratio between sides B and A of a right triangle
leftSide = leftB/leftA;
rightSide = rightB/rightA;

%Conditions to check
if(leftSide > cutoffValue && isempty(useableLeft) == 1 )
    useableLeft = firstFrame;
end
if(rightSide > cutoffValue && isempty(useableRight) == 1)
    useableRight = lastFrame;
end

if(firstFrame == 1)
    useableLeft = firstFrame;
end

if(lastFrame == dataFrames)
    useableRight = lastFrame;
end

if(isempty(useableRight) == 0 && isempty(useableLeft) == 0)
    stillLoop = 0;
end

radiusCounter = radiusCounter + 1;

```

```

end

fprintf('Frame %d. Number Frames %d \n', i, useableRight -
useableLeft) Now that we have the left and right most points
around (i) to use to compute the generalized curvature values,
lets compute them.

%Form a window from useableLeft to useableRight, either subtract
point (i) or mean subtract from every point in the window.
window = data(:,useableLeft:useableRight);
meanWindow = mean(window,2);
sizeWindow = size(window);
for j = 1:sizeWindow(1,2)
    window(:,j) = window(:,j) - meanWindow;
end

%Make the matrix square a normalize so we can compare singular
values with eigenvalues.
window = window';
window = window'*window/size(window,1);

%Compute the eigenvalues
[evs] = eig(window);
eigenvalues(:,i) = flipud(evs);

%Store the eigenvectors from the matrix U (at the beginning of

```

```

%the for loop - note this piece of code could have come much
%sooner).
fullEigenvectors{k} = cat(3,fullEigenvectors{k}, U(:,1:k+1));

%Using the eigenvalues and the constants, compute the generalized
%curvature value.
generalizedCurvature(k,i) = constantList(k) * ...
    sqrt(eigenvalues(k+1,i) / (eigenvalues(1,i) * ...
    eigenvalues(k,i)) );

end

end

```

B.3. NUMERICAL DERIVATIVE BASED CURVATURE

```

function [gc] = numericalCurvature(curve, lastCurvatureDimension)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:
% numericalCurvature.m
%
% INPUT:
% curve - [m n] matrix where each columns is a point in m-dimensional
% space
%

```

```

% lastCurvatureDimension - the last desired generalized curvature value
%
% Computes the generalized curvature values based on numerical
% derivatives, a qr factorization, and the standard formula.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sizeCurve = size(curve);

nd = zeros(sizeCurve(1,1), sizeCurve(1,2), lastCurvatureDimension+2);
nd(:,:,1) = curve;
for j = 2:lastCurvatureDimension+1
    for i = 2:sizeCurve(1,2)-1
        nd(:,i,j) = nd(:,i,j-1) - nd(:,i+1,j-1);
    end
end

nd(:,:,1) = [];

evs = zeros(sizeCurve(1,1), sizeCurve(1,2), lastCurvatureDimension+1);
for i = 1:sizeCurve(1,2)
    temp(:, :) = nd(:,i,:);
    [Q,~] = qr(temp);
    evs(:,i,:) = Q(:,1:lastCurvatureDimension+1);
end

gc = zeros(lastCurvatureDimension, sizeCurve(1,2));

```

```

for j = 1:lastCurvatureDimension
    for i = 1:sizeCurve(1,2)-1
        gc(j,i) = abs(dot(evs(:,i,j)-evs(:,i+1,j),evs(:,i,j+1)))...
            /norm(nd(:,i,1)));
    end
end
end

```

B.4. DYNAMIC TIME WARPING

```

function [distance] = dtw(signal1,signal2>window)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   FUNCTION:
%   dtw.m
%
%   INPUT:
%   signal1 - size is ns1*k. Each row for time. Each column is for channel
%
%   signal2 - size is ns2*k. Each row for time. Each column is for channel
%
%   window - searches for best match within this window size. If this
%   parameter is not given, it is set to Inf
%
%   OUTPUT:
%   distance - distance (based on l2-norm) using the warped path between

```

```

% signals
%
% Creates the best non-linear warping path between the two signals such
% that the l2-distance between paired frames are minimized
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin<3
    window=Inf;
end

ns1=size(signal1,1);
ns2=size(signal2,1);
if size(signal1,2)~=size(signal2,2)
    error('Error in dtw(): dimensions of the two signals do not match.');
```

```
end

window=max(window, abs(ns1-ns2)); % adapt window size

%% initialization
D=zeros(ns1+1,ns2+1)+Inf; % cache matrix
D(1,1)=0;

%% begin dynamic programming
for i=1:ns1
    for j=max(i-window,1):min(i+window,ns2)
        oost=norm(signal1(i,:)-signal2(j,:));
        D(i+1,j+1)=oost+min( [D(i,j+1), D(i+1,j), D(i,j)] );
```

```
    end
end
distance=D (ns1+1,ns2+1);
```