DISSERTATION

ALGORITHMS AND GEOMETRIC ANALYSIS OF DATA SETS THAT ARE INVARIANT UNDER A GROUP ACTION

Submitted by

Elin Rose Smith

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2010

COLORADO STATE UNIVERSITY

July 27, 2010

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY ELIN ROSE SMITH ENTITLED ALGORITHMS AND GEOMETRIC ANALYSIS OF DATA SETS THAT ARE INVARIANT UNDER A GROUP ACTION BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Dan Bates

Michael Kirby

Ross McConnell

Advisor: Chris Peterson

Department Head: Simon Tavener

ABSTRACT OF DISSERTATION

ALGORITHMS AND GEOMETRIC ANALYSIS OF DATA SETS THAT ARE INVARIANT UNDER A GROUP ACTION

We apply and develop pattern analysis techniques in the setting of data sets that are invariant under a group action. We apply Principal Component Analysis to data sets of images of a rotating object in Chapter 5 as a means of obtaining visual and low-dimensional representations of data. In Chapter 6, we propose an algorithm for finding distributions of points in a base space that are (locally) optimal in the sense that subspaces in the associated data bundle are distributed with locally maximal distance between neighbors. In Chapter 7, we define a distortion function that measures the quality of an approximation of a vector bundle by a set of points. We then use this function to compare the behavior of four standard distance metrics and one non-metric. Finally, in Chapter 8, we develop an algorithm to find the approximate intersection of two data sets.

Elin Rose Smith
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Fall 2010

ACKNOWLEDGEMENTS

"We are so often caught up in our destination that we forget to appreciate the journey, especially the goodness of the people we meet on the way." – Author Unknown

There are several people who deserve thanks for making significant contributions to my mathematical career and to the work presented in this paper. I would first like to thank my advisor, Chris Peterson, for his support, knowledge, and unending enthusiasm for all things.

I am grateful to those who have been a part of my committee, Chris Peterson, Michael Kirby, Ross McConnell, Dan Bates, and Jeff Achter, for their excellent questions, comments, and insights.

The Pattern Analysis Lab at Colorado State University provided the technical resources for this work. In addition, I appreciate the supportive group of researchers and students in the lab with whom I have had the pleasure of working.

I have recently had the good fortune to collaborate with Louis Scharf and Tony Maciejewski, of the Electrical Engineering Department. I would like to take this opportunity to thank them for many enjoyable meetings and great ideas.

I am indebted to the students, faculty, and staff of Colorado State University. Looking back on six years of hard work, I remember not the hours but the study sessions with friends, the instruction and friendship of my teachers, and the students in my classes through whom I discovered my love of teaching.

And last, but certainly not least, I am thankful to my family and friends for their laughter, support, and love.

Contents

Chapter 1

Introduction

The focus of this paper is in the application and development of pattern analysis techniques in the setting of data sets that are invariant under a group action. Pattern analysis techniques have immediate applications in a variety of fields, including face and object recognition, data reduction, medicine, genetics, and astronomy [9, 11, 29, 33, 41, 50], to name just a few. In this paper, we focus, in particular, on data sets in the context of image analysis. Data sets of images lend themselves to visual analysis and hence can provide intuition for developing methods of analysis.

We use techniques from a range of fields, including geometric data analysis, linear and multilinear algebra, algebraic geometry, differential geometry, Fourier analysis, and signal processing [2, 13, 21, 24, 26, 27, 28, 31, 37, 39, 44, 45, 47, 49]. The analysis of group actions in the setting of image recognition and classification has been fruitful yet there remain a number of outstanding open problems [5, 10, 7, 11, 22, 31, 35, 43, 52].

We begin the paper with a discussion of some standard pattern analysis techniques and background information in Chapter 2. Then in Chapter 3, we give examples of data sets that are invariant under a group action as the context for the data sets that we study in later chapters.

We establish in Chapter 4 that in data collection, noise is introduced by the

overhead lights as well as by the camera itself. We present a method for removing the noise from the lights along with the results of its application to a data set of images.

The main objects of study in this paper are data sets of images of a rotating object. We apply Principal Component Analysis to such data sets in Chapter 5 as a means of obtaining visual and low-dimensional representations of the data.

In Chapter 6, we consider the following problem. Suppose that a fixed number of images, say n, are to be collected from locations along a circle around an object. We develop an algorithm to find distributions of the n camera locations which are (locally) optimal in the sense that subspace representations of the object at the n locations are distributed with maximal distance between neighbors. We refer to this algorithm as the Nearest Neighbor Dispersion Algorithm.

In Chapter 7, we define a function that measures the success of an application of the Nearest Neighbor Dispersion Algorithm. We use this function to compare the behavior of some standard distance metrics and one non-metric.

Finally, in Chapter 8, we develop an algorithm to find the approximate intersection of two data sets. The context for this problem is this. Suppose that two sets of images are collected along two distinct great circles on a sphere surrounding an object. The underlying geometry suggests that there exist two pairs of images from the two data sets that correspond to the antipodal points of intersection of the great circles. These pairs of images should be similar, up to rotation. The success of this algorithm is measured visually by the output images from the algorithm.

Chapter 2

Background

2.1 Introduction

A classical problem in pattern analysis is concerned with classification – given a training set of data, can we separate the set into meaningful classes and correctly classify novel data? This question occurs frequently in a variety of fields, including face recognition, medicine, genetics, and astronomy [9, 11, 29, 33, 41, 50], to name just a few. One of the main goals of the work presented in this paper is to find an optimal representation of the space of illumination spaces associated to various perspectives of an object that may then be used successfully in classification problems. The techniques we use come from a variety of fields, including geometric data analysis, linear and multilinear algebra, algebraic geometry, differential geometry, basic Lie theory, Fourier analysis, and signal processing [2, 13, 20, 21, 26, 24, 27, 28, 31, 37, 39, 44, 45, 47, 49]. One technique that is frequently employed in data analysis and which we apply in our work is Principal Component Analysis. In Section 2.2, we summarize the main theory for this technique, using the notation and derivation provided in [31]. In Section 2.3, we develop the basic theory and properties pertaining to the singular value decomposition, which we apply to a variety of data sets. We give a brief introduction to vector bundles in Section 2.4. Then in Section 2.5, we review

the Grassmann variety and principal angles. We use principal angles as a measure of distance between illumination spaces. Distance functions and unitarily invariant functions are discussed in Sections 2.6 and 2.7. In Section 2.8, we state the definition of the illumination space of an object along with some of its properties. In Section 2.9, we introduce the Koszul complex, a chain complex with which we construct a trial data set. The main content of this paper is in development of techniques for analysis of data sets that are invariant under a particular group action. We therefore close the chapter with Section 2.10, in which we give a brief review of group actions.

2.2 Principal Component Analysis

Principal Component Analysis is a technique used frequently in classification problems, missing data reconstruction, and dimensionality reduction; see for example [9, 29, 30, 31]. Principal Component Analysis (PCA) is known under several names, including the Karhunen-Loève (KL) expansion, proper orthogonal decomposition (POD), and empirical orthogonal functions (EOFs). To begin, consider a data set represented as a matrix X, where each column represents a single sample. Samples often represent variation in time, pose, illumination, or camera location. The goal of PCA is to find the optimal ordered unitary basis for X in the sense that given any truncation of the basis, the mean squared error over all unitary bases is minimized.

Throughout this discussion, we assume that the data has been mean-subtracted. Given a data matrix X composed of column vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}$, we define the mean of the vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}$ to be

$$\langle \mathbf{x} \rangle := \frac{1}{p} \sum_{\mu=1}^{p} \mathbf{x}^{(\mu)}.$$

We mean-subtract X by subtracting $\langle \mathbf{x} \rangle$ from each $\mathbf{x}^{(\mu)}$ of X. The geometric effect of mean-subtraction is to move the centroid of the data set to the zero of the coordinate system.

Let us start by stating more explicitly what we mean by an optimal basis. Suppose our data set X is $n-$dimensional. Let $\mathcal{B} = \left\{\phi^{(1)}, \ldots, \phi^{(n)}\right\}$ be an ordered unitary basis for X. Any point $\mathbf{x} \in X$ can be expressed in terms of this basis: $\mathbf{x} = \alpha_1\phi^{(1)} + \cdots + \alpha_n\phi^{(n)}$. The $d-$term truncation of this expansion for $\mathbf{x}$ is given by $\mathbf{x}_d = \alpha_1\phi^{(1)} + \cdots + \alpha_d\phi^{(d)}$. The mean squared truncation error is given by $\epsilon = \left\langle \|\mathbf{x} - \mathbf{x}_d\|^2 \right\rangle$, where the norm denotes the Euclidean norm, and $\langle * \rangle$ is the usual average. We define $\mathcal{B}$ to be an optimal basis if, among all unitary bases, $\mathcal{B}$ minimizes the mean squared truncation error, ϵ.

One method of constructing such an optimal basis $\mathcal{B}$ from a data set X is given by Principal Component Analysis. Before we state the basis of the algorithm, we make one observation. A basis $\mathcal{B}$ which minimizes the mean squared truncation error simultaneously maximizes the mean squared projection of the data onto itself. This property can be seen as follows. The mean squared truncation error is

$$\begin{aligned}
\epsilon &= \left\langle \|\mathbf{x} - \mathbf{x}_d\|^2 \right\rangle \\
&= \left\langle \left(\sum_{j=d+1}^{n} \alpha_j \phi^{(j)}, \sum_{k=d+1}^{n} \alpha_k \phi^{(k)} \right) \right\rangle \\
&= \left\langle \sum_{j,k=d+1}^{n} \alpha_j \alpha_k \left(\phi^{(j)}, \phi^{(k)} \right) \right\rangle \\
&= \left\langle \sum_{j=d+1}^{n} \alpha_j^2 \right\rangle, \text{due to the orthogonality of the } \left\{\phi^{(j)}\right\}_{j=1}^{n}, \\
&= \left\langle \sum_{j=d+1}^{n} \left(\mathbf{x}, \phi^{(j)} \right)^2 \right\rangle.
\end{aligned}$$

Hence, to construct a basis $\mathcal{B}$, which minimizes mean squared truncation error, we need only construct a basis that maximizes the first d squared projections of the data onto the basis. That is, we wish to construct $\mathcal{B}$ so that it maximizes

$$\left\langle \sum_{j=1}^{d} \left(\mathbf{x}, \phi^{(j)} \right)^2 \right\rangle.$$

Given this framework, we may now state the steps of the Principal Component Analysis construction of an optimal basis.

- STEP 1 Find W_1, the best one-dimensional subspace.
- STEP 2 Find W_2, the best one-dimensional subspace orthogonal to W_1.
- STEP i Find W_i, the best one-dimensional subspace such that $W_i \perp W_j$ for all $j < i$,

where 'best' is in the sense that the subspace maximizes the mean squared projection of all columns of X onto itself.

These steps may be carried out to construct $\mathcal{B}$ using Lagrange multipliers. This method of construction is known as the *direct method.* An alternative method for computing $\mathcal{B}$ is given via the singular value decomposition of the matrix X. The left-singular vectors of X are the columns of $\mathcal{B}$, and the singular values are the square roots of the eigenvalues of the matrix XX^H, where H denotes the Hermitian transpose. For details on the equivalence of these two methods, see [31]. Throughout the work described in this paper, we use the singular value decomposition consistently as the method to determine the best $d-$dimensional representation of the data. We therefore use Section 2.3 to state the singular value decomposition explicitly.

One final important fact about the optimal basis $\mathcal{B}$ determined by Principal Component Analysis is that any truncation of $\mathcal{B}$ captures more statistical variance than any other basis of the same dimension. Let $\left\{\psi^{(i)}\right\}_{i=1}^{n}$ be a different basis for X. Given an element $\mathbf{x} \in X$, let the $d-$term truncation of the expansion of $\mathbf{x}$ be written as $\mathbf{x}_d = \sum_{j=1}^{d} \beta_j \psi^{(j)}$. We define a measure of the variance of X with respect to the basis given by $\left\{\psi^{(i)}\right\}_{i=1}^{n}$ to be

$$\rho_j = \left\langle \beta_j^2 \right\rangle .$$

Suppose also that the $d-$term truncation of $\mathbf{x}$ with respect to the basis $\mathcal{B}$ is $\mathbf{x}_d = \alpha_1 \phi^{(1)} + \cdots + \alpha_d \phi^{(d)}$. It can be shown that

$$\sum_{j=1}^{d} \rho_j \leq \sum_{j=1}^{d} \left\langle \alpha_j^2 \right\rangle .$$

Equality occurs when the basis $\left\{\psi^{(i)}\right\}_{i=1}^{n}$ is the basis given by PCA. Hence, PCA constructs a basis capturing more variance than any other basis.

2.3 The Singular Value Decomposition

We begin by stating the existence of the decomposition.

Theorem 2.1. *(Singular Value Decomposition)*
Let $X \in k^{m\times n}$, with $k = \mathbb{R}$ or $\mathbb{C}$, and let $\ell = min\{m, n\}$. Then there exist unitary matrices U and V defined over k and a diagonal matrix Σ such that

$$X = U\Sigma V^H,$$

where U has size $m \times m$, V has size $n \times n$, and Σ has size $m \times n$, and where H denotes the Hermitian transpose.

For both $k = \mathbb{R}$ and $k = \mathbb{C}$, the entries of $\Sigma = \text{diag}\left(\sigma^{(1)}, \ldots, \sigma^{(\ell)}\right)$ are nonnegative real numbers and are conventionally ordered by $\sigma^{(1)} \geq \cdots \geq \sigma^{(\ell)} \geq 0$. We refer to the values $\sigma^{(1)}, \ldots, \sigma^{(\ell)}$ as the *singular values of X*. Given a choice of ordering of the singular values, the matrix Σ is uniquely determined by X. If $m > n$, then Σ is of the form

$$\begin{pmatrix} \sigma^{(1)} & & 0 \\ & \ddots & \\ 0 & & \sigma^{(n)} \\ 0 & & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix},$$

and if $m < n$, then Σ is of the form

$$\begin{pmatrix} \sigma^{(1)} & & 0 & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ 0 & & \sigma^{(m)} & 0 & \cdots & 0 \end{pmatrix}.$$

A constructive proof of the singular value decomposition (SVD) is given in [31]. We now state a few important properties of the SVD that will be useful in our work.

We first note the relationship between the number of nonzero singular values and the rank of X. Since Σ is a diagonal matrix, the rank of Σ is equal to the number of nonzero singular values. Also, orthogonal transformations leave the rank unchanged. Therefore, since $X = U\Sigma V^H$, we must have that $\text{rank}(X)$ is equal to the number of nonzero singular values.

Next, let $\text{rank}(X) = r$ and define $\Sigma_i = \text{diag}(0, \ldots, 0, \sigma^{(i)}, 0, \ldots, 0)$. Then

$$\begin{aligned} X &= U\Sigma V^H \\ &= U(\Sigma_1 + \cdots + \Sigma_r)V^H \\ &= U\Sigma_1 V^H + \cdots + U\Sigma_r V^H \\ &= \sigma^{(1)}\mathbf{u}^{(1)}\mathbf{v}^{(1)H} + \cdots + \sigma^{(r)}\mathbf{u}^{(r)}\mathbf{v}^{(r)H} \\ &= \sum_{j=1}^{r} \sigma^{(j)}\mathbf{u}^{(j)}\mathbf{v}^{(j)H}. \end{aligned}$$

It follows that column i of X is given by

$$\mathbf{x}^{(i)} = \sum_{j=1}^{r} \sigma^{(j)}\mathbf{v}_i^{(j)}\mathbf{u}^{(j)}.$$

And hence, *the set of the first r left singular vectors form a basis for the column space of the matrix X*. This property implies that we may apply the SVD to find a change of basis for X.

We also note that the left singular vectors are frequently referred to as eigenvectors because they are in fact eigenvectors of the matrix XX^H. To see this, observe that $X = U\Sigma V^H$ and U, V orthogonal implies $X^H U = V\Sigma^H$. Although the property that

the left singular vectors are eigenvectors of XX^H holds in general, we will discuss only the case when $m \geq n$ and X has full rank. Then for each $i = 1, \ldots, n$, we have

$$\begin{aligned} X^H \mathbf{u}^{(i)} &= \sigma^{(i)} \mathbf{v}^{(i)} \\ XX^H \mathbf{u}^{(i)} &= \sigma^{(i)} X \mathbf{v}^{(i)} \\ XX^H \mathbf{u}^{(i)} &= \left(\sigma^{(i)}\right)^2 \mathbf{u}^{(i)}, \end{aligned}$$

where we have used the fact that $XV = U\Sigma$. Therefore the left singular vectors of X are eigenvectors of XX^H with eigenvalues equal to the singular values squared.

In practice, we frequently compute what is referred to as the *thin SVD*, as opposed to the SVD described above. The thin SVD enjoys many of the properties of the SVD, but is less computationally expensive. The thin SVD is used when $m > n$ and is given by $X = \hat{U}\hat{\Sigma}V^H$, where $\hat{U}$ is the matrix U without the last $m - n$ columns, and Σ is an $n \times n$ diagonal matrix defined by $\text{diag}\left(\sigma^{(1)}, \ldots, \sigma^{(n)}\right)$. The computation of the thin SVD is significantly faster than the standard SVD if $m >> n$. The matrix $\hat{U}$ is not orthogonal in general, but since we often only require the computation of the first few left singular vectors, this is acceptable.

2.4 Vector Bundles

Many data sets have a natural structure as a vector bundle, as is the case with several of the data sets discussed in this paper. We therefore give a brief review of vector bundles here. For more details on this topic, see [45] or [46]. Recall that a vector bundle parametrizes a family of vector spaces E via a space X. We refer to a vector bundle as real if the vector spaces are defined over $\mathbb{R}$ and as complex if the vector spaces are defined over $\mathbb{C}$. Let k be a field equal to either $\mathbb{R}$ or $\mathbb{C}$. We formally define a topological vector bundle as follows.

Definition 2.2. *A (topological) vector bundle E of rank r over X is a (topological) base space X and a (topological) total space E, together with a projection $\pi : E \to X$,*

satisfying the following two conditions:

- *There exists an open cover* $\bigcup U_i$ *of* X *where each* $\pi^{-1}(U_i)$ *is isomorphic to* $U_i \times k^r$ *via a fiber-preserving isomorphism* φ_i*. That is, if* p *is the natural projection onto* U_i*, then the following diagram commutes for each* i*.*

$$\begin{array}{ccc} \pi^{-1}(U_i) & \xrightarrow{\varphi_i} & U_i \times k^r \\ & {\scriptstyle \pi} \searrow \quad \swarrow {\scriptstyle p} & \\ & U_i & \end{array}$$

- *The maps* φ_i *are linearly compatible. By this, we mean that on* $U_i \cap U_j$*, the following composition is a linear map of* k^r *for every fixed* x *:*

$$\begin{aligned} \varphi_j \circ \varphi_i^{-1} : (U_i \cap U_j) \times k^r &\rightarrow (U_i \cap U_j) \times k^r \\ (x, v) &\mapsto (x, (\varphi_j \circ \varphi_i^{-1}(v)). \end{aligned}$$

The definition of a vector bundle above is in the category of topological spaces. We can similarly define vector bundles in other categories. For instance, we might be interested in vector bundles in the category of differentiable manifolds or algebraic varieties. In the setting of discrete data sets, we frequently make the assumption that we are sampling from a differentiable vector bundle. For more on vector bundles, see [18, 23, 38, 40].

2.5 The Grassmann Variety and Principal Angles

In order to analyze a given data set, we frequently require some measure of distance between points in the set. In several of our data sets, points in the set are viewed as points in the Grassmann variety. We begin with a general definition of the Grassmannian.

Definition 2.3. *The* Grassmann variety*, or* Grassmannian*, is defined to be the set of all linear subspaces of a vector space* V *of a fixed dimension* k*.*

Throughout this paper, we will work with the real Grassmannian, and will denote by $\mathrm{Gr}(k,n)$ the set of all k-dimensional subspaces of $\mathbb{R}^n$. As a special case, consider $\mathrm{Gr}(1,n)$. This is the set of linear subspaces of $\mathbb{R}^n$, and we therefore have the following equivalence:

$$\mathrm{Gr}(1,n) \cong \mathbb{P}^{n-1}(\mathbb{R}).$$

For more on the Grassmannian, see [23, 25, 45, 46].

Any measure of distance on the Grassmannian must be a unitarily invariant function if we wish to measure distance in a consistent way. Fortunately, much work has been done on unitarily invariant norms; we briefly summarize this work in this section. We will begin with the definition and computation of principal angles and will list some of the most commonly used metrics on the Grassmannian. We will close with a discussion of the connection between unitarily invariant norms and principal angles.

Principal angles are defined recursively. Informally, the process of finding principal angles is as follows. We are given two subspaces X and Y of k^n, with $k = \mathbb{R}$ or $k = \mathbb{C}$. Begin by finding the smallest angle achieved between any pair of vectors, one from X and one from Y. This angle is the first principal angle, θ_1. Suppose that the vectors $x \in X$ and $y \in Y$ have angle θ_1 between them. Then the second principal angle can be found by finding the smallest angle achieved between a pair of vectors in the spaces given by the orthogonal complement of x in X and the orthogonal complement of y in Y. Each successive principal angle is found by repeating this process. There are always t principal angles, where $m = \min\{\dim(X), \dim(Y)\}$. More formally, principal angles are defined as follows.

Definition 2.4. *Let $X, Y \subseteq k^n$, with $k = \mathbb{R}$ or $k = \mathbb{C}$, $\dim(X) = s, \dim(Y) = t$, and $m = \min\{s,t\}$. Then the ordered principal angles $\theta_1, \ldots, \theta_m \in \left[0, \frac{\pi}{2}\right]$ are defined recursively as*

$$\theta_j := \max_{x \in X} \max_{y \in Y} x^H y = x_j^H y_j$$

subject to $\|x\| = \|y\| = 1$, and $x^H x_i = 0, y^H y_i = 0$ for all $1 \leq i \leq j-1$.

For more on principal angles, see [1, 36]. The numerical computation of principal angles is closely related to the computation of the singular value decomposition. Computing principal angles via singular the singular value decomposition has been shown to be a numerically stable method [14, 22]. Bjorck and Golub prove the following theorem in [8], which states the general relationship between principal angles and the singular values of a particular matrix.

Theorem 2.5. *Let X and Y be subspaces of k^n, with $k = \mathbb{R}$ or $k = \mathbb{C}$, and let $\dim(X) = s, \dim(Y) = t$, and $m = \min\{s, t\}$. Let Q_X and Q_Y be unitary bases for X and Y, respectively. If the singular values of $Q_X^H Q_Y$ are $\sigma_1, \ldots, \sigma_m$ then $\sigma_i = \cos\theta_i$ for each $i = 1, \ldots, m$, where θ_i is the i^{th} principal angle.*

Using this theorem along with the following lemma, we can show that the computation of principal angles via singular values is independent of choice of unitary basis.

Lemma 2.6. *Let $X, Y \subseteq k^n$, with $k = \mathbb{R}$ or $k = \mathbb{C}$, $\dim(X) = s, \dim(Y) = t$, and $m = \min\{s, t\}$. Let Q_X and Q_Y be unitary bases for X and Y, respectively. Let the singular values of the matrix $Q_X^H Q_Y$ be $\sigma_1, \ldots, \sigma_m$. If W_X and W_Y are any other unitary bases for X and Y, then the singular values of the matrix $W_X^H W_Y$ are $\sigma_1, \ldots, \sigma_m$.*

Proof. Since Q_X and W_X are both unitary bases for the same space X, there exists a unitary change of basis matrix M_1 such that $W_X = Q_X M_1$. Similarly, there exists a unitary matrix M_2 such that $W_Y = Q_Y M_2$. Let D be the matrix $W_X^H W_Y$, and let the singular value decomposition of $Q_X^H Q_Y$ be $Q_X^H Q_Y = U \Sigma V^H$. Then

$$\begin{aligned} D &= W_X^H W_Y \\ &= (Q_X M_1)^H (Q_Y M_2) \\ &= M_1^H Q_X^H Q_Y M_2 \\ &= M_1^H U \Sigma V^H M_2 \\ &= \left(M_1^H U\right) \Sigma \left(V^H M_2\right). \end{aligned}$$

Hence, the singular values of D are the same as those of $Q_X^H Q_Y$. $\square$

The following corollary will be useful in Section 2.7.

Corollary 2.7. *Let X and Y be two subspaces of k^n, with $k = \mathbb{R}$ or $k = \mathbb{C}$. The computation of the principal angles between X and Y via singular values is independent of the choice of unitary basis for either space.*

2.6 Commonly Used Distance Metrics

We consider four metrics and one non-metric on the Grassmannian. We will briefly discuss the context for each one along with its definition. We then discuss the special case of these metrics in $\mathrm{Gr}(1, n)$, the space of linear subspaces of $n-$dimensional space.

The differential topology on the Grassmannian has several realizations, and it is frequently the case that different realizations lead to different associated distance metrics. Let us begin with the realization of the Grassmannian as a quotient of the orthogonal group. It can be shown that

$$\mathrm{Gr}(k, n) = \frac{O(n)}{O(k) \times O(n-k)}.$$

For more details on this equivalence, see [34]. In [4], Basri and Jacobs show that the standard metric on $O(n)$ descends to a metric on $\mathrm{Gr}(k, n)$. In other words, the standard metric respects cosets when the Grassmannian is viewed as a quotient group. This metric is referred to as the geodesic distance, or arc length, and has been shown to be useful in packings in the Grassmannain [3]. It is defined as follows (for details, see [16]).

Definition 2.8. *Suppose that two subspaces $X, Y \subseteq \mathbb{R}^n$ have principal angles $\theta = (\theta_1, \ldots, \theta_m)$. Then the* geodesic distance metric *is defined to be*

$$d_g(X, Y) = \sqrt{\sum_{i=1}^{m} \theta_i^2} = \|\theta\|_2 .$$

Second, the Grassmannian can be viewed as a submanifold of projective space via the Plücker embedding. That is,

$$\mathrm{Gr}(k,n) = \mathbb{P}^{\binom{n}{k}-1}(\mathbb{R}) \subset \mathbb{P}(\Lambda^k \mathbb{R}^n).$$

Given this embedding, the Grassmannian inherits the Fubini-Study metric on projective space (see [23]).

Definition 2.9. *Suppose that two subspaces $X, Y \subseteq \mathbb{R}^n$ have principal angles $\theta = (\theta_1, \ldots, \theta_m)$. Then the* Fubini-Study distance metric *is given by*

$$d_{FS}(X,Y) = \arccos\left(\Pi_{i=1}^{m} \cos\theta_i\right).$$

Third, using the projection described in [12], the Grassmannian can be viewed as a submanifold of Euclidean space:

$$\mathrm{Gr}(k,n) \subset \mathbb{R}^{(n^2+n-2)/2}.$$

Restricting the usual Euclidean distance metric to $\mathrm{Gr}(k,n)$, we obtain the chordal, or projection F, distance metric. The chordal distance metric has been shown to be useful in packings in the Grassmannian (see [3, 12]).

Definition 2.10. *Suppose that two subspaces $X, Y \subseteq \mathbb{R}^n$ have principal angles $\theta = (\theta_1, \ldots, \theta_m)$. Then the* chordal distance metric *is given by*

$$d_c(X,Y) = \sqrt{\sum_{i=1}^{m} (\sin\theta_i)^2} = \|\sin\theta\|_2 .$$

Finally, we can view points in the Grassmannian as subspaces of $\mathbb{R}^n$ and use the usual subspace distance defined in [21].

Definition 2.11. *Suppose that two subspaces $X, Y \subseteq \mathbb{R}^n$ have principal angles $\theta = (\theta_1, \ldots, \theta_m)$. Then the* subspace distance metric *is given by*

$$d_{ss}(X,Y) = \max_{i=1,\ldots,m} \{\sin\theta_i\} = \|\sin\theta\|_\infty .$$

We also consider one measure of distance, which in most cases, is not a metric. It deserves consideration given that it has been used successfully in other contexts (for example, in the separability of face illumination spaces [10, 7]).

Definition 2.12. *Suppose that two subspaces $X, Y \subseteq \mathbb{R}^n$ have principal angles $\theta = (\theta_1, \ldots, \theta_m)$. Then the* smallest principal angle distance function *is given by*

$$d_{spa}(X, Y) = \min_{i=1,\ldots,m} \{\theta_i\}.$$

In general, this measure of distance is not a metric because if $X, Y \subseteq \mathbb{R}^n$ with $\dim(X) > 1$ and $\dim(Y) > 1$, then it is possible to have $d_{spa}(X, Y) = 0$ even though $X \neq Y$. For example, two planes that intersect in a line would have distance zero even though they are distinct objects.

Note that if we consider one-dimensional subspaces of $\mathbb{R}^n$, then several of these distance measures are the same. Suppose that L and M are linear spaces in $\mathbb{R}^n$. Then $\theta = \theta_1$, and we have

$$\begin{aligned} d_g(L, M) &= \sqrt{\sum_{i=1}^{1} \theta_i^2} \\ &= \sqrt{\theta_1^2} \\ &= |\theta_1| \\ &= \theta_1 \\ &= \min(\{\theta_1\}) \\ &= d_{spa}(L, M), \end{aligned}$$

since principal angles are nonnegative by definition. Also, since each $\theta_i \in \left[0, \frac{\pi}{2}\right]$, we have

$$\begin{aligned} d_{FS}(L, M) &= \arccos\left(\Pi_{i=1}^{1} \cos\theta_i\right) \\ &= \arccos(\cos\theta_1) \\ &= \theta_1 \\ &= d_{spa}(L, M). \end{aligned}$$

Again using the fact that principal angles are in the interval $\left[0, \frac{\pi}{2}\right]$, we have that the chordal distance metric is equal to the subspace metric in this case:

$$\begin{aligned} d_c(L, M) &= \sqrt{\sum_{i=1}^{1} (\sin\theta_i)^2} \\ &= \sqrt{(\sin\theta_1)^2} \\ &= |\sin\theta_1| \\ &= \sin\theta_1 \\ &= \max(\{\sin\theta_1\}) \\ &= d_{ss}(L, M). \end{aligned} \tag{2.13}$$

2.7 Unitarily Invariant Functions

In defining a function on the Grassmannian, we require that it be unitarily invariant. The nature of unitarily invariant functions and principal angles are in fact closely related. We first give two definitions and then we state this connection. For further details, see [48].

Definition 2.14. *A matrix norm $\|\cdot\|$ on $k^{m\times n}$ with $k = \mathbb{R}$ or $k = \mathbb{C}$ is* unitarily invariant *if for any unitary matrices A and B, we have $\left\|A^H M B\right\| = \|M\|$ for any matrix $M \in k^{m\times n}$.*

Definition 2.15. *A vector norm $\Phi : \mathbb{R}^n \to \mathbb{R}$ is a* symmetric gauge function *if Φ is an absolute norm and is permutation invariant. That is, Φ must satisfy each of the following:*

1. *If $x \neq 0$, then $\Phi(x) > 0$.*
2. *$\Phi(\alpha x) = |\alpha|\,\Phi(x)$, for $\alpha \in \mathbb{R}$.*
3. *$\Phi(x + y) \leq \Phi(x) + \Phi(y)$.*
4. *If P is a permutation matrix, then $\Phi(Px) = \Phi(x)$.*

5. $\Phi(|x|) = \Phi(x)$.

Von Neumann showed in [51] that if $\|\cdot\|$ is a unitarily invariant matrix norm, then there exists a corresponding symmetric gauge function Φ that is a function of the singular values of its argument. Part of this connection is easy to see: Suppose that $\|\cdot\|$ is a unitarily invariant norm and that a matrix M has singular value decomposition $U\Sigma V^H$. Then we have

$$\begin{aligned} \|M\| &= \left\|U\Sigma V^H\right\| \\ &= \|\Sigma\| . \end{aligned}$$

Therefore, $\|\cdot\|$ is a function of the singular values of its argument.

In addition to the connection due to Von Neumann, there exists a connection in the other direction: a bi-variate function on the Grassmannian defined in terms of principal angles (and hence defined in terms of the singular values of a specific matrix) is unitarily invariant. We conclude this section with a proof of this fact.

Consider the natural action of the unitary group on the vector space k^n given by multiplication on the left. This induces an action on the subspaces of k^n. In the following theorem, we say that a function is unitarily invariant if its evaluation is unaffected by such an action.

Theorem 2.16. *Let $X, Y \subseteq k^n$, with $k = \mathbb{R}$ or $k = \mathbb{C}$, $\dim(X) = s, \dim(Y) = t$, and $m = \min\{s, t\}$. Let the principal angles between X and Y be $\theta = (\theta_1, \ldots, \theta_m)$. Define f to be the natural map*

$$\begin{aligned} f : Gr(s, n) \times Gr(t, n) &\to \mathbb{R}^m \\ (X, Y) &\mapsto (\theta_1, \ldots, \theta_m) . \end{aligned}$$

Then f is unitarily invariant.

Proof. Let $X, Y \subseteq k^n$. Recall from Theorem 2.5 and Corollary 2.7 that principal angles between spaces can be computed via arccosines of singular values independently

of choice of unitary basis. Let us therefore begin by choosing unitary bases Q_X and Q_Y for the spaces X and Y. Suppose that the singular values of $Q_X^H Q_Y$ are $\sigma_1, \ldots, \sigma_m$. To show that f is unitarily invariant, we need only show that after an action of the unitary group on both X and Y, the singular values of the appropriate matrix are equal to $\sigma_1, \ldots, \sigma_m$. A unitary action on X and Y can be realized by left multiplication of Q_X and Q_Y by a unitary matrix. Let A be a unitary matrix. Then we have

$$\begin{aligned} (AQ_X)^H (AQ_Y) &= Q_X^H A^H A Q_Y \\ &= Q_X^H Q_Y. \end{aligned}$$

It follows that the singular values of $(AQ_X)^H(AQ_Y)$ are equal to the singular values of $Q_X^H Q_Y$, and hence the computations of the principal angles between the two pairs of spaces are equal as well. Therefore, f is unitarily invariant. □

As a consequence of this theorem, all of the distance measures defined in Section 2.6 are unitarily invariant. Note also that because principal angles are defined in terms of singular values, the singular value decomposition gives a numerically stable means of computing the approximate distance between any pair of subspaces. We also have a way of creating new distance measures that are unitarily invariant: we can weight the measures we have discussed already, or we can create new ones as functions of the singular values. The advantage gained by Theorem 2.16 is that to create new unitarily invariant distance functions, we need not look farther than those functions which can be written in terms of the principal angles. For more on unitarily invariant functions, see [32].

2.8 The Illumination Space of an Object

In this section, we give a brief review of the illumination space of an object. The fact that an object can appear dramatically different under varying lighting conditions suggests two potential approaches to object recognition algorithms. One is to seek

an illumination invariant image (see for example [42]). The second approach is to represent an object by its appearance under a variety of lighting conditions. Recent papers have shown that representing the various illuminations of an object can be extremely useful in object recognition algorithms (see for example [10, 7]). A representation of an object that contains information about the appearance of the object under all possible lighting conditions is the illumination space.

If we consider all possible illuminations of an object from all light source locations, then the result is an infinite dimensional space. Such an space is difficult to compute and to use in object recognition algorithms. However, several papers have demonstrated that the illumination space of an object is in fact close to being flat. That is, the illumination space is close to lying on a low-dimensional linear space [4, 6, 19].

The structure of the illumination space is known as well. In [6], Belhumeur and Kriegman prove two facts about the illumination space. Assume that the set of images is collected at a fixed resolution and that the object is convex and has a Lambertian reflectance function. Assume also that an arbitrary number of point light sources at infinity are used. Then the illumination space of the object is a convex cone in $\mathbb{R}^n$ and its dimension is equal to the number of distinct surface normals.

Because of the nature of the illumination space as a low-dimensional object, we consider it reasonable to approximate the illumination cone with few eigenimages when we have a convex object. In some cases, we will take a one-dimensional approximation. In others, we will approximate it with the three color filter sheets given by capturing images using the red, green, and blue color filters.

2.9 The Koszul Complex

We will use the Koszul complex as a means of creating a data set, so we give a brief introduction to the definition and some properties here. We follow David Eisenbud's introduction in [17].

Definition 2.17. *Suppose R is a ring and $x \in R$. We define the* Koszul complex of x *to be the complex* $K(x)$:

$$0 \longrightarrow \overset{0}{R} \xrightarrow{x} \overset{1}{R} \longrightarrow 0.$$

The cohomological degree is given above each copy of R. Many authors define the degree in the opposite order, but we will keep this ordering to be consistent with [17]. We denote by $H^i(K(x))$ the homology of $K(x)$ at degree i.

In this example, we have

$$H^1(K(x)) = \frac{\ker(0)}{\mathrm{im}(x)} = \frac{R}{(x)}.$$

And

$$H^0(K(x)) = \frac{\ker(x)}{\mathrm{im}(0)} = \mathrm{Ann}_R(x).$$

Note that x is a nonzerodivisor precisely when $H^0(K(x)) = 0$. $K(x)$ is therefore a free resolution of $\frac{R}{(x)}$ when x is a nonzerodivisor.

The Koszul complex can be similarly defined for any number of variables. Let us look at the Koszul complex of two ring elements x and y.

Definition 2.18. *Suppose R is a ring and $x, y \in R$. We define the* Koszul complex of x and y *to be the complex* $K(x, y)$:

$$\begin{array}{ccccccccc} & & \textit{0} & & \textit{1} & & \textit{2} & & \\ \textit{0} & \longrightarrow & R & \xrightarrow{\binom{x}{y}} & R^2 & \xrightarrow{(-y \;\; x)} & R & \longrightarrow & \textit{0.} \end{array}$$

Now we have

$$H^2(K(x,y)) = \frac{\ker(0)}{\mathrm{im}(-y \;\; x)} = \frac{R}{-yR + xR} \cong \frac{R}{(x,y)},$$

$$H^1(K(x,y)) = \frac{\ker(-y \;\; x)}{\mathrm{im}\binom{x}{y}} = \frac{\{(a,b) | -ay + bx = 0\}}{\{rx, ry | r \in R\}},$$

and

$$H^0(K(x,y)) = \frac{\ker\binom{x}{y}}{\mathrm{im}(0)} = \mathrm{Ann}_R(x,y).$$

It can be shown that if x is a nonzerodivisor, $H^1(K(x,y)) = 0$ if and only if x, y is a regular sequence. Since $H^2(K(x,y)) \cong \frac{R}{(x,y)}$, we have that $K(x,y)$ is a free

resolution of $\frac{R}{(x,y)}$ when x, y is a regular sequence and x is a nonzerodivisor. This fact generalizes in the following way.

Theorem 2.19. *Suppose that R is a local ring with maximal ideal m, and that $x_1, \ldots, x_n$ is a sequence of elements in m. Then $x_1, \ldots, x_n$ is a regular sequence iff $H^{n-1}(K(x_1, \ldots, x_n)) = 0$. When this happens, $K(x_1, \ldots, x_n)$ is the minimal free resolution of $\frac{R}{(x_1,\ldots,x_n)}$.*

2.10 Group Actions

In this paper, we present a variety of data sets, each of which is invariant under an action by a group. We therefore finish this chapter with a review of group actions. We follow the framework of Dummit and Foote in Chapters 1 and 4 of [15]. Throughout this section, let G be a group and A a set. We begin with the definition of a group action.

Definition 2.20. *A map $f : G \times A \to A$, denoted $\cdot$, is a* group action *if it satisfies the following two properties:*

- $g_1 \cdot (g_2 \cdot a) = (g_1 g_2) \cdot a$, *for all $g_1, g_2 \in G$ and for all $a \in A$, and*
- $1 \cdot a = a$ *for all $a \in A$.*

Let us state a theorem that gives an equivalent way to think about group actions in general before considering specific examples.

Theorem 2.21. *An action by the group G on the set A is in one-to-one correspondence with the set of homomorphisms from G into S_A, the symmetric group on the elements of A.*

Proof. First, for each $g \in G$, we may define a map $\sigma_g \colon A \to A$, where $\sigma_g(a) \colon = g \cdot a$. We begin by showing that each σ_g is in fact a permutation of A. That is, each σ_g is a bijection from A to A.

σ_g is one-to-one:

Suppose that there exist $a, b \in A$ such that $\sigma_g(a) = \sigma_g(b)$. Since G is a group, there exists $g^{-1} \in G$, and therefore we have

$$\begin{aligned} \sigma_{g^{-1}}(\sigma_g(a)) &= \sigma_{g^{-1}}(\sigma_g(b)) \\ g^{-1}(g \cdot a) &= g^{-1}(g \cdot b) \\ (g^{-1}g) \cdot a &= (g^{-1}g) \cdot b \\ 1 \cdot a &= 1 \cdot b \\ a &= b. \end{aligned}$$

σ_g is onto:

Since G is a group, there exists $1 \in G$, and by definition of a group action, we have $1 \cdot a = a$. Therefore, we have that each σ_g is a permutation of A.

We next claim that the map $\varphi\colon G \to S_A$ defined by $g \mapsto \sigma_g$ is a homomorphism. We must show that $\varphi(g_1 g_2) = \varphi(g_1) \circ \varphi(g_2)$ for all $g_1, g_2 \in G$. We will show that this equality is true by showing that it holds for all elements a in A. Let $a \in A$. Then we have

$$\begin{aligned} \varphi(g_1 g_2)(a) &= \sigma_{g_1 g_2}(a) \\ &= (g_1 g_2) \cdot a \\ &= g_1 \cdot (g_2 \cdot a) \\ &= \sigma_{g_1}(\sigma_{g_2}(a)) \\ &= \varphi(g_1) \circ \varphi(g_2). \end{aligned}$$

The map φ is referred to as the *permutation representation* associated to the action of G on A. What we have done so far is to show that given a group action $G \times A \to A$, we have a way to assign a homomorphism from G to S_A. We will show that there is a bijection between actions of G on A and homomorphisms from G to S_A by showing that there is an inverse to this assignment.

Suppose that $\varphi\colon G \to S_A$ is a homomorphism. Then define a map $G \times A \to A$ by

$g \cdot a := \varphi(g)(a)$. It should be clear from the definitions that the maps between group actions and homomorphisms from G to S_A are inverses. □

Let us now take a look at some examples.

Example 2.22. *The trivial group action is the action by which the group action is $g \cdot a = a$ for all $g \in G$ and for all $a \in A$. In this case, each element $g \in G$ is associated to the identity permutation. This example shows that the map that defines the associated permutation representation need not be injective. When it is injective, we say that the action is* faithful. *The* kernel *of the action $G \times A \to A$ is defined as $\{g \in G \mid g \cdot a = a \, \forall a \in A\}$. In this example, the kernel is all of G and the action is faithful only when $|G| = 1$.*

Example 2.23. *Let D_{2n} be the dihedral group of order $2n$. Recall that the dihedral group is the set of symmetries of a regular n-gon. With a fixed labeling of the vertices of the n-gon, we have that each $\alpha \in D_{2n}$ defines an element $\sigma_\alpha \in S_n$. The map $D_{2n} \times \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ defined by $(\alpha_i, i) \mapsto \sigma_\alpha(i)$ is a faithful group action of D_{2n} on $\{1, 2, \ldots, n\}$. In the case when $n = 3$, the map is surjective. Hence in this case, we have that there is an isomorphism between D_6 and S_3. For any n larger than 3, this is not the case. The fact that it is the case when $n = 3$ tells us that every permutation of the vertices is achievable by a symmetry of the triangle.*

One very important property of a group action for our work is the following.

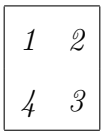
Definition 2.24. *Let X be a subset of A, and suppose that there exists a group action of G on A. Denote by GX the set*

$$GX := \{g \cdot x | \, g \in G, x \in X\}.$$

If $GX = X$, then we say that X is invariant *under the group action of G.*

Note that since $1 \in G$, we always have that $X \subseteq GX$. The content of this definition is therefore in the containment of GX in X. We give one example and one non-example.

Example 2.25. *Consider the dihedral group of order 8, with the square having the following labeling of vertices.*



Let X be the set of reflections about the vertical axis of the square (as a set of orderings of the vertices, it is $\{[1,2,3,4],[2,1,4,3]\}$). Note that X is a subgroup of D_8. There is a group action of D_8, and therefore X, on the set $\{1,2,3,4\}$, defined by $\alpha \cdot i = j$, where j is the element of $\{1,2,3,4\}$ to which i is sent under the motion α on the square. The subset $\{1,2\}$ is invariant under the action of X, but the set $\{1,2,3\}$ is not.

Finally, we state a property of group actions that we will require of some data sets defined by group actions.

Definition 2.26. *A group action G on A is said to be* transitive *if for every $x, y \in A$, there exists $g \in G$ such that $g \cdot x = y$.*

Example 2.27. *The dihedral group D_8 acts transitively on $\{1,2,3,4\}$ because for any pair $(i,j) \in \{1,2,3,4\} \times \{1,2,3,4\}$, there is some motion of the square that sends i to j. The action of the subgroup X on $\{1,2,3,4\}$ described in the previous example is not a transitive action, because, for instance, there is no element of X that sends 1 to 3.*

Chapter 3

Data Sets That are Invariant Under a Group Action

3.1 Introduction

We present here examples of data sets that can be understood through group actions as well as ways in which they may be analyzed. For a brief introduction to group actions, see Section 2.10. The applications that we discuss are in the context of image analysis. The general goal is the following: suppose that we are given a data set, which is invariant under an action of a particular group G. Suppose also that we choose two elements of the data set, x and y. In some cases, we will seek to determine if x and y are equivalent under the group action. That is, we wish to know if an element g exists such that $g \cdot x = y$. In other cases, we start with a transitive action and seek to find an element $g \in G$, such that $g \cdot x = y$. Because we are working with real data sets, there is also induced noise. This means that we cannot hope for symbolic computations that find such a g. Rather, we hope to find numeric methods, which allow us to find an approximation of g with some reliability. We present here a couple simple examples for the purpose of discussion. For a more detailed analysis of equivalence of subspaces associated to matrices, see for example [53].

3.2 The Trivial Group

Let us begin with the simplest case, that of the trivial group. If our group G consists only of the identity element, then we seek to know when two elements are equivalent under multiplication by the identity. This question is trivial if there is no noise, but when noise exists, it becomes an interesting question.

Example 3.1. *Let us consider the set of matrices of size $m \times n$, with real entries. If the noise is known to be bounded by a sufficiently small ϵ, then it will be possible to put the matrix into reduced row echelon form, with the assumption that values within some value c_ϵ of zero are made to be zero. This gives a way to compare matrices. For example, consider the matrix*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Let B be the matrix attained by adding random noise to each entry of A. That is, we add a random number in the interval $[-\epsilon, \epsilon]$ to each entry of A. With $\epsilon = 10^{-13}$, we get

$$B = \begin{bmatrix} 0.99999999999992 & 2.00000000000005 & 3.00000000000006 \\ 4.00000000000010 & 4.99999999999996 & 5.99999999999997 \\ 7.00000000000009 & 7.99999999999998 & 8.99999999999994 \end{bmatrix}.$$

Let $c_\epsilon = 1.598721155460225 * 10^{-14}$. Then the reduced row echelon form of B is the 3×3 identity matrix, even though

$$rref(A) = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

Changing ϵ to 10^{-14}, we get $rref(B) = rref(A)$.

An alternative method of determining if A is equivalent to B under the action of the trivial group is to use principal angles. Suppose that we flatten both A and B

and then normalize. Call the normalized, flattened vectors A_f and B_f, respectively. Then the singular value of $A_f^T B_f$ is the cosine of the principal angle between the two linear subspaces of $\mathbb{R}^9$. With ϵ as large as 10^{-7}, the principal angle is numerically zero, so this method performs significantly better than simply computing the reduced row echelon form. Even up to $\epsilon = 10^{-1}$, we find a principal angle of approximately 0.011724 *radians, suggesting that principal angles still detect the closeness of the matrices A and B.*

3.3 The Symmetric Group

Let us now consider an action of a subgroup of S_n on $\mathrm{GL}(n, \mathbb{R})$. Note that S_n acts on a matrix $M \in \mathrm{GL}(n, \mathbb{R})$ by permuting the columns of M. Equivalently, S_n is the set of matrices P, which have exactly one one in each row and in each column, and the action is post-multiplication by P.

Now let G be the subgroup of S_n consisting of cyclic permutations. G is the set of permutations on $\{1, 2, \ldots, n\}$ defined by

$$\sigma(a_i) = a_{i+k(\mathrm{mod} n)}.$$

Example 3.2. *There are many settings in which the action of G on $GL(n, \mathbb{R})$ might occur and in which we might seek to determine if two elements of $GL(n, \mathbb{R})$ are equivalent. Suppose that observations are made of an object with periodic behavior. Let us assume that two data sets, D_1 and D_2, are collected, each containing information about one complete period. Let us also assume that a third data set, D_3, is created by making observations of a different object. We would like to be capable of determining that D_1 and D_2 are equivalent and that D_3 is not equivalent to either D_1 or D_2. This question is one that we can describe in the language of the action of G on $GL(n, \mathbb{R})$: given the nature of these data sets, we have that $D_1 \equiv D_2$, but $D_3 \not\equiv D_i$ for $i = 1, 2$.*

The technique of computing reduced row echelon form that worked well with minimal noise in the previous example is now of no use. This is due to the fact that, in

general, column swaps result in different reduced row echelon forms. For example, let A be the matrix as defined in the previous example and let B be the cyclic permutation of A given by

$$B = \begin{bmatrix} 3 & 1 & 2 \\ 6 & 4 & 5 \\ 9 & 7 & 8 \end{bmatrix}.$$

Even without adding in noise, we get that the reduced row echelon form of B is

$$rref(B) = \begin{bmatrix} 1 & 0 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 0 \end{bmatrix},$$

while

$$rref(A) = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

We can use the method of computing principal angles that was used in the previous example, but we find that its signal is weaker. As an example, if we use the matrix B above, we find that the smallest principal angle between the flattened, normalized vectors is approximately 0.251978 *radians. This is significantly larger than the principal angles that suggested equivalence of matrices in the previous example. However, computing principal angles gives some advantage in that the method shows robustness with added noise. If we do the same computation, but this time add noise to B with ϵ as large as* 10^{-7}*, the principal angle remains* 0.251978.

A better technique than either reduced row echelon form or principal angles in this setting is to take advantage of a property of the discrete Fourier transform. We give a brief review of the transform and the property that we wish to use here before returning to this problem.

Recall that given a sequence $\{x_n\}_{n=0}^{N-1}$, the Fourier transform of $\{x_n\}_{n=0}^{N-1}$ is defined

to be the sequence $\{X_k\}$*, where*

$$X_k = \mathcal{F}\left(\{x_n\}\right)_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}}.$$

The behavior of the sequence X_k *given a circular shift of the sequence* x_n *is known and is described by the shift theorem. We state it and give a proof here.*

Theorem 3.3. *Fix an integer* m *and suppose that* $\mathcal{F}(\{x_n\})_k = X_k$*. Then*

$$\mathcal{F}(\{x_{n-m}\})_k = X_k e^{\frac{-2\pi ikm}{N}},$$

where the indices of the sequence $\{x_{n-m}\}$ *are taken mod* N*.*

Proof. We have

$$\begin{aligned}
\mathcal{F}(\{x_{n-m}\})_k &= \sum_{r=0}^{N-1} x_{N-m+r} e^{\frac{-2\pi ikr}{N}} \\
&= e^{\frac{-2\pi ikm}{N}} \sum_{r=0}^{N-1} x_{N-m+r} e^{\frac{-2\pi ik(-m+r)}{N}}, \\
&= e^{\frac{-2\pi ikm}{N}} \sum_{r=0}^{N-1} x_{N-m+r} e^{\frac{-2\pi ik(N-m+r)}{N}}, \\
&= e^{\frac{-2\pi ikm}{N}} X_k,
\end{aligned}$$

where we have used the periodicity of the exponential function. □

One consequence of the shift theorem is that a circular shift of the sequence $\{x_n\}$ *results in an output sequence whose coefficients have the same magnitude as the coefficients in the original output sequence* X_k*. We can therefore exploit this property in our example. There are several approaches we can take to exploit this property; we will discuss one here and will use another in Chapter 8.*

If one data matrix D_i *is equivalent to a different data matrix* D_j *under the action by* G*, then the magnitude of the Fourier coefficients of each row will be the same. So we can start by taking the difference of the absolute values of the coefficients row by row. In the example above, we find that the difference in the magnitudes are*

numerically zero for ϵ as large as 10^{-7}. Note, however, that it is possible that each row was permuted by a different cyclic permutation. Therefore, this method can be used to determine when two data sets are not equivalent, but more is needed to determine that they are equivalent. Once it has been determined that the rows of each matrix have approximately equal magnitudes of Fourier coefficients, one can inspect whether or not there exists a cyclic permutation of the columns. The advantage is that it is often the case that one must inspect a significantly smaller set of matrices for a cyclic permutation after eliminating the majority using Fourier coefficients.

3.4 The Special Orthogonal Group

In Chapters 5, 6, and 7, we consider a data set that is invariant under an action of the special orthogonal group, $SO(n, \mathbb{R})$. The setup for data set collection is the following. Images of an object on a rotating record player are captured. For some data sets, we use only grayscale intensity images; these are generated using a linear combination of the amount of light received on each pixel in the red, green, and blue color filters. The linear combination is given by $v_{\text{gray}} = 0.2989 v_{\text{red}} + 0.5870 v_{\text{green}} + 0.1140 v_{\text{blue}}$, where v_* represents the value of the pixel in the given color filter. In other data sets, we use the information from all three color filters individually, and in others, we retain all the information for all three color filters.

The camera is located at roughly the same height as the object so that the camera captures side views of the object. Although we rotate the object on the record player, we are, in effect, modeling the situation in which an object is fixed and a camera is allowed to move in a circle about the object. Because any motion of the camera along the circle results in an image that is in the data set, we say that the data is invariant under an action of $SO(2, \mathbb{R})$.

We exploit the invariance of this set under the action by $SO(2, \mathbb{R})$ in several ways. Because $SO(2, \mathbb{R}) \cong S^1$, each data point taken during a single rotation can

be viewed as corresponding to a point on S^1. In Chapter 5, we use this fact to get a visual representation of the closed loop corresponding to data sets collected for several different objects.

In Chapters 6 and 7, we again use this structure of the data set to construct a vector bundle over S^1 on which we can apply an algorithm to determine (locally) optimal camera location distribution.

3.5 $SO(n, \mathbb{R}) \times SO(n, \mathbb{R})$

In Chapter 8, we combine a data set of images of a rotating object with artificial rotation. That is, for each image of the object, we artificially generate images that represent rotation of the camera along the axis between the camera and the object. In this way, we model a data set in which the camera is allowed to move along a circle around the object and is allowed to rotate along its own horizontal axis. Therefore the data set is invariant under an action by the group $SO(n, \mathbb{R}) \times SO(n, \mathbb{R})$. We exploit this invariance by designing an algorithm which can detect the intersection of two such data sets.

3.6 Extensions

We note here that the data sets we study in this paper are fixed under several variations of state that may not be fixed in many real-world settings. For instance, when we collect images of a rotating object, we assume that the images are registered to have roughly the same origin of rotation. In contrast, large variations in registration occur if, for example, the camera being used is hand-held. An extension of this work would therefore expand these algorithms to be applied to a data set which is invariant under an action, not just of the special orthogonal group, but of the group $SE(n)$. Recall that $SE(n)$ is the special Euclidean group, or the symmetry group of

$n-$dimensional Euclidean space. That is, we would like to allow not only for rotations of an object, but also for shifts of camera location.

Chapter 4

Processing High Speed Data Using the Fast Fourier Transform

4.1 Introduction

The standard wall socket on a typical building in the US provides a rectified 60Hz current. As a consequence, lights powered by such a source pulse at a rate of 120 cycles per second. This pulsating of a light source is far too fast to be observed by the human eye but can be observed through the aid of a high speed camera. A phenomenon known as aliasing can occur when an object is illuminated with such lighting and a sequence of images of the object are captured at a rate comparable to 120 frames per second. An acoustic version of this phenomenon is the background beating one can hear when listening to two slightly differing frequencies. In this chapter, we present evidence that the variation in lighting is observed in our collected data. We then apply the Fourier transform as a method for removing these oscillations.

4.2 Control Data Sets

We begin with examples of data sets consisting of pictures of a stationary object. Eight data sets of approximately 2000-4000 pictures each are collected at a rate of 1000 pictures per second and a resolution of 256×256 pixels. All analysis is done separately for each color filter and for grayscale. One data set is collected under overhead and supplemental lighting, which is powered by an alternating current. The other data sets are collected under lighting powered by a direct current. The pictures are of a stationary object, so any variation in images is from either the light sources or the camera itself.

4.3 Data Set of a Rotating Object

Pictures are captured at a rate of approximately 220 pictures per second and at a resolution of 512×512 pixels. The object is rotating at $33\frac{1}{3}$ rotations per minute. In this example, the object is a white, silver, and blue tennis shoe. A picture of the shoe can be seen in Figure 4.1. All analysis is done separately for each of the three color filters as well as for grayscale.

Figure 4.1: Shoe

4.4 Methods

As described in Chapter 5, each digital image matrix can be flattened to a point in $\mathbb{R}^{r^2}$, where the resolution is $r \times r$. After flattening each image matrix in this way and considering it as a column vector, we construct a data matrix, X, by concatenating column vectors in order of time of image capture. We use sufficiently many images to capture approximately one full rotation of the object. Hence, the number of columns of X varies slightly throughout analysis by data set and by filter.

Let $\hat{X}$ be the projection of X into $\mathbb{R}^3$ via the method described in Section 5.2. Hence X has 3 rows and the number of columns of X is equal to the number of pictures in the data set. Define $\mathbf{v}^{(i)}$ to be the row vector given by the i^{th} row of $\hat{X}$ for $i = 1 \ldots 3$. Let $\mathbf{f}^{(i)}$ be the discrete Fourier transform of $\mathbf{v}^{(i)}$. Note that the vectors $\mathbf{f}^{(i)}$ have the property that the first entry is the sum of the entries in $\mathbf{v}^{(i)}$ and that the rest of the entries in each $\mathbf{f}^{(i)}$ come in conjugate pairs. Entry 2 is the complex conjugate of the last entry, entry 3 is the complex conjugate of the second-to-last entry, and so forth. In this experiment, we keep the first 10 coefficients as well as the corresponding 9 coefficients at the end of the vector $\mathbf{f}^{(i)}$. The rest of the coefficients are set to zero. The number 10 gives good results in this example, but any number may be used with slightly varying results. Using less than the first 5 coefficients or more than the first 20 leads to noticeably worse reconstructions. We next compute vectors $\mathbf{g}^{(i)}$ by performing the inverse discrete Fourier transform on the zeroed $\mathbf{f}^{(i)}$'s.

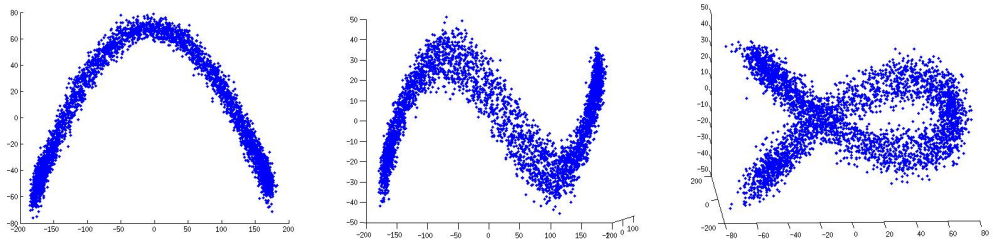
4.5 Results

4.5.1 Control Data Sets

Stationary Data Under AC Lighting

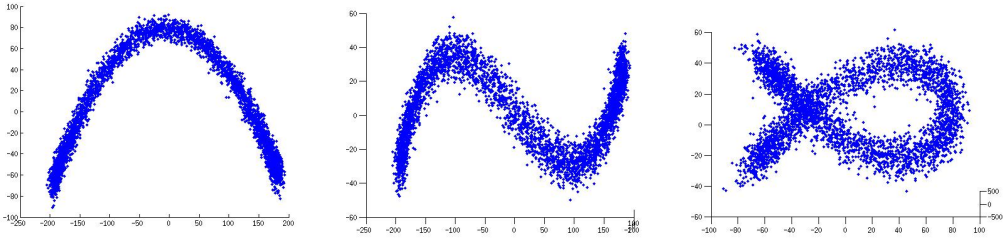
In Figures 4.2-4.4, we see various views of the 3-dimensional projections of the stationary punch bowl under alternating current lighting in the red, green, and blue

filters. Observe that there is a strong signal, which produces similar results in each of the three filters. If we plot the points as a time series with each point plotted in the order in which it was captured, then we see that the points travel back and forth along the arch. This is highly suggestive that one end of the arch corresponds to maximal light, while the other corresponds to minimal light.



(a) View 1 (b) View 2 (c) View 3

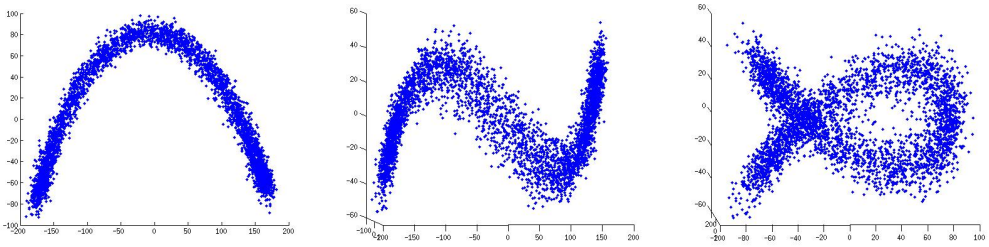
Figure 4.2: Stationary Data Under AC Lights, Red Filter



(a) View 1 (b) View 2 (c) View 3

Figure 4.3: Stationary Data Under AC Lights, Green Filter

Observe the graph of the absolute value of the Fourier coefficients, shown in Figure 4.5. We see that there are several frequencies at which the data gives a relatively strong signal. However, the overwhelming majority of the Fourier basis vectors have a large magnitude. Hence, the basis vectors corresponding to those frequencies alone do not provide a good representation of the alternating current. It will therefore be necessary in any processing of the data using the Fourier transform to zero out a significant number of basis vectors.



(a) View 1 (b) View 2 (c) View 3

Figure 4.4: Stationary Data Under AC Lights, Blue Filter

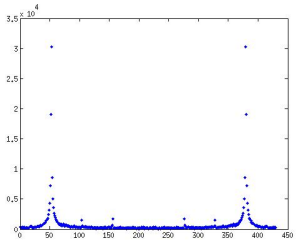


Figure 4.5: Absolute Value of Fourier Basis Coefficients

In Figure 4.6, we see the plot of the red filter data projected onto one dimension. That is, we plot only the first coordinate of the three-dimensional projection. In order to make the plot less crowded, we show only the first 100 points. Note that the AC light data set is periodic, with period approximately $\frac{1000}{120} = 8\frac{1}{3}$. This is consistent with the period of the alternating current, suggesting that this variation is, in fact, a result of the pulsating of the lights.

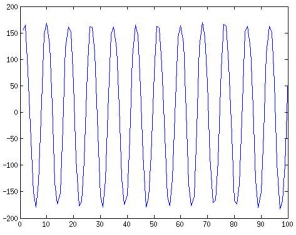
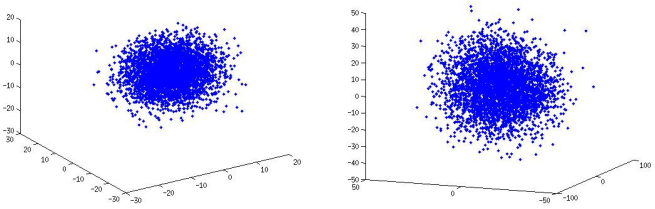


Figure 4.6: First Coordinate of Projection

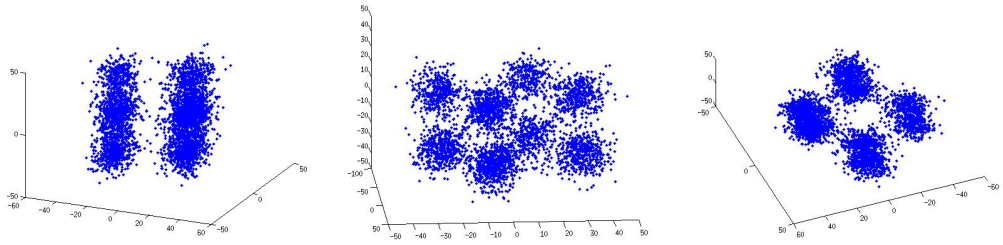
Stationary Data Under DC Lighting

The projections of a stationary red punch bowl collected under direct current lighting are shown in Figures 4.7 and 4.8. In contrast to the projection of the AC data in the red and green filters, the data sets corresponding to DC lighting are distributed about a single point in space and appear to have no organization with respect to time. Surprisingly, we see that the data in the blue filter is distributed about eight points in space. Because the lighting for this data set is powered by a direct current, it seems likely that this variation has a source within the camera.



(a) Red Filter (b) Green Filter

Figure 4.7: Stationary Red Punch Bowl Under DC Lights, Red and Green Filters

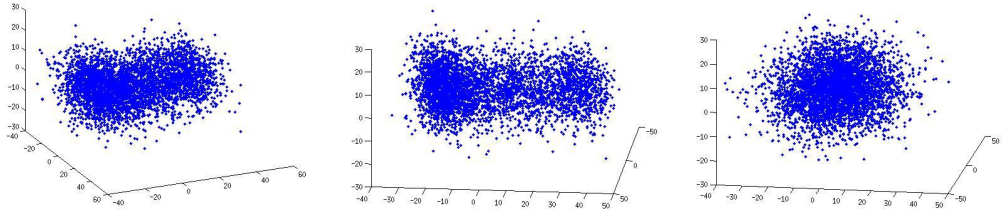


(a) View 1 (b) View 2 (c) View 3

Figure 4.8: Stationary Red Punch Bowl Under DC Lights, Blue Filter

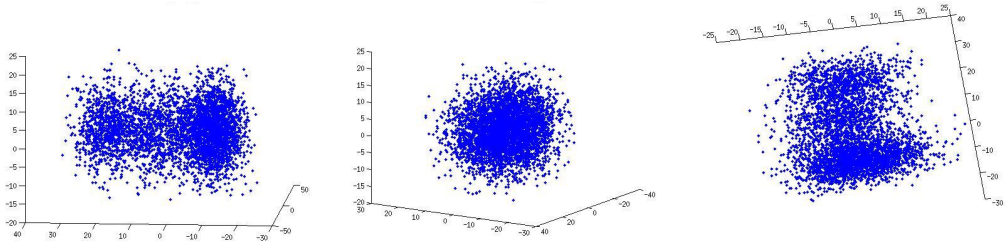
Given that images of a red object show unusual structure in the blue filter only suggests that the camera generates structured noise when there is very little light in a given filter. We therefore collect two more data sets of stationary objects under direct current lighting. The first is a set of pictures of a blue cup. If the hypothesis

is correct, we should see a distribution of points about 8 locations in the red filter and a distribution about one location in the green and blue filters. The results of this experiment are presented in Figures 4.9-4.11. Note that we do not see such distributions. We do, however, see some structure in each of the filters. We can again conclude that there is some source of structured noise from the camera itself.



(a) View 1 (b) View 2 (c) View 3

Figure 4.9: Stationary Blue Cup Under DC Lights, Red Filter

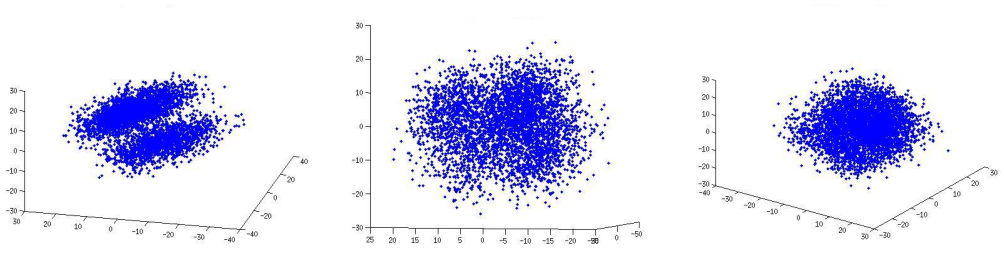


(a) View 1 (b) View 2 (c) View 3

Figure 4.10: Stationary Blue Cup Under DC Lights, Green Filter

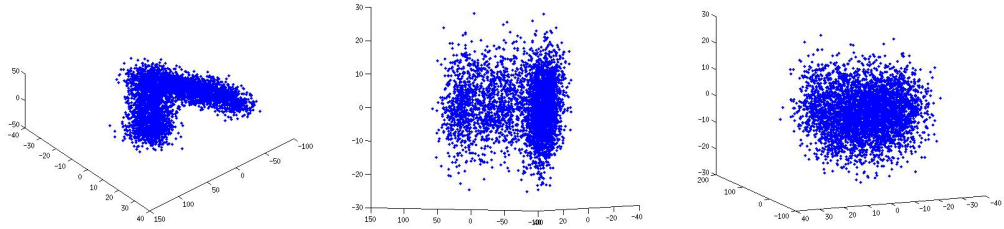
In Figures 4.12-4.14, we see the results of a projection of a data set of pictures of a blue cup with red and green tape attached. If any set should generate a random distribution of points about one location in space for each filter, this is it. We see, however, that some unusual structure still exists in the projection.

We test the hypothesis that the noise is somehow related to the lighting by taking pictures of a direct current light source. The projection of this data set is shown in Figure 4.15. Note that here we actually see well-distributed points about one location



(a) View 1 (b) View 2 (c) View 3

Figure 4.11: Stationary Blue Cup Under DC Lights, Blue Filter

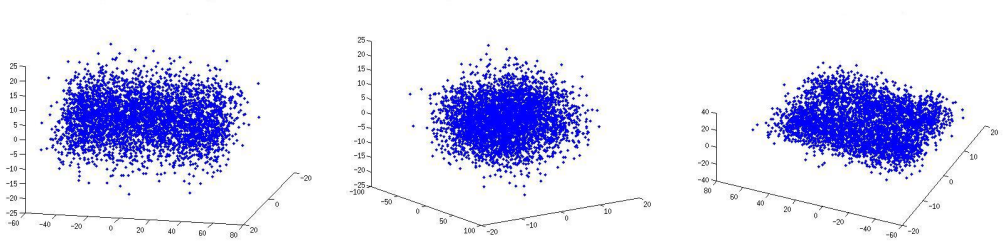


(a) View 1 (b) View 2 (c) View 3

Figure 4.12: Stationary Blue Cup with Red and Green Tape, DC Lights, Red Filter

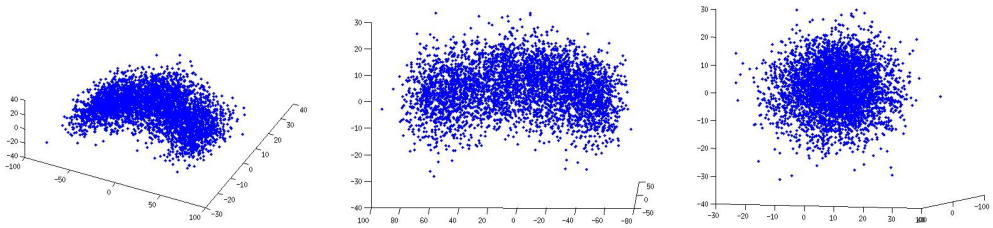
in each filter. It is therefore reasonable to assume that any other structure in the data is not a result of the direct current lighting.

Finally, we collect a data set by taking pictures of nothing. That is, we cover the lens of the camera and take pictures. The projection of this data set is shown in Figures 4.16 and 4.17. Here again, we see a well-behaved distribution in the red and green filters and an unusual distribution in the blue filter. We must conclude that the camera adds some noise to any collected data set, with the blue filter being most susceptible. Because the noise structure changes upon collection of different data sets, it is not something that we can consistently correct. It is, however, on a relatively small scale.



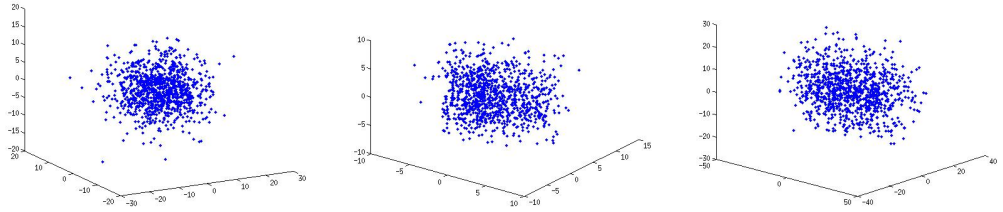
(a) View 1 (b) View 2 (c) View 3

Figure 4.13: Stationary Blue Cup with Red and Green Tape, DC Lights, Green Filter



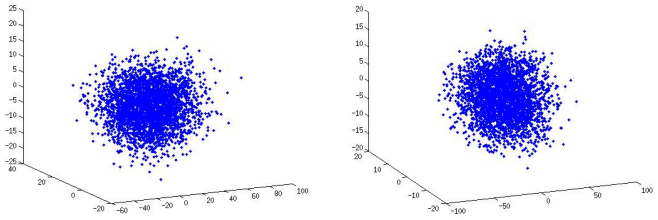
(a) View 1 (b) View 2 (c) View 3

Figure 4.14: Stationary Blue Cup with Red and Green Tape, DC Lights, Blue Filter



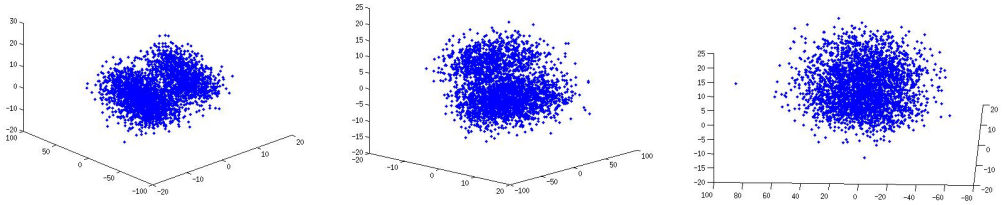
(a) Red Filter (b) Green Filter (c) Blue Filter

Figure 4.15: Spotlight (DC Light)



(a) Red Filter (b) Green Filter

Figure 4.16: No Light, Red and Green Filters



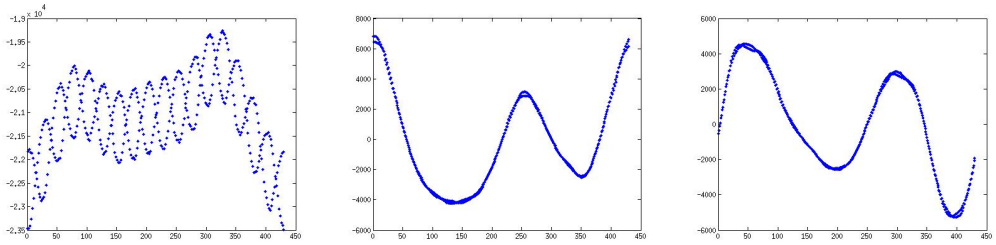
(a) View 1 (b) View 2 (c) View 3

Figure 4.17: No Light, Blue Filter

4.5.2 Data Set of a Rotating Object

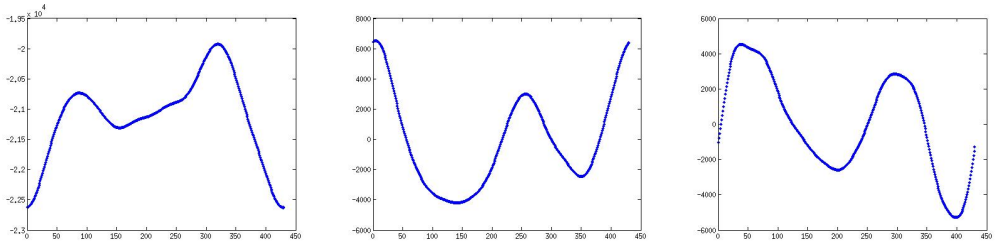
In Figures 4.18, 4.20, 4.22, 4.24, and 4.26, we show the results of this experiment with no processing to remove the effects of the lights. Let $\hat{X}$ denote the projection of X as described in Chapter 5. Figures 4.18, 4.20, 4.22, and 4.24 show plots of each row of $\hat{X}$ in the various filters. Figure 4.26 shows the projection into $\mathbb{R}^3$ for each filter. Each point is an ordered triple defined by a column of $\hat{X}$.

Figures 4.19, 4.21, 4.23, and 4.25 show the plots of the $\mathbf{g}^{(i)}$, that is, the reconstruction of the rows of $\hat{X}$ after zeroing some Fourier coefficients. Figure 4.27 shows the reconstruction of the projection to $\mathbb{R}^3$ using the $\mathbf{g}^{(i)}$ for each filter. These reconstructions suggest that the actual curves representing the data in $\mathbb{R}^3$ are smooth and not self-intersecting.



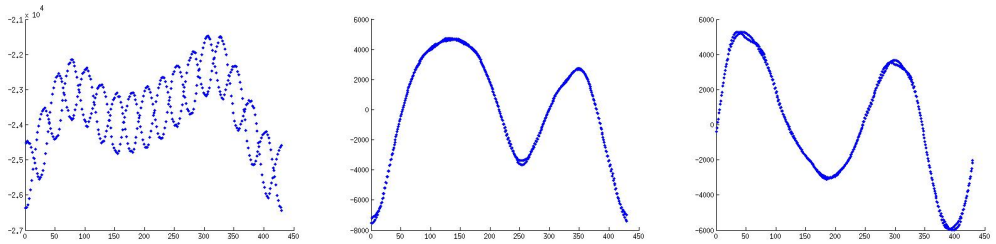
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.18: Rows of $\hat{X}$ Before Processing, Red Filter



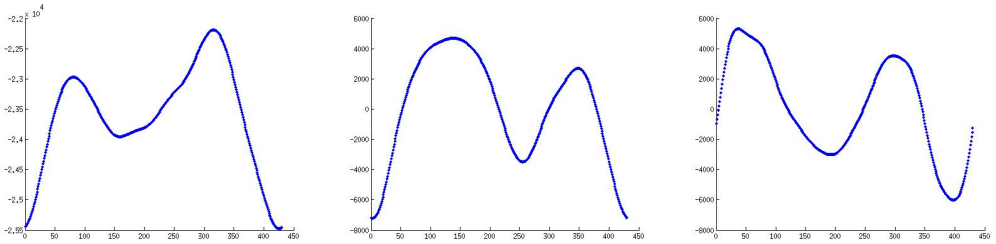
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.19: Rows of $\hat{X}$ After Processing, Red Filter



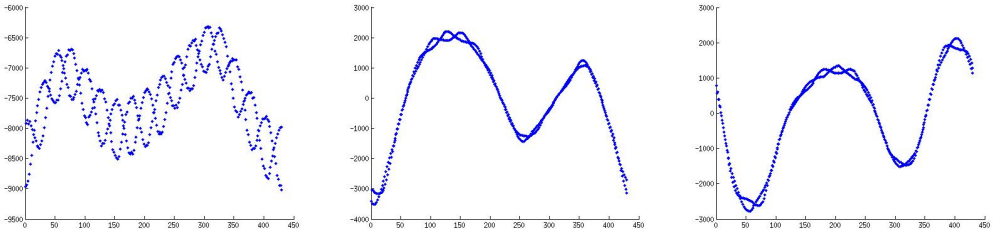
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.20: Rows of $\hat{X}$ Before Processing, Green Filter



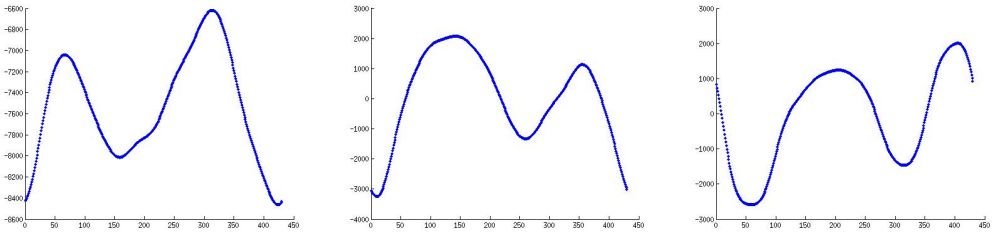
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.21: Rows of $\hat{X}$ After Processing, Green Filter



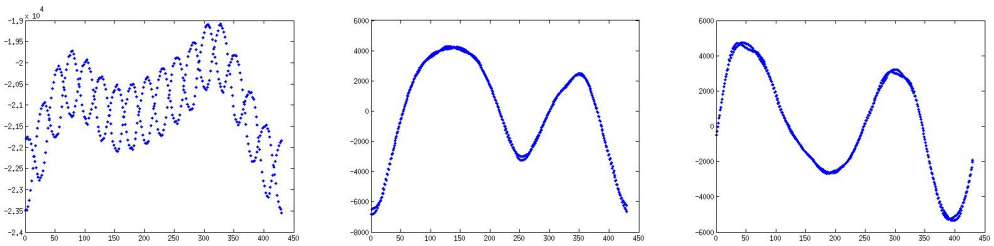
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.22: Rows of $\hat{X}$ Before Processing, Blue Filter



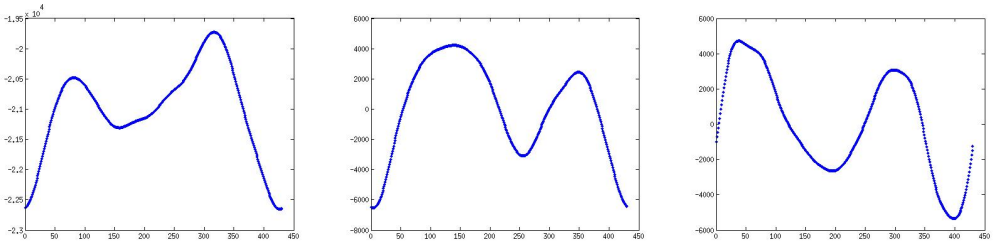
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.23: Rows of $\hat{X}$ After Processing, Blue Filter



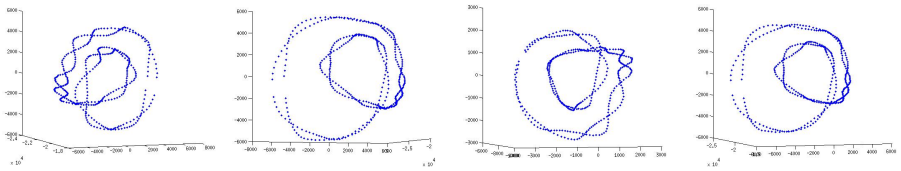
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.24: Rows of $\hat{X}$ Before Processing, Grayscale



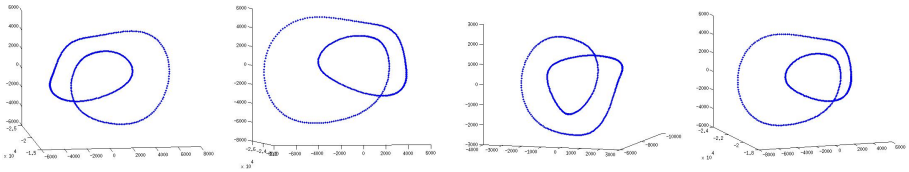
(a) First Coordinate of $\hat{X}$ (b) Second Coordinate of $\hat{X}$ (c) Third Coordinate of $\hat{X}$

Figure 4.25: Rows of $\hat{X}$ After Processing, Grayscale



(a) Red Filter (b) Green Filter (c) Blue Filter (d) Grayscale

Figure 4.26: Projections into $\mathbb{R}^3$ Before Processing



(a) Red Filter (b) Green Filter (c) Blue Filter (d) Grayscale

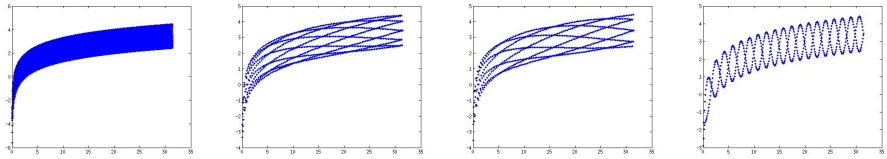
Figure 4.27: Projections into $\mathbb{R}^3$ After Processing

4.6 Aliasing

We note here that the plots of the individual coordinates of $\hat{X}$ are examples of aliasing. In the plots of the first coordinate, for example, it appears that there are two sine waves superimposed on another signal. We claim that this is an effect of our sampling rate and does not represent the alternating current signal or the signal from the data.

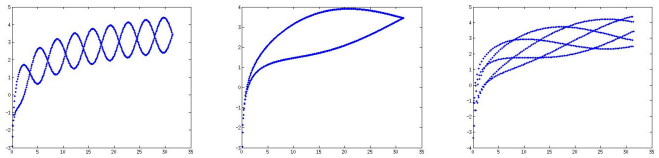
Recall that we capture pictures at approximately 220 pictures per second, and the rectified current completes 120 cycles per second. It follows that by the time we capture the second picture, the current for the lights has already completed more than one cycle. Therefore, one should not expect to observe the actual sine wave of the alternating current.

As an example of the artifacts one can see as a result of aliasing, we have included Figures 4.28 and 4.29. All graphs are samples of $\sin(60x) + \log(x)$ over the interval $[0, 10\pi]$. In Figure 4.28a, we see that with very frequent sampling, the true nature of the function is observable. To get a good view of the function in Figure 4.28a, the figure must be significantly enlarged. We get the strongest effects from aliasing in Figure 4.29b, where the sampling rate is a multiple of the frequency of $\sin(60x)$.



(a) 10,000 Samples (b) 1000 Samples (c) 800 Samples (d) 620 Samples

Figure 4.28: Various Samplings of $\sin(60x) + \log(x)$



(a) 610 Samples (b) 600 Samples (c) 500 Samples

Figure 4.29: Various Samplings of $\sin(60x) + \log(x)$

Chapter 5

A Geometric Representation of a Rotating Object

5.1 Introduction

We use a high speed digital camera to capture a sequence of still images of an object. The object rotates along the axis through the center of the object and perpendicular to the axis between the object and the camera. A digital image can be stored as a matrix of integer entries, where each integer is a measure of the intensity of light on the corresponding pixel. This matrix can then be viewed as a point in a high-dimensional vector space $\mathbb{R}^n$. Let X be the data set consisting of points in $\mathbb{R}^n$ corresponding to images of one object at various stages of rotation. We conjecture that for certain choices of projection into lower dimensional space, such a data set will project to a sampling of a curve homeomorphic to S^1. In this paper, we focus on projections of data sets using Principal Component Analysis.

5.2 Methods

Images are collected using a Phantom Vision High Speed Camera 4.2 as collections of three matrices, one for each filter color: red, green, and blue. The image resolution is either 512×512 or 256×256, depending on the data set. We will denote the resolution of an image by $r \times r$. Hence, one image is represented by three $r \times r$ matrices. Throughout this chapter, we conduct analysis in each filter individually. We therefore refer to an image as one $r \times r$ matrix. We will use p to denote the number of images in a given data set.

Given a digital image I collected in a chosen filter, we wish to map it to a point in $\mathbb{R}^n$, for some large n. Under this map, we want each integer entry of I to be preserved, so we use $n = r^2$. There are many ways to map a matrix I to $\mathbb{R}^n$; we use the following method. Concatenate rows of I in order from top to bottom. Hence, the map is given by

$$\begin{aligned} \mathbb{R}^{r\times r} &\rightarrow \mathbb{R}^{r^2} \\ I &\mapsto v, \text{ defined by} \\ v_i &= I_{jk}, \text{ where } j = \left\lfloor \frac{i}{r} \right\rfloor + 1, \text{ and } k = i \mathrm{mod} r. \end{aligned}$$

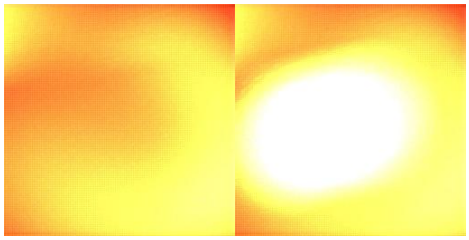
Mapping a matrix into $\mathbb{R}^n$ in this way is referred to as *flattening* a matrix.

We take pictures of an object spinning on a record player. We complete this experiment for data sets consisting of images of the following objects: an orange plastic jack o' lantern; a (somewhat flat) red, white, and blue volleyball; a shiny, red punch bowl on a stand; and a red punch bowl with white tape on it. We place varying numbers of pieces of tape on the punch bowl in order to create data sets with different types of symmetry. The first of these data sets is images of a punch bowl with one piece of horizontal tape. The second is of a punch bowl with one horizontal piece on one side and two pieces of tape forming an $\times$ on the other side. The third data set is of a punch bowl with two vertical pieces of tape on opposite sides of the bowl, one horizontal piece in between, and an $\times$ opposite the horizontal piece. The final set is

of a punch bowl with vertical pieces of tape on opposite sides and horizontal pieces on the opposite sides in between. Pictures of these objects can be seen in Figure 5.1 and in Figures 5.2 to 5.5. We note here that the strength of the light source varies dramatically in the red punch bowl data sets, but that the tape is clear in each set and therefore the light variation should have little effect on projection results.

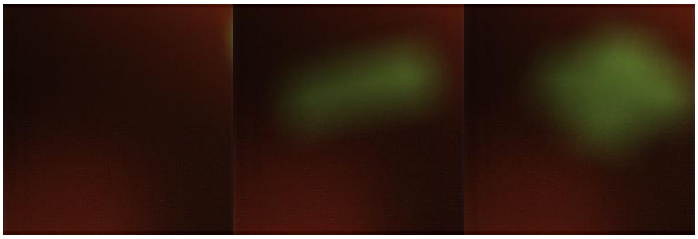
(a) Jack o' Lantern (b) Volleyball (c) Punch Bowl

Figure 5.1: Original Images: Jack O' Lantern, Volleyball, and Punch Bowl



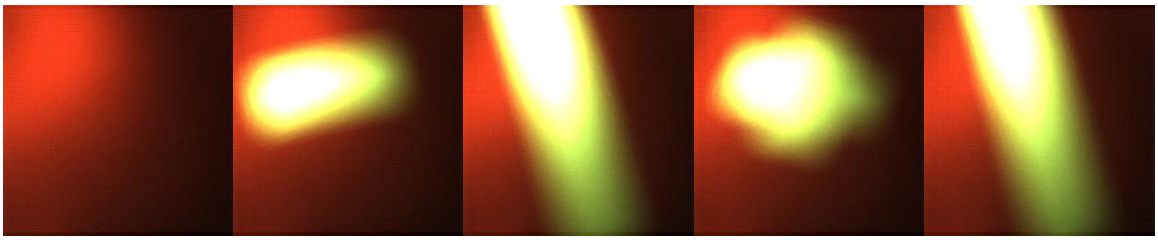
(a) Punch Bowl (b) Tape

Figure 5.2: Original Images: Punch Bowl with One Tape Data Set



(a) Punch Bowl (b) Tape 1 (c) Tape 2

Figure 5.3: Original Images: Punch Bowl with Two Tapes Data Set



(a) Punch Bowl (b) Tape 1 (c) Tape 2 (d) Tape 3 (e) Tape 4

Figure 5.4: Original Images: Punch Bowl with 1 Pair and 2 Different Tapes Data Set

(a) Punch Bowl (b) Tape 1 (c) Tape 2 (d) Tape 3 (e) Tape 4

Figure 5.5: Original Images: Punch Bowl with 2 Pairs Tapes Data Set

We create a data matrix X, consisting of all flattened images of a rotated object. This is done by concatenating column vectors in $\mathbb{R}^n$ representing images of the object. Concatenation is ordered by time of image capture. We compute the singular value decomposition of X, to get matrices U, Σ, and V satisfying $X = U\Sigma V^\top$. Recall that the matrix U consists of ordered left singular vectors, or principal components. The first column of U gives the mean of the data. Each successive column, or principal component, captures the next most important feature of the data. Therefore, by using the second, third, and fourth columns of U to project the data into $\mathbb{R}^3$, we simultaneously mean-subtract the data and get a representation of each picture in $\mathbb{R}^3$ which contains information about the three most important features of the object. That is, we have chosen to project onto the three vectors which capture more statistical variance in the data than any other three orthonormal vectors. If $\hat{U}$ is columns two through four of U, then the projection π is given by

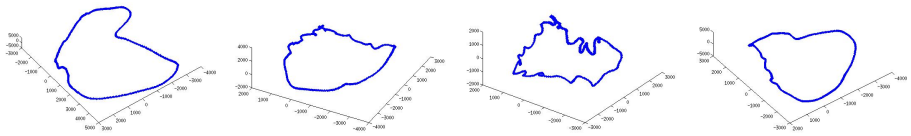
$$\begin{aligned} \pi\colon \mathbb{R}^{r^2 \times p} &\to \mathbb{R}^{3\times p} \\ X &\mapsto \hat{U}^\top X. \end{aligned}$$

The image of X is a matrix, each of whose columns is a three-tuple corresponding to an image in the original data set. While many low-dimensional representations of the data can be useful, we choose to project into $\mathbb{R}^3$ because it is the largest dimension in which we can get a natural visual representation of the data.

5.3 Results

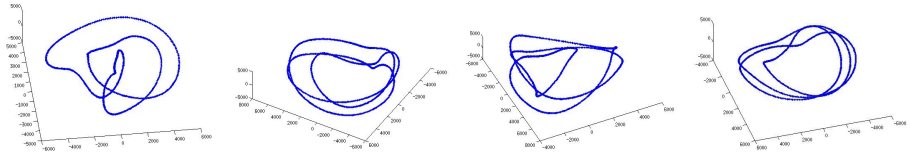
We begin with projections into $\mathbb{R}^3$ for each data set and each filter for three data sets. Each of these sets of images were captured at a rate of approximately 220 pictures per second and at a resolution of 512×512 pixels. Enough pictures are used to capture approximately three rotations of the object. The objects used in these sets are the jack 'o lantern, the volleyball, and the punch bowl. Results for all objects are

from data collected under direct current lighting to rule out any effects from lighting variation. The projections are shown in Figures 5.6, 5.7, and 5.8.



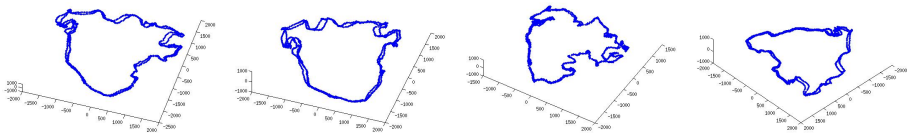
(a) Red Filter (b) Green Filter (c) Blue Filter (d) Grayscale

Figure 5.6: Projections into $\mathbb{R}^3$ of Jack o' Lantern



(a) Red Filter (b) Green Filter (c) Blue Filter (d) Grayscale

Figure 5.7: Projections into $\mathbb{R}^3$ of Volleyball

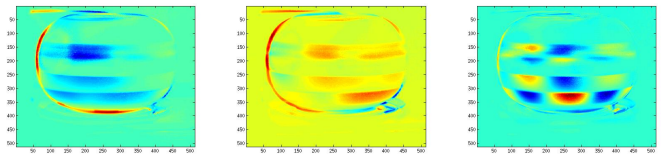


(a) Red Filter (b) Green Filter (c) Blue Filter (d) Grayscale

Figure 5.8: Projections into $\mathbb{R}^3$ of Punch Bowl

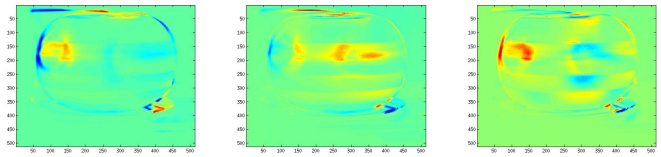
We note here that the most well-behaved curves belong to the volleyball. We conjecture that this is due to the fact that the volleyball has more features to distinguish one image from another. In contrast, the red punch bowl looks very similar from every angle and its corresponding projection into $\mathbb{R}^3$ appears to sample a continuous but not differentiable curve. We also note that in each projection, the points appear to lie on a non-self-intersecting closed loop. Therefore, these experiments give support to the conjecture that this method of projection of a data set consisting of at least one full rotation of an object does indeed result in a curve homeomorphic to S^1.

We also present in Figures 5.9 through 5.20 the eigenpictures for each object in each filter. That is, we have used the inverse of the flattening map to unflatten each column of the projection matrix $\hat{U}$. These give some insight into the most important features of the images with which to differentiate one perspective of an object from another. Note that we do not use the same color scale for each picture; the colors used to represent pixel values have been scaled to display variation within each picture. In Figures 5.21 through 5.23, we display the eigenpictures corresponding to columns one through 14 of the matrix U. Only eigenpictures two through four are used in the projection to $\mathbb{R}^3$, but considering more eigenpictures allows one to see the way in which Principal Component Analysis creates an ordered basis. Columns which appear early in U contain coarse information about the data, whereas ones which appear later pick up higher frequency information. Observe in Figure 5.24 that eigenpictures 1000 through 1002 contain very high frequency information. A careful examination of these eigenpictures reveals that they are not entirely random noise but instead contribute to the fine detail components of images in the original set.



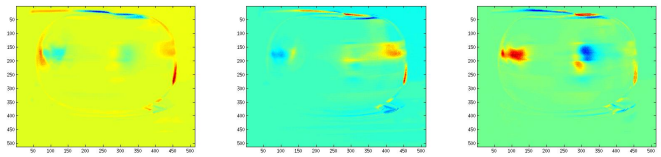
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.9: Eigenpictures: Jack o' Lantern in Red Filter



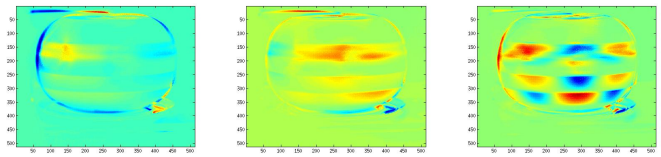
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.10: Eigenpictures: Jack o' Lantern in Green Filter



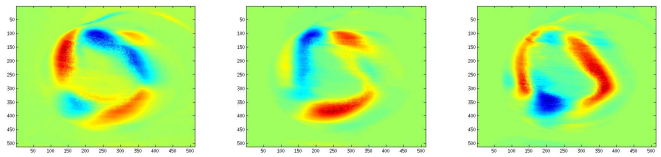
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.11: Eigenpictures: Jack o' Lantern in Blue Filter



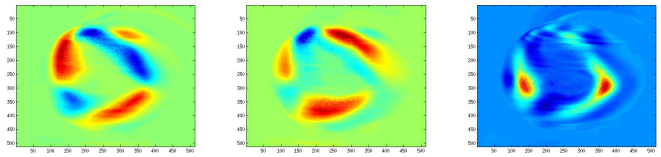
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.12: Eigenpictures: Jack o' Lantern in Gray Filter



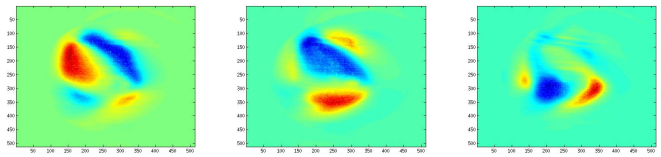
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.13: Eigenpictures: Volleyball in Red Filter



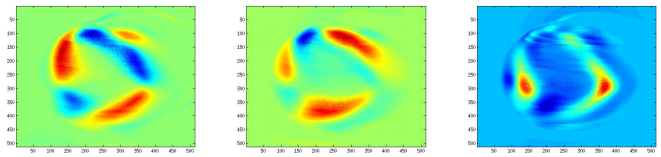
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.14: Eigenpictures: Volleyball in Green Filter



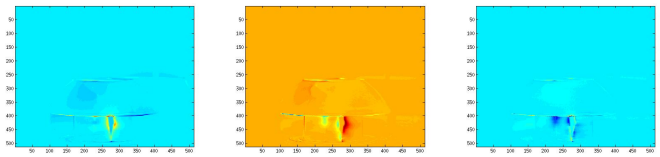
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.15: Eigenpictures: Volleyball in Blue Filter



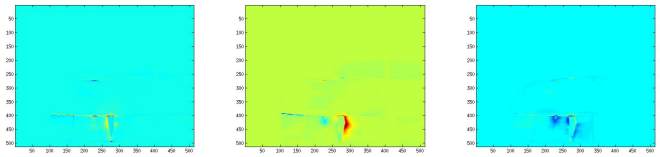
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.16: Eigenpictures: Volleyball in Gray Filter



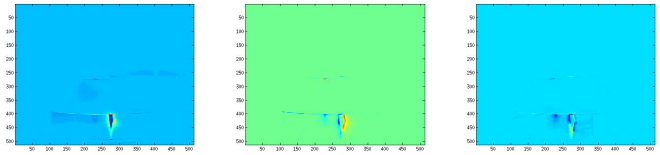
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.17: Eigenpictures: Punch Bowl in Red Filter



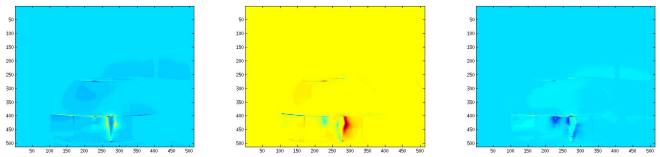
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.18: Eigenpictures: Punch Bowl in Green Filter



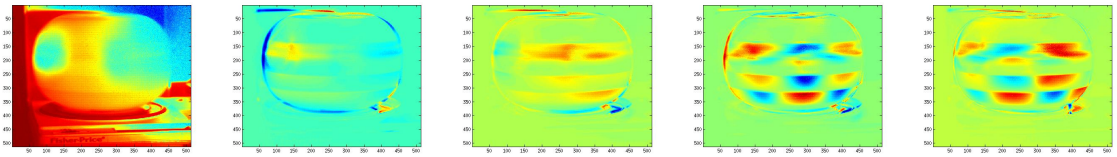
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.19: Eigenpictures: Punch Bowl in Blue Filter



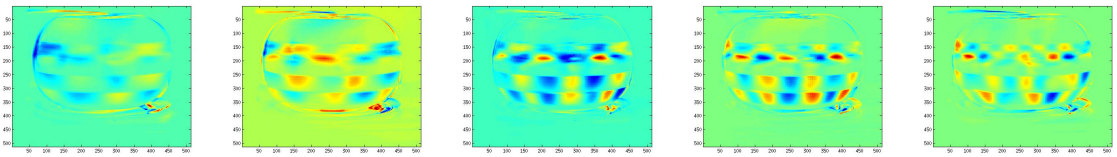
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.20: Eigenpictures: Punch Bowl in Gray Filter



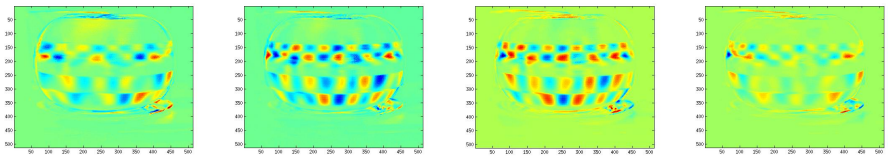
(a) Eigenimage 1 (mean) (b) Eigenimage 2 (c) Eigenimage 3 (d) Eigenimage 4 (e) Eigenimage 5

Figure 5.21: Eigenpictures: Jack o' Lantern in Gray Filter



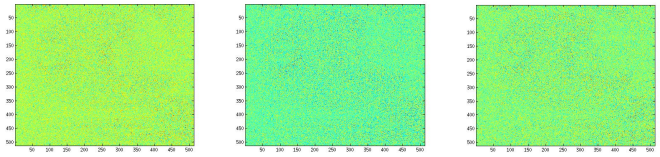
(a) Eigenimage 6 (b) Eigenimage 7 (c) Eigenimage 8 (d) Eigenimage 9 (e) Eigenimage 10

Figure 5.22: Eigenpictures: Jack o' Lantern in Gray Filter



(a) Eigenimage 11 (b) Eigenimage 12 (c) Eigenimage 13 (d) Eigenimage 14

Figure 5.23: Eigenpictures: Jack o' Lantern in Gray Filter



(a) Eigenimage 1000 (b) Eigenimage 1001 (c) Eigenimage 1002

Figure 5.24: Eigenpictures: Jack o' Lantern in Gray Filter

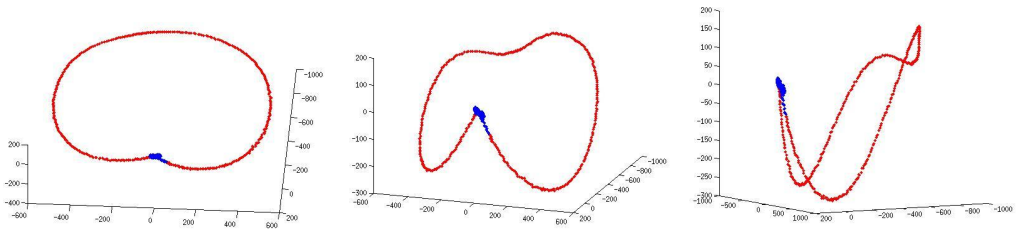
In Figures 5.25 to 5.36, we have projections of the red punch bowl with varying numbers of pieces of white tape. These data sets are collected at a speed of 1000 pictures per second and a resolution of 256×256. Enough pictures are used to capture approximately one rotation of the object. The light source is powered by a direct current, as before.

Figures 5.25 to 5.27 show projections of images of a red punch bowl with one piece of white tape attached. The points colored red represent images in which some portion of the tape is visible. We see that the blue points, corresponding to images in which no tape is visible, all project to a small region in $\mathbb{R}^3$. This is expected since each of the images in which only red punch bowl can be seen is relatively indistinguishable from other images in which only red punch bowl can be seen.

Figures 5.28 to 5.30 display projections of a red punch bowl with pieces of tape on opposite sides of the bowl. One one side is a horizontal piece of tape, and on the other is two pieces of tape that form an $\times$. The points colored red correspond to images in which some portion of the horizontal piece of tape is visible, while those colored green correspond to the $\times$ tape region. Notice that the projections of the two different shapes of tape trace out paths in different regions of $\mathbb{R}^3$. Therefore, classification methods using PCA should be capable of distinguishing between the two pieces of tape.

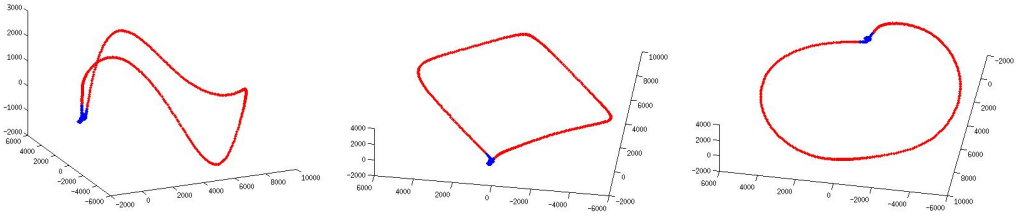
Figures 5.31 to 5.33 show projections of the data set of the red punch bowl with two vertical pieces of tape, one horizontal piece, and one $\times$. The points colored red correspond to images in which the horizontal tape is visible, the green correspond to the $\times$ tape, and the cyan and magenta correspond to different vertical stripes. We see a similar phenomenon here as with the single piece of white tape data, in that all of the points where no tape is visible project to the same region. This example shows that our conjecture about each data set projecting to a curve homeomorphic to S^1 is false. This set appears to project to four curves connected at a point, each of which is homeomorphic to S^1.

Figures 5.34 to 5.36 are projections of the data set with two pairs of similar pieces of tape, one pair being vertical strips, and one pair being horizontal strips. The vertical strips are opposite each other, as are the horizontal strips. The projection is similar to the other set with four pieces of tape, except that now we see two pairs of curves homeomorphic to S^1 connected at a point. Notice that again the curves corresponding to different shapes of tape largely occupy separate regions of space. These results suggest that shape recognition algorithms based on PCA are likely to be successful, but perspective recognition algorithms are not.



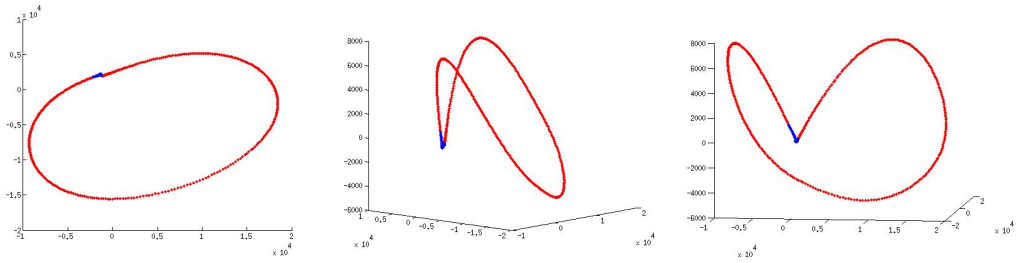
(a) View 1 (b) View 2 (c) View 3

Figure 5.25: Red Punch Bowl, One Tape, Red Filter



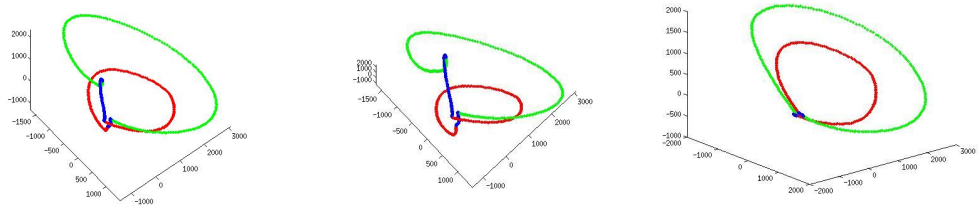
(a) View 1 (b) View 2 (c) View 3

Figure 5.26: Red Punch Bowl, One Tape, Green Filter



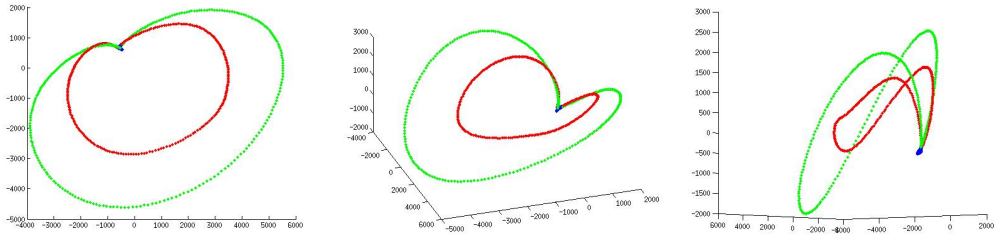
(a) View 1 (b) View 2 (c) View 3

Figure 5.27: Red Punch Bowl, One Tape, Blue Filter



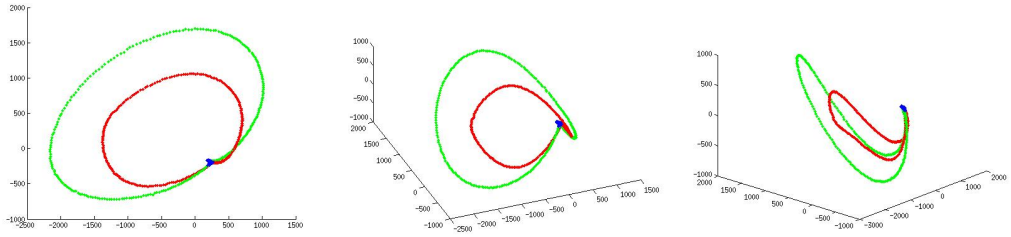
(a) View 1 (b) View 2 (c) View 3

Figure 5.28: Red Punch Bowl, Two Different Tapes, Red Filter



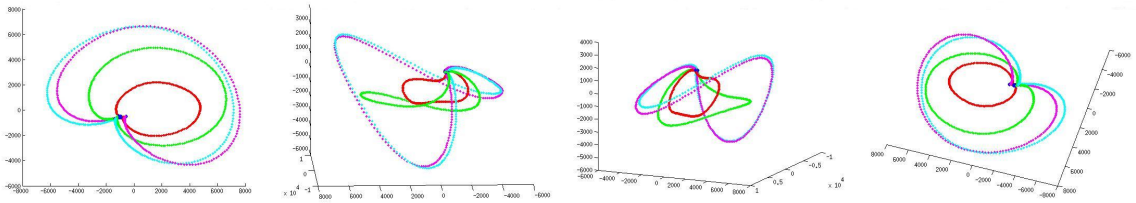
(a) View 1 (b) View 2 (c) View 3

Figure 5.29: Red Punch Bowl, Two Different Tapes, Green Filter



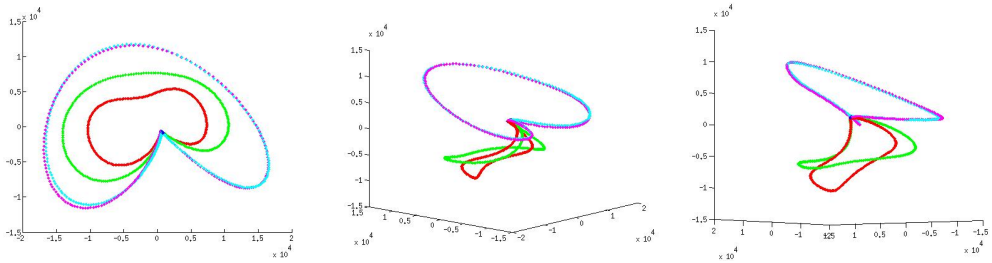
(a) View 1 (b) View 2 (c) View 3

Figure 5.30: Red Punch Bowl, Two Different Tapes, Blue Filter



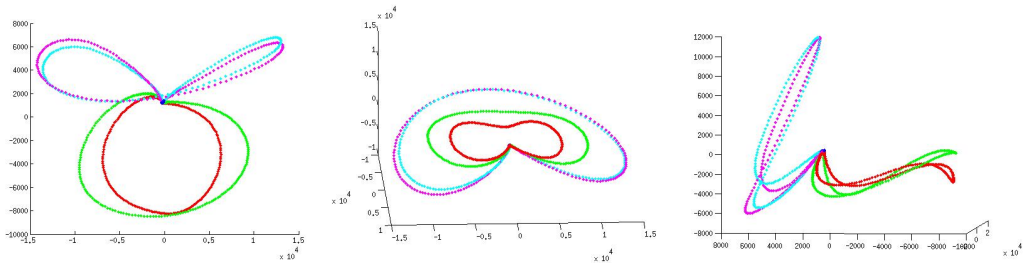
(a) View 1 (b) View 2 (c) View 3 (d) View 4

Figure 5.31: Red Punch Bowl, 1 Pair and 2 Different Tapes, Red Filter



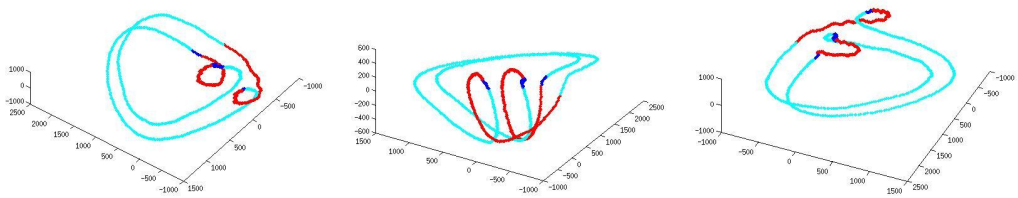
(a) View 1 (b) View 2 (c) View 3

Figure 5.32: Red Punch Bowl, 1 Pair and 2 Different Tapes, Green Filter



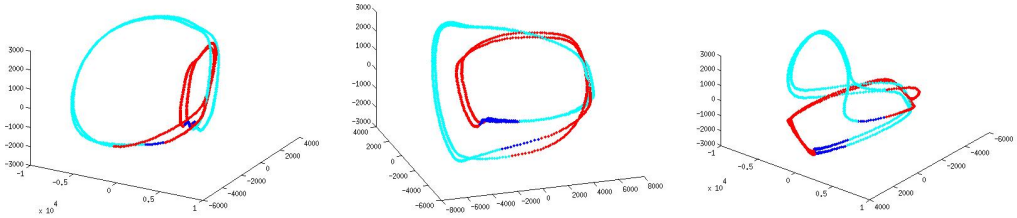
(a) View 1 (b) View 2 (c) View 3

Figure 5.33: Red Punch Bowl, 1 Pair and 2 Different Tapes, Blue Filter



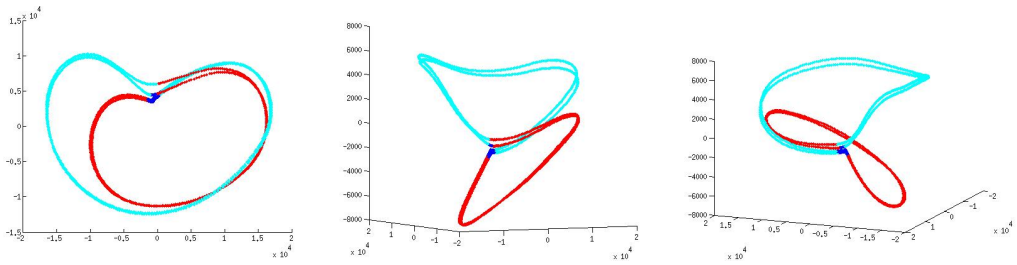
(a) View 1 (b) View 2 (c) View 3

Figure 5.34: Red Punch Bowl, Two Pairs of Tape, Red Filter



(a) View 1 (b) View 2 (c) View 3

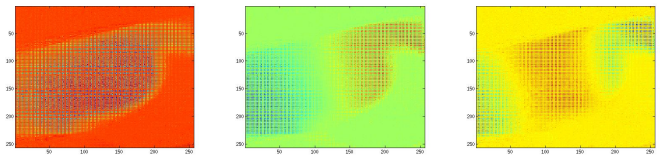
Figure 5.35: Red Punch Bowl, Two Pairs of Tape, Green Filter



(a) View 1 (b) View 2 (c) View 3

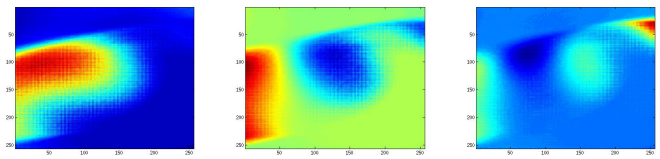
Figure 5.36: Red Punch Bowl, Two Pairs of Tape, Blue Filter

In Figures 5.37 to 5.48, we see the eigenimages used for each of these projections in each filter. Note that images are not normalized for intensity. Therefore eigenimages are biased to represent perspectives with higher intensity images, such as where tape is visible.



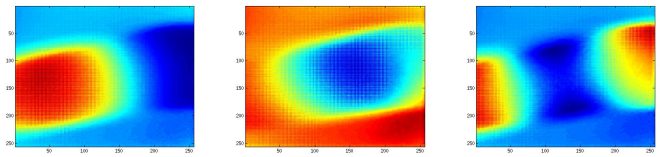
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.37: Eigenpictures: Punch Bowl, 1 Tape in Red Filter



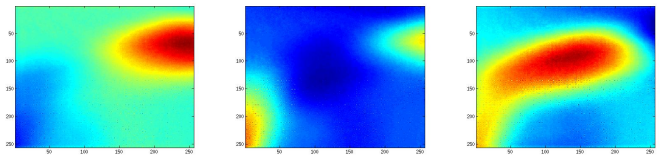
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.38: Eigenpictures: Punch Bowl, 1 Tape in Green Filter



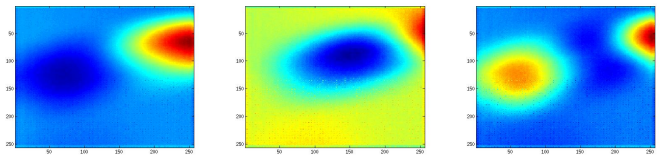
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.39: Eigenpictures: Punch Bowl, 1 Tape in Blue Filter



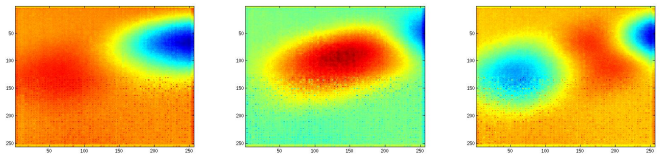
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.40: Eigenpictures: Punch Bowl, 2 Tapes in Red Filter



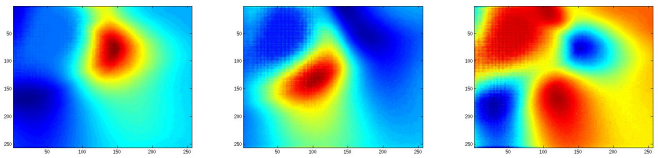
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.41: Eigenpictures: Punch Bowl, 2 Tapes in Green Filter



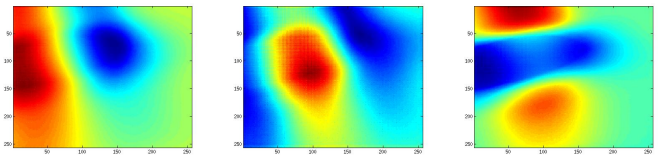
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.42: Eigenpictures: Punch Bowl, 2 Tapes in Blue Filter



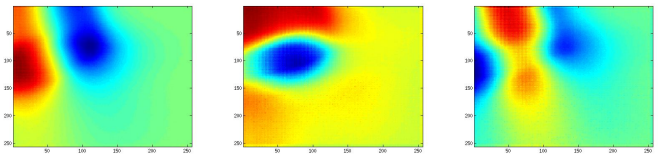
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.43: Eigenpictures: Punch Bowl, 1 Pair and 2 Different Tapes in Red Filter



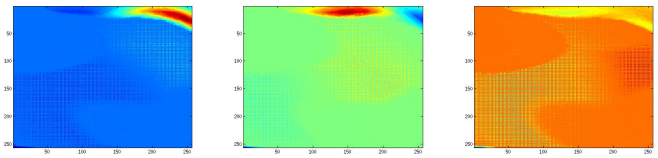
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.44: Eigenpictures: Punch Bowl, 1 Pair and 2 Different Tapes in Green Filter



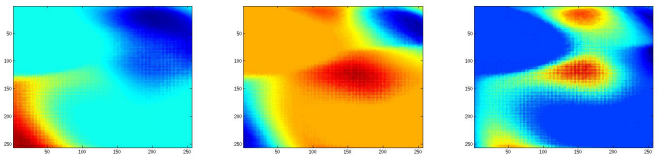
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.45: Eigenpictures: Punch Bowl, 1 Pair and 2 Different Tapes in Blue Filter



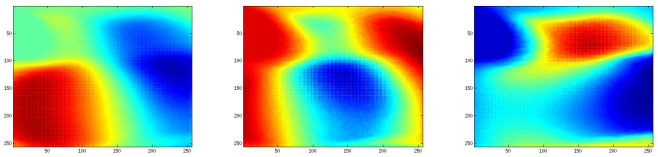
(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.46: Eigenpictures: Punch Bowl, 2 Pairs of Tape in Red Filter



(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.47: Eigenpictures: Punch Bowl, 2 Pairs of Tape in Green Filter



(a) Eigenimage 1 (b) Eigenimage 2 (c) Eigenimage 3

Figure 5.48: Eigenpictures: Punch Bowl, 2 Pairs of Tape in Blue Filter

In view of the projections of objects with symmetry into $\mathbb{R}^3$, let us discuss the more general situation. Suppose that an object looks exactly the same from two different points of view. Then the images corresponding to those two perspectives occupy the same point in pixel space, $\mathbb{R}^n$. Therefore, the projection of those two points to $\mathbb{R}^3$ will be the same because a projection is a legitimate function and cannot have two different outputs for the same input.

Now suppose that an object has a sequence of images that is identical to a sequence of images from a different perspective. Suppose also that the first sequence projects to the path α. Then we must have that the second sequence also projects to α. The projection is a 2-to-1 map above α. It follows that if α is a loop, then the image of the projection consists of at least two loops. If noise is present in the data, we will see two distinct but spatially close loops. If there is no noise, we will see the same loop traversed twice.

Because the real data that we collect has noise, there are no pictures that are exactly alike, but many are similar. Therefore, we see that sometimes a great number of pictures get projected to almost the same point in space. For example, with the data set of a punch bowl with one piece of white tape, approximately 1250 out of 1722 pictures are mapped to a relatively small region: if the mean of the 1250 points is m, then all 1250 points p have the property that the norm of $|p - m|$ is less than 40.59, whereas $|p - m|$ is as large as 994 for points corresponding to locations where tape is visible.

The noise in the data and the slight variation in the objects themselves also allows us to see distinct paths corresponding to sequences of pictures that appear to be the same. For example, the projections of the two different vertical strips of tape are two loops that are nearly identical but are nevertheless distinct.

Chapter 6

An Algorithm for Determining Optimal Camera Location Distribution

6.1 Introduction

In this chapter, we develop and apply an algorithm which finds minimal energy point configurations on a vector bundle over S^1 using a distance measure on the Grassmannian. Consider the following problem. Place an object at the center of a circle of possible camera locations. Suppose that the number of camera locations and the resolution of the camera are fixed. At each camera location, we take a fixed number of images and find a finite dimensional vector space representation for the set. In this way, we attach a point on the Grassmannian to a finite but large number of points on the circle. One example of such a data set would be to collect images under varying illumination and attach the illumination space of the object at each point on the circle. Given this setup, we seek to determine the best n locations from which to take pictures in order to approximate the given vector bundle.

Note that depending on the convexity and symmetry of the object, the answers

vary drastically. If a perfectly symmetric object is placed at the center of the sphere, then every distribution should be optimal. If an object looks completely different from every angle, then the optimal distribution should be uniform. In contrast, if an object has little variation in features on one particular side, say side L, but has large variations in features on a different side, say side R, then the optimal camera location distribution will involve very few pictures of side L and many pictures of side R. We propose two algorithms to determine camera location distributions.

Algorithm 6.1. (Nearest Neighbor Dispersion Algorithm – Global)

Input: *An initial configuration C of n points on S^1, parameters ϵ, d, T, and k, a unitarily invariant function ∂ on the Grassmannian, and a data bundle over S^1.*

Output: *A final configuration C of n points on S^1.*

Algorithm:

- *Pick a point $x \in C$ at random.*
- *Let the vector spaces associated to the points in C be denoted $\left\{U_{x_j}\right\}_{j=1}^{n}$.*
- *For each $x_j \neq x$, compute the singular value decomposition of $U_x^H U_{x_j}$ to get the corresponding principal angles between the subspaces U_x and U_{x_j}.*
- *Compute the distance $\partial(U_x, U_{x_j})$ between U_x and U_{x_j} for each $x_j \neq x$. Denote the j which gives the smallest distance by j_{min}.*
- *Move x by a distance ϵ in such a way that its distance with $x_{j_{min}}$ increases.*

Repeat. After each k iterations, decrease ϵ by d. Complete the above steps a total of T times.

Algorithm 6.2. (Nearest Neighbor Dispersion Algorithm – Local)

Input: *An initial configuration C of n points on S^1, parameters ϵ, d, T, and k, a unitarily invariant function ∂ on the Grassmannian, and a data bundle over S^1.*

Output: *A final configuration C of n points on S^1.*

Algorithm:

- *Pick a point $x \in C$ at random.*
- *Let the vector spaces associated to the points in C be denoted $\left\{U_{x_j}\right\}_{j=1}^{n}$.*
- *Denote by x_a the point in C that achieves the smallest distance to x on S^1 on one side and by x_b the point in C that achieves the smallest distance to x on S^1 on the other side. For both $j = a$ and $j = b$, compute the singular value decomposition of $U_x^H U_{x_j}$ to get the corresponding principal angles between the subspaces U_x and U_{x_j}.*
- *Compute the distance $\partial(U_x, U_{x_j})$ between U_x and U_{x_j} for $j = a$ and $j = b$. Denote the j which gives the smallest distance by j_{min}.*
- *Move x by a distance ϵ in such a way that its distance with $x_{j_{min}}$ increases.*

Repeat. After each k iterations, decrease ϵ by d. Complete the above steps a total of T times.

The two algorithms are driven by different information and can produce very different results. In both cases, we hope that by annealing the distance by which a point moves in a given step, the camera locations will converge to a local (and possibly global) minimum of an energy function f. Measuring distances on the Grassmannian instead of on S^1 should lead to a camera location distribution which will provide a (locally) optimal n-location representation of the vector bundle associated to the object.

6.2 Implementation

6.2.1 A Koszul Complex as a Trial Data Set

We begin by constructing the Koszul complex as a trial data set on which to implement the algorithm. This setting differs from the data set of images in several

ways. First, we have an infinite set of possible locations. It is therefore possible to pack points into a region more tightly than in the real data setting. Second, it takes significantly longer for the algorithm to converge to a stable set than in the real data setting. Finally, because the points on the circle do not correspond to a perspective of an object, it is more difficult to judge the success of the algorithm. We treat this set as an initial set on which to get the algorithm working.

We use a certain Koszul complex to define a rank three vector bundle over S^1. However, it also has a natural line bundle associated to it via the kernel. It is possible to get a visual representation of the final configuration's line bundle (and hence, also of the associated rank three vector bundle) over S^1 by plotting each line as a point on S^2. We do this by plotting the intersection of each line with the upper hemisphere of S^2. We show the result of this visualization in Figure 6.1. Based on the approximately even spacing of the points, we believe the algorithm is performing well on this set. The final configuration used to generate this line bundle is shown in Figure 6.2. This configuration is the result of an initial configuration of 200 randomly spaced points. The local version of the Nearest Neighbor Dispersion Algorithm was used to obtain this configuration, using five million steps and the Fubini-Study metric.

Applications of the global version of the Nearest Neighbor Dispersion Algorithm on this set are unsuccessful. We conjecture that the global version requires a higher-dimensional ambient space in order to succeed. This is a question for future research. Because the local version of the algorithm produces good results in this initial setting and the global version does not, we choose to focus on the application of the local version of the algorithm when we work with other data sets. Henceforth, when we refer to the Nearest Neighbor Dispersion Algorithm, we implicitly mean the local version.

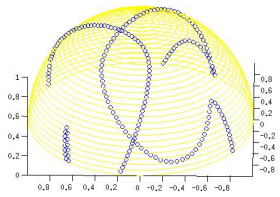


Figure 6.1: Line Bundle for Final Configuration

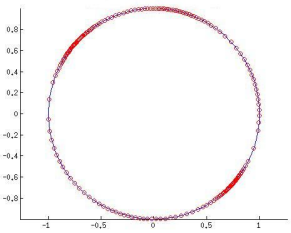


Figure 6.2: Final Configuration with 200 Points

6.2.2 Red Filter Images as a Line Bundle Data Set

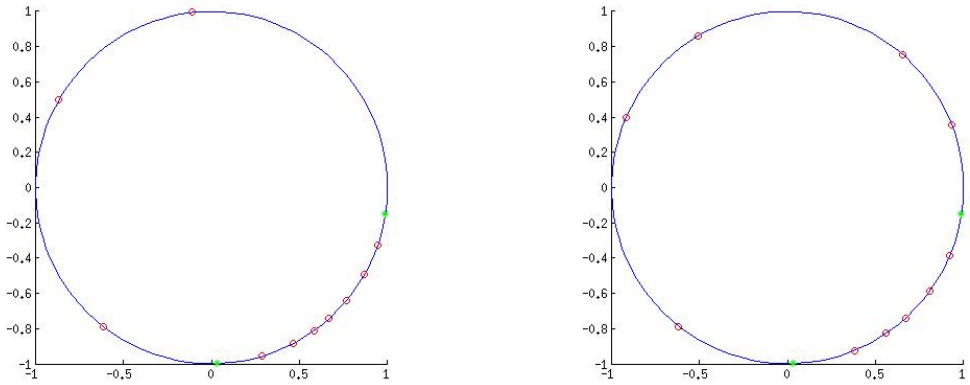
We construct three line bundle data sets over S^1 in the following way. Using the high-speed camera, we capture pictures of a red punch bowl with one or more pieces of white tape on it as it rotates on a record player. Pictures are collected at a resolution of 256×256 and at a rate of 1000 pictures per second. We use only the red filter, and we flatten each image as described in Section 5.2. Hence, at each point in our sample of S^1, we attach a vector with real entries of length $256^2 = 65,536$. In this way, we simulate a camera moving along a circle, capturing pictures of a stationary object at the center.

Because we have a finite sample of S^1, the number of camera locations that can appear in the final configuration is equal to the number of pictures needed for one rotation of the object. The object rotates at an approximate rate of $33\frac{1}{3}$ rotations per minute, so one rotation corresponds to approximately (1000 pictures per second) $\times$ (60 seconds per minute) $\times$ ($\frac{1}{33\frac{1}{3}}$ minutes per rotation) = 1800 pictures.

Our first data set consists of pictures of a red punch bowl with one horizontal piece of white tape on it. Images from the set can be seen in Figure 5.2. We experimentally determine that the following values for the parameters ϵ, d, T, and k work well: $\epsilon \approx 0.11$ radians (or 6.35°), $k \approx 5000$, and $T \approx 14$. We set the parameter d to be approximately 0.018 radians, or 1.06° for the first five iterations. For the last nine iterations, we set d to be approximately 0.004 radians, or 0.211°.

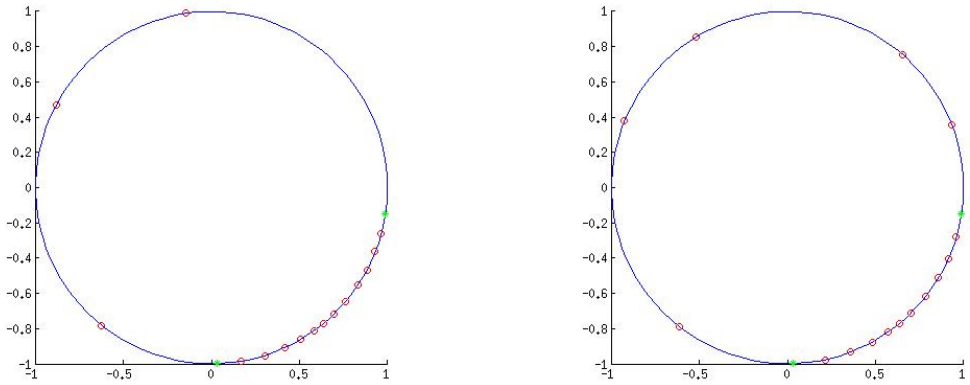
In Figures 6.3 to 6.11, we show the final configuration of camera locations after running the algorithm on a random starting configuration. We show the results from two implementations of the algorithm for each of 10, 15, 20, 25, 30, 35, 40, 45, and 50 points. The green asterisks on the circle mark the beginning and end of the set of locations in which some portion of the white tape is visible. Note that, as expected, the points tend to converge to the region where the white tape is visible. Also, the number and location of points outside the visible tape region varies. This is unsurprising because there is nothing to distinguish locations that lie outside the

visible tape region.



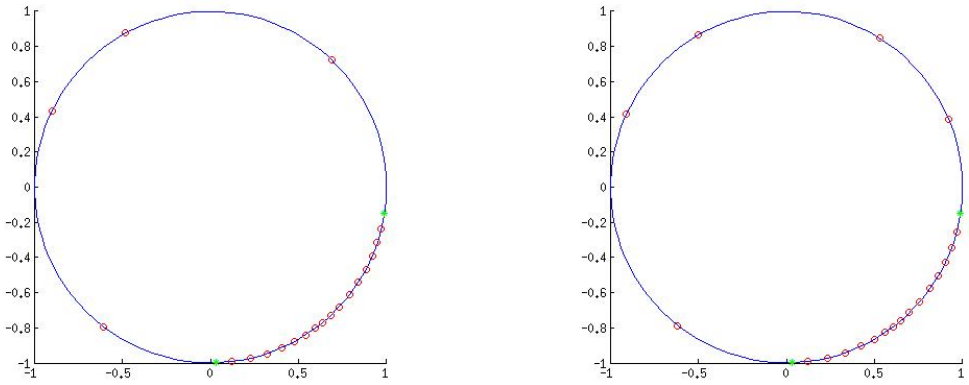
(a) Run 1 (b) Run 2

Figure 6.3: Final Configurations, 1 Tape: 10 Points



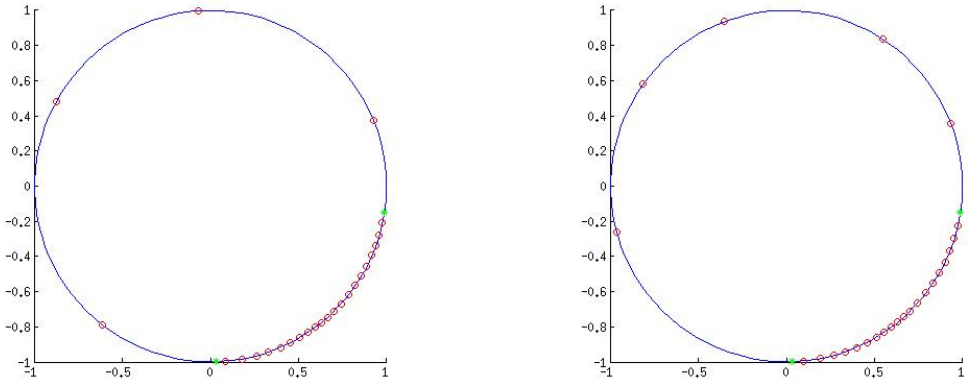
(a) Run 1 (b) Run 2

Figure 6.4: Final Configurations, 1 Tape: 15 Points



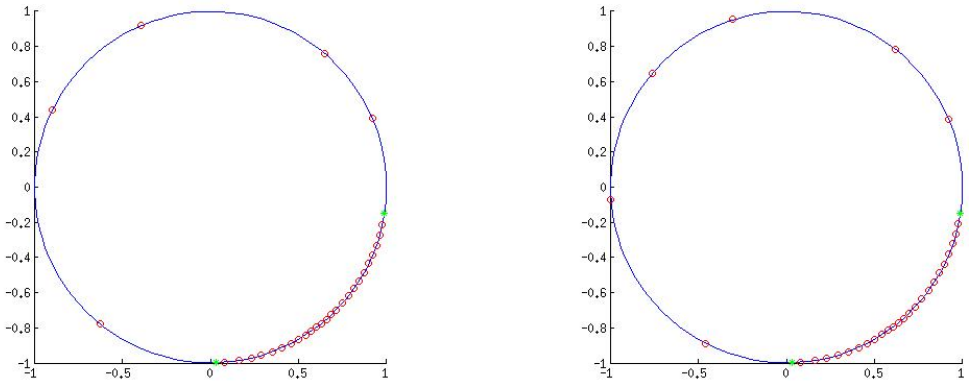
(a) Run 1 (b) Run 2

Figure 6.5: Final Configurations, 1 Tape: 20 Points



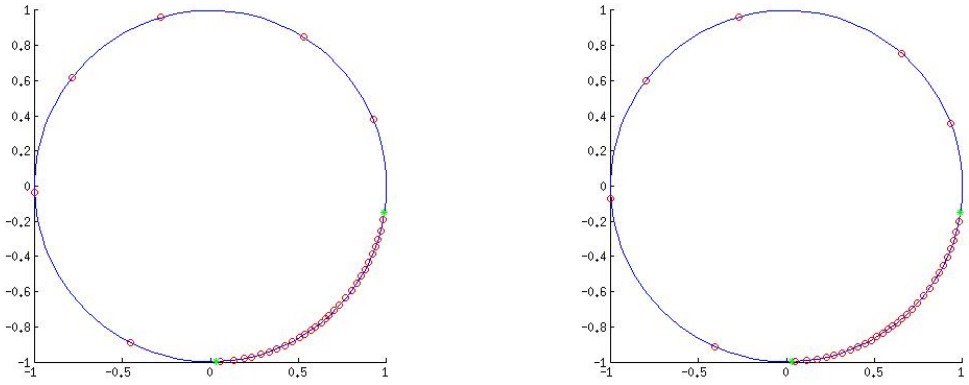
(a) Run 1 (b) Run 2

Figure 6.6: Final Configurations, 1 Tape: 25 Points



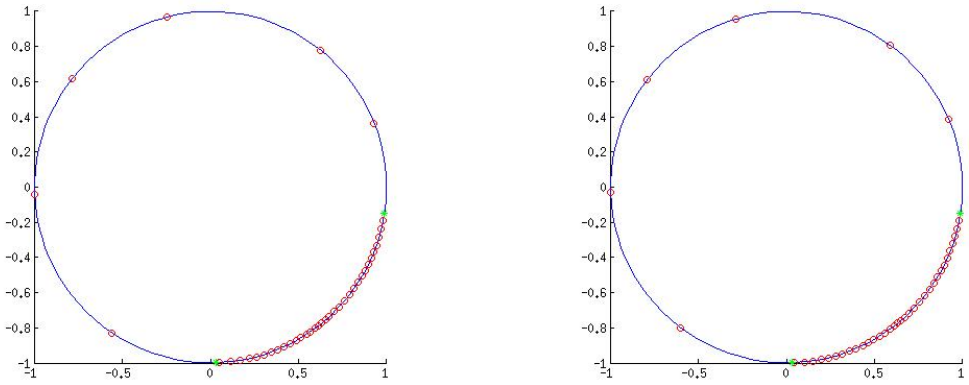
(a) Run 1 (b) Run 2

Figure 6.7: Final Configurations, 1 Tape: 30 Points



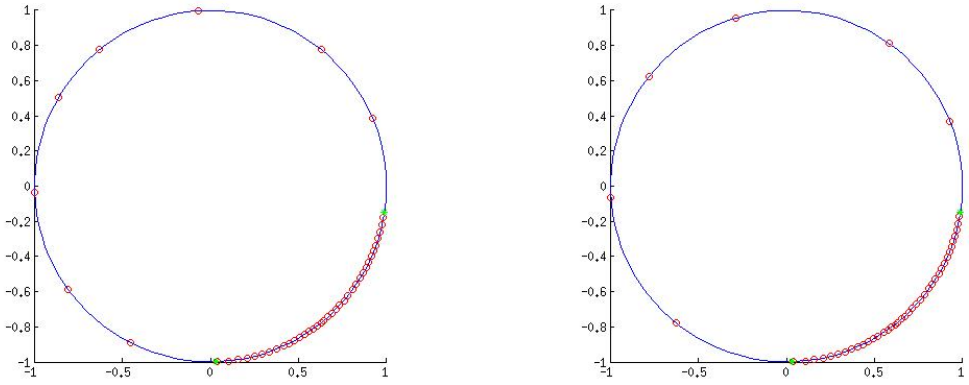
(a) Run 1 (b) Run 2

Figure 6.8: Final Configurations, 1 Tape: 35 Points



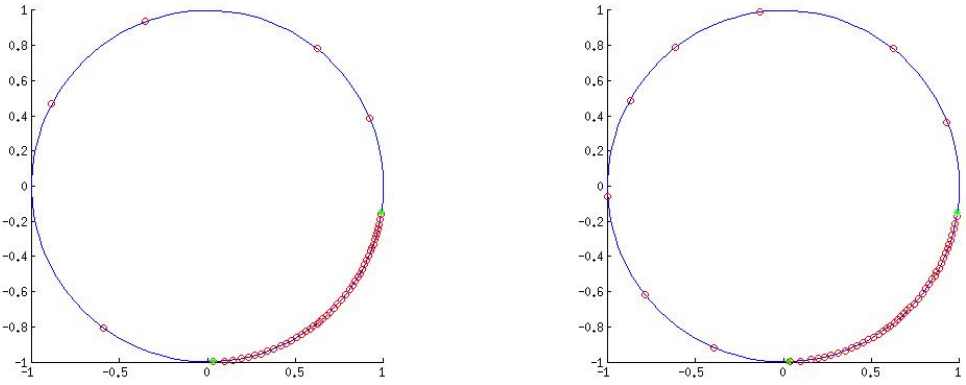
(a) Run 1 (b) Run 2

Figure 6.9: Final Configurations, 1 Tape: 40 Points



(a) Run 1 (b) Run 2

Figure 6.10: Final Configurations, 1 Tape: 45 Points



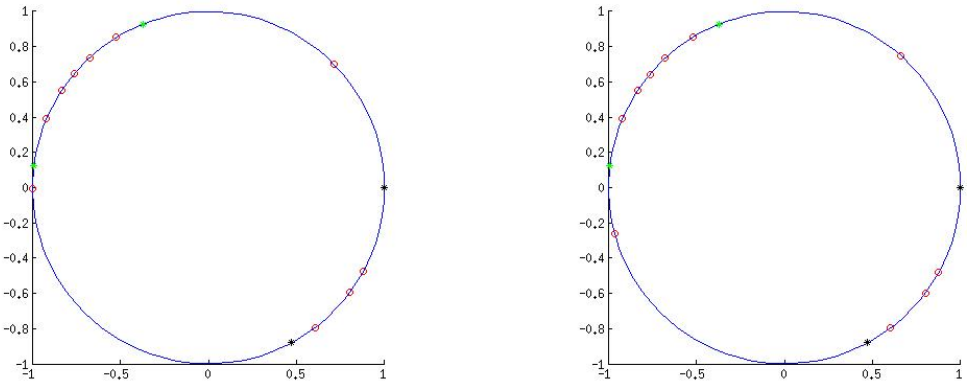
(a) Run 1

(b) Run 2

Figure 6.11: Final Configurations, 1 Tape: 50 Points

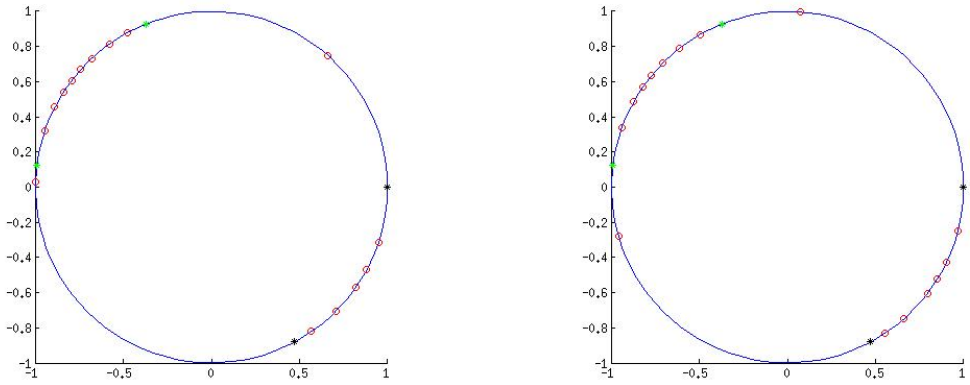
For our second data set, we use the same parameters ϵ, d, T, and k. Images of a red punch bowl with tape on two sides are collected at a rate of 1000 pictures per second and a resolution of 256×256 pixels. The tape on one side is a short horizontal piece. On the opposite side, we place two short pieces of tape to form an $\times$. Figure 5.3 shows images from the data set.

In Figures 6.12 to 6.20, we show the final configuration of camera locations after running the algorithm on a random starting configuration. The green asterisks denote the start and end of the section in which the horizontal piece of tape is visible. The black asterisks denote the start and end of the section in which the $\times$ tape is visible. Note that the points tend to converge to those two regions. As before, the number and location of points outside the visible tape regions varies with each run of the algorithm.



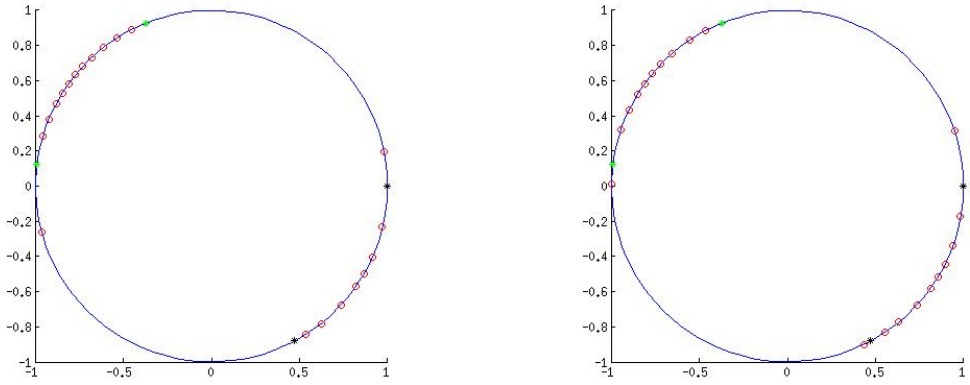
(a) Run 1 (b) Run 2

Figure 6.12: Final Configurations, 2 Tapes: 10 Points



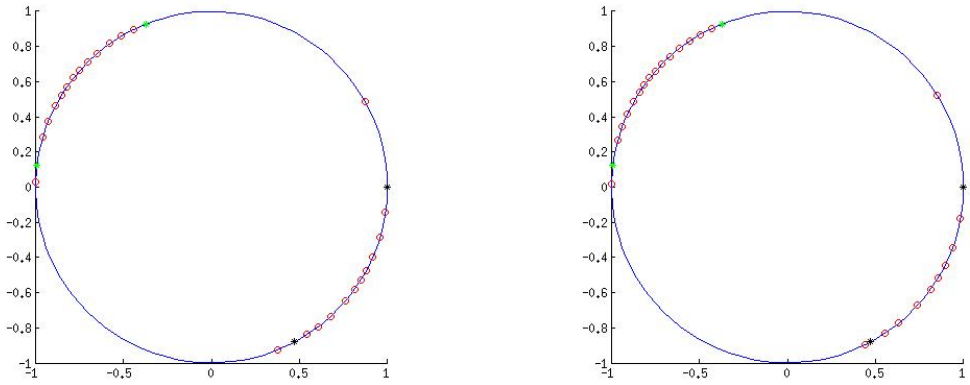
(a) Run 1 (b) Run 2

Figure 6.13: Final Configurations, 2 Tapes: 15 Points



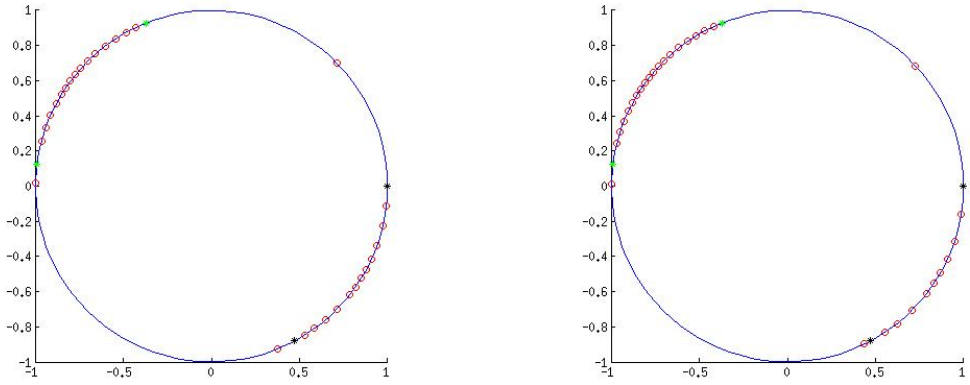
(a) Run 1 (b) Run 2

Figure 6.14: Final Configurations, 2 Tapes: 20 Points



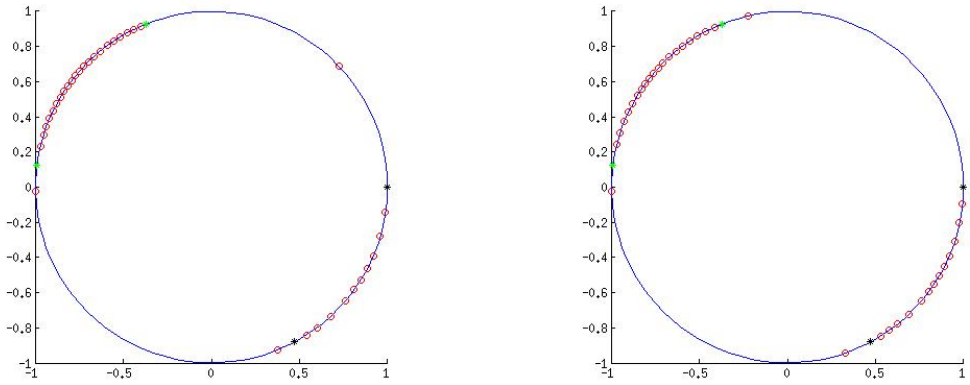
(a) Run 1 (b) Run 2

Figure 6.15: Final Configurations, 2 Tapes: 25 Points



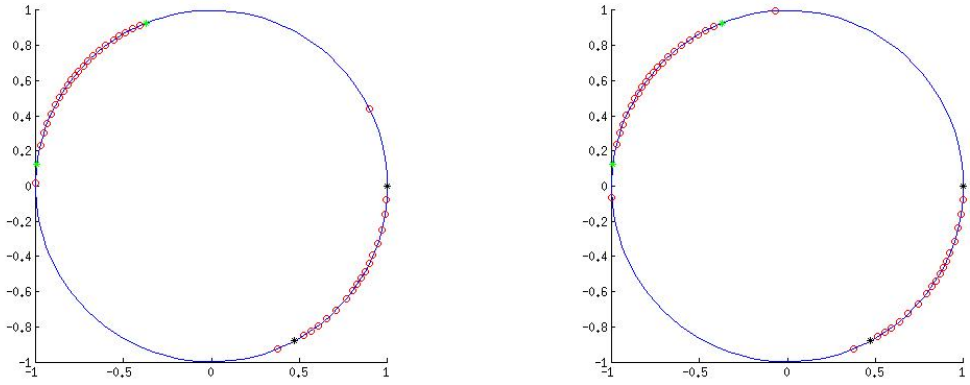
(a) Run 1 (b) Run 2

Figure 6.16: Final Configurations, 2 Tapes: 30 Points



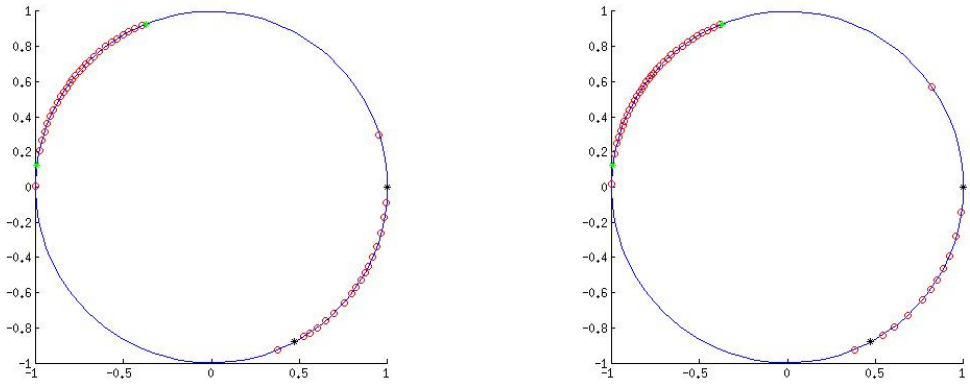
(a) Run 1 (b) Run 2

Figure 6.17: Final Configurations, 2 Tapes: 35 Points



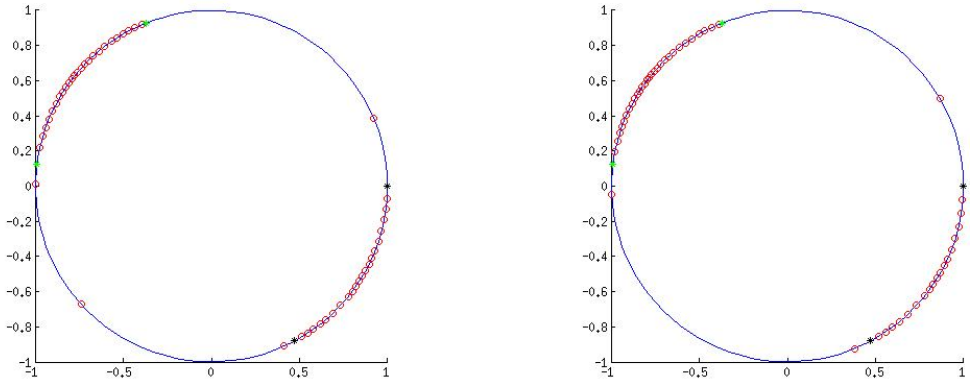
(a) Run 1 (b) Run 2

Figure 6.18: Final Configurations, 2 Tapes: 40 Points



(a) Run 1 (b) Run 2

Figure 6.19: Final Configurations, 2 Tapes: 45 Points

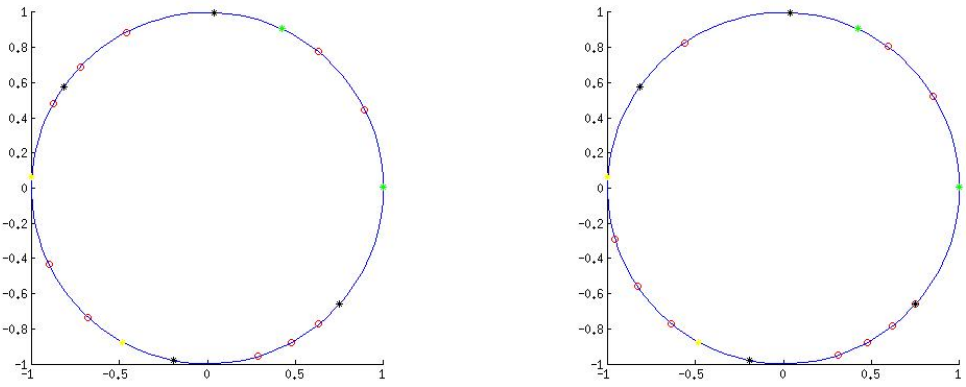


(a) Run 1 (b) Run 2

Figure 6.20: Final Configurations, 2 Tapes: 50 Points

For our next data set, we use the same parameters ϵ, d, T, and k. Images of a red punch bowl with tape on four locations are collected at a rate of 1000 pictures per second and a resolution of 256×256. The tape on one side is a short horizontal piece. On the opposite side, we place two short pieces of tape to form an $\times$. On the two sides in between the horizontal and $\times$ tape, we place a short vertical piece of tape. Images from this data set can be seen in Figure 5.4.

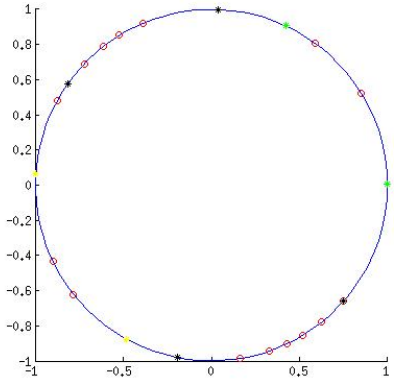
In Figures 6.21 to 6.29, we show the final configuration of camera locations after running the algorithm on a random starting configuration. The green asterisks denote the start and end of the section in which the horizontal piece of tape is visible. The black asterisks denote the start and end of the two sections in which the vertical tape is visible. The yellow asterisks denote the start and end of the section in which the $\times$ tape is visible. Note that the points tend to converge to those four regions. The regions with the vertical tape seem to attract more points than the others.



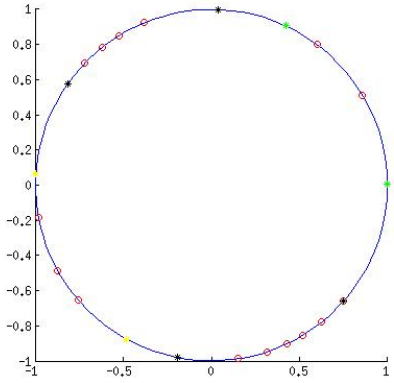
(a) Run 1

(b) Run 2

Figure 6.21: Final Configurations, 2 Horiz., 1 Vert., 1 $\times$ Tape: 10 Points

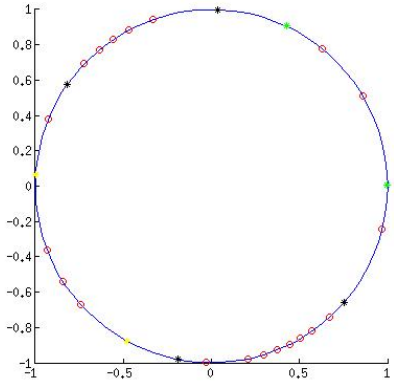


(a) Run 1

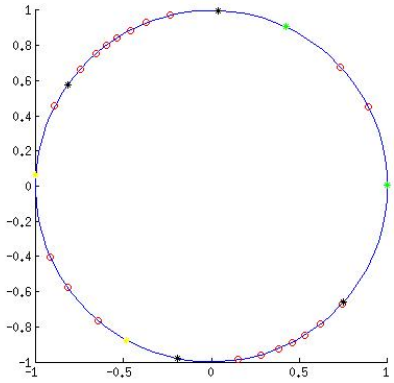


(b) Run 2

Figure 6.22: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 15 Points

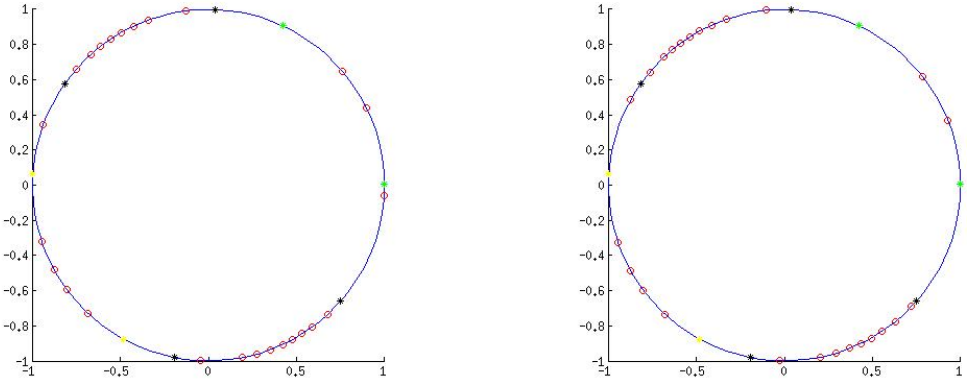


(a) Run 1



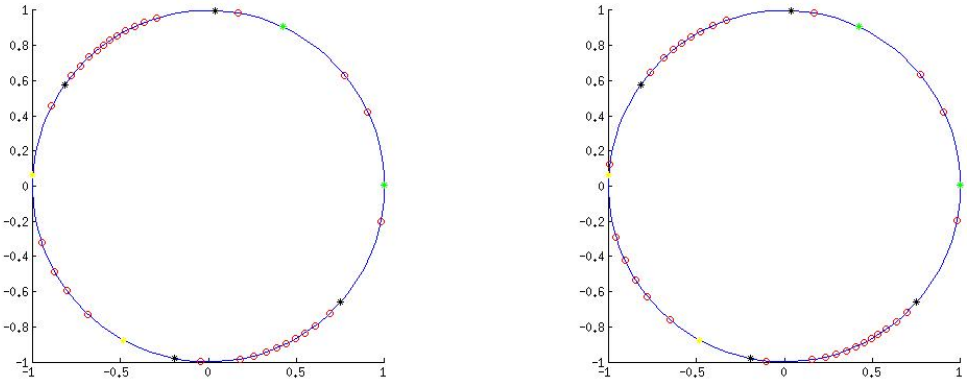
(b) Run 2

Figure 6.23: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 20 Points



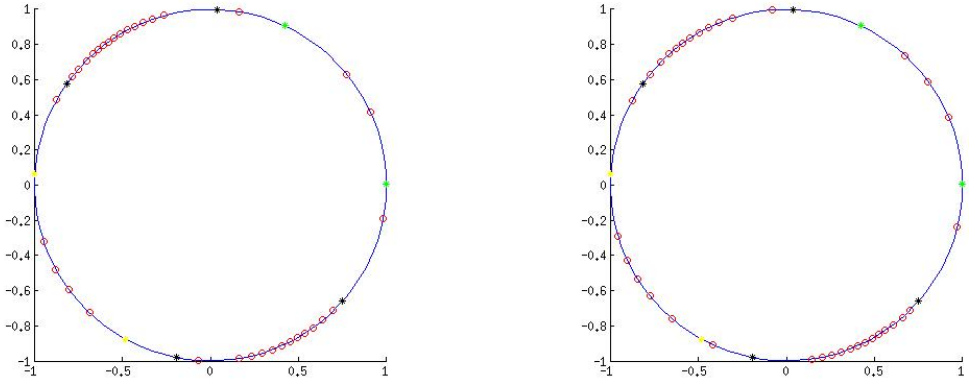
(a) Run 1 (b) Run 2

Figure 6.24: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 25 Points



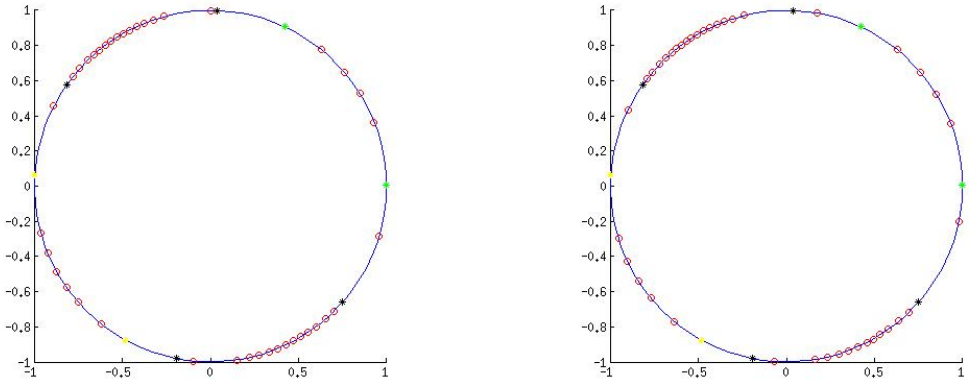
(a) Run 1 (b) Run 2

Figure 6.25: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 30 Points



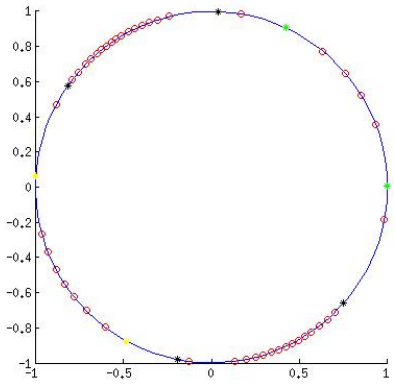
(a) Run 1 (b) Run 2

Figure 6.26: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 35 Points

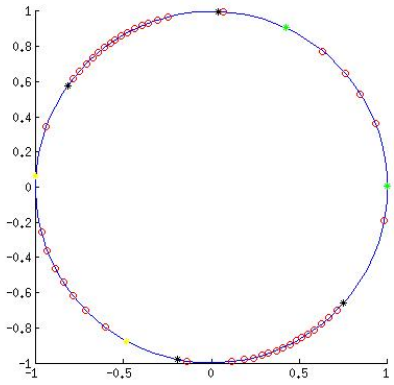


(a) Run 1 (b) Run 2

Figure 6.27: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 40 Points

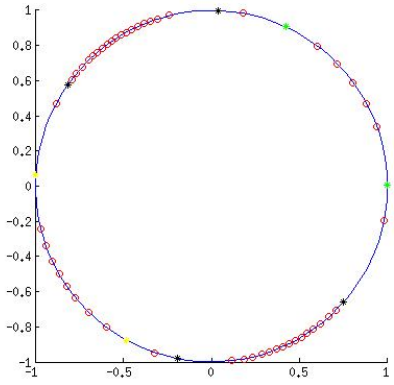


(a) Run 1

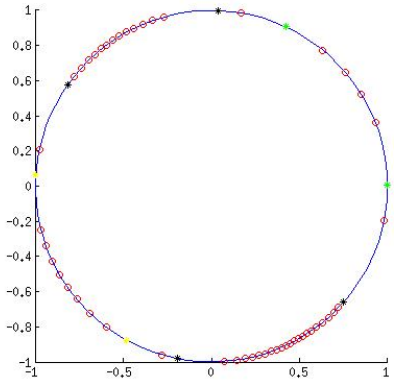


(b) Run 2

Figure 6.28: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 45 Points



(a) Run 1



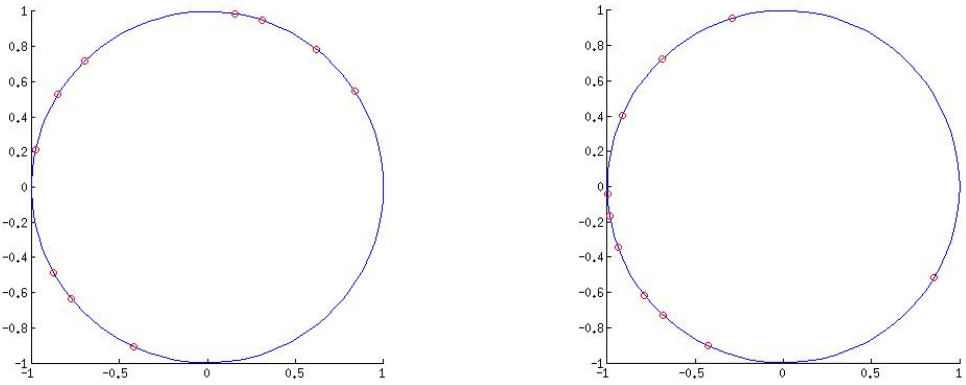
(b) Run 2

Figure 6.29: Final Configurations, 2 Horiz., 1 Vert., 1 × Tape: 50 Points

6.2.3 Red, Green, and Blue Filter Images as an Approximation of an Illumination Space Data Set

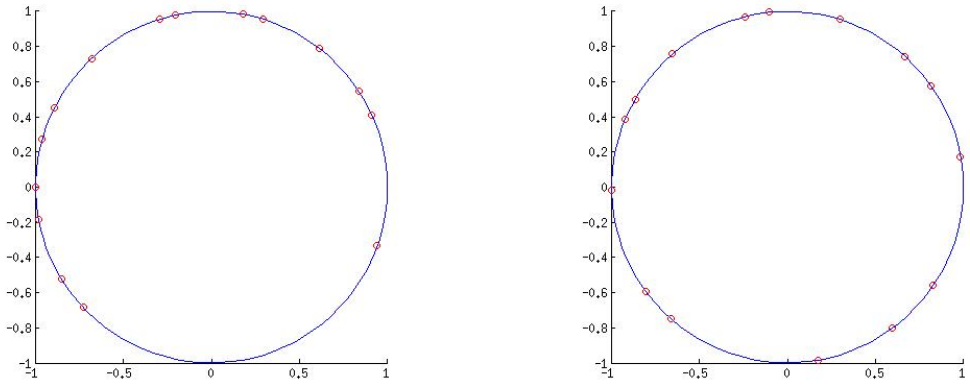
We create a data set of images of a red, white, and blue volleyball on the record player. With this data set, we use all three color filters. The three-dimensional vector space that we attach at each point can be seen as an approximation of the illumination space at that point. In this way, we sample a data set with a natural structure as a rank three vector bundle that is a subbundle of the trivial bundle over S^1.

Images are cropped to have a resolution of 256×256. Only a portion of the volleyball is visible in each image; no other background objects can be seen. Images are captured at a rate of approximately 220 pictures per second. Our data set of pictures representing one rotation of the volleyball has 425 pictures. Because each view of the volleyball has something to distinguish it, we expect that the points will converge to an approximately uniform distribution. Figures 6.30 to 6.32 show the final configurations for 2 runs of the algorithm on each of 10, 15, and 20 points using the Fubini-Study metric. The points are spread out somewhat evenly.



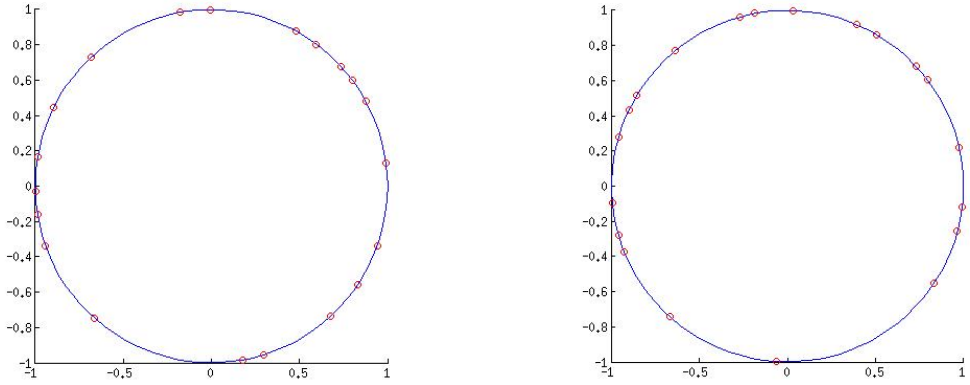
(a) Run 1 (b) Run 2

Figure 6.30: Final Configurations, Volleyball: 10 Points



(a) Run 1 (b) Run 2

Figure 6.31: Final Configurations, Volleyball: 15 Points



(a) Run 1 (b) Run 2

Figure 6.32: Final Configurations, Volleyball: 20 Points

6.3 Distance Functions

For a given configuration C on S^1, and for a given choice of distance funtion on the Grassmannian, we may gain some understanding of the data set by graphing the distance function based at each point of C. What we mean by this is the following: Fix a point $x \in C$. Let $Y \subset S^1$ be the finite set of possible camera locations. For each $y \in Y$, we calculate $\partial(x, y)$, for a chosen unitarily invariant norm, ∂, on the Grassmannian. Hence, for a fixed x, we have a function of y, which describes the closeness of x to each other point y in C. We compute this evaluation of the distance function based at the point x for each $x \in C$.

In Figures 6.34 to 6.43, we see the figures corresponding to the distance function based at each point of a 50 point configuration. This configuration, shown in Figure 6.33, is a final configuration after running the Nearest Neighbor Dispersion Algorithm on the data set of images of the red punch bowl with one piece of white tape. For an introduction to this data set, see Section 5.2. Points one through five are the only points in the region of S^1 in which no portion of the tape is visible. Notice that the figures corresponding to these five points have remarkably different distance function graphs than those of the other 45 points. For each base point x in the first five points of the configuration, x has zero distance with itself but has a minimum distance of 0.00820 ± 0.00015 with every other sample point on S^1. This suggests that there is negligible distance between each pair of points in the region with no visible tape. In contrast, we see a distance function that appears to be an approximation of a continuous function for all points in the region in which some tape is visible. This contrast suggests that such distance functions could be used for feature recognition.

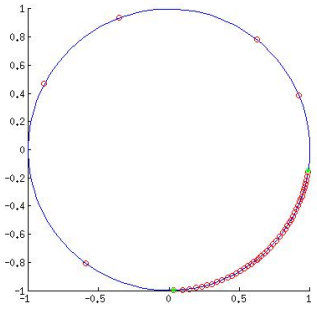
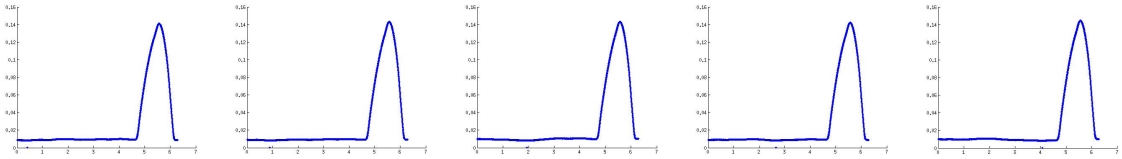
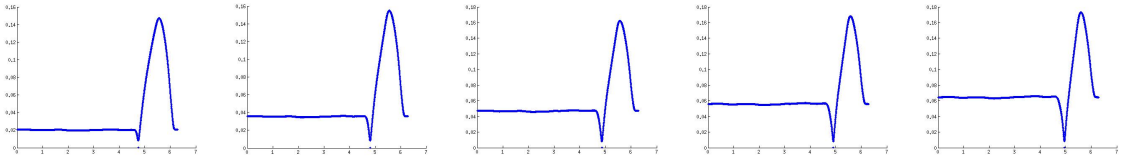


Figure 6.33: Final Configuration, 1 Tape: 50 Points



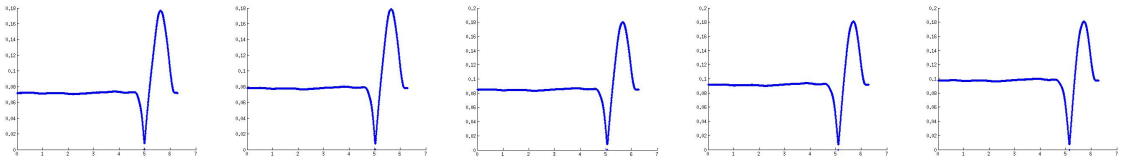
(a) Point 1 (b) Point 2 (c) Point 3 (d) Point 4 (e) Point 5

Figure 6.34: 1 Tape: Distance Function Based at Points 1-5



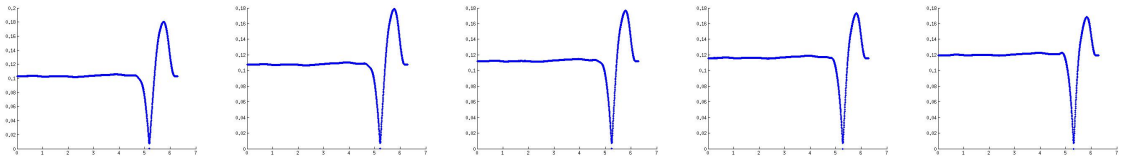
(a) Point 6 (b) Point 7 (c) Point 8 (d) Point 9 (e) Point 10

Figure 6.35: 1 Tape: Distance Function Based at Points 6-10



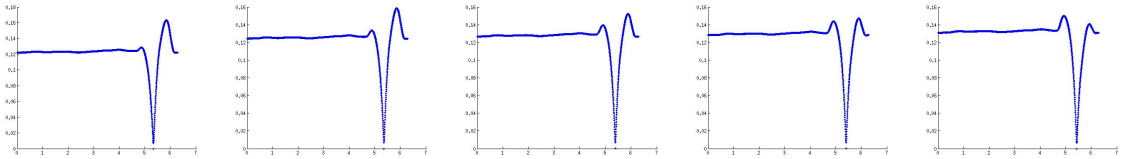
(a) Point 11 (b) Point 12 (c) Point 13 (d) Point 14 (e) Point 15

Figure 6.36: 1 Tape: Distance Function Based at Points 11-15



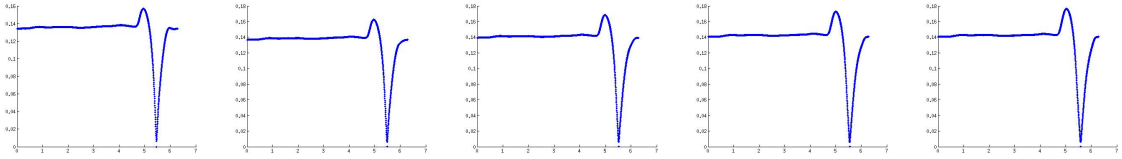
(a) Point 16 (b) Point 17 (c) Point 18 (d) Point 19 (e) Point 20

Figure 6.37: 1 Tape: Distance Function Based at Points 16-20



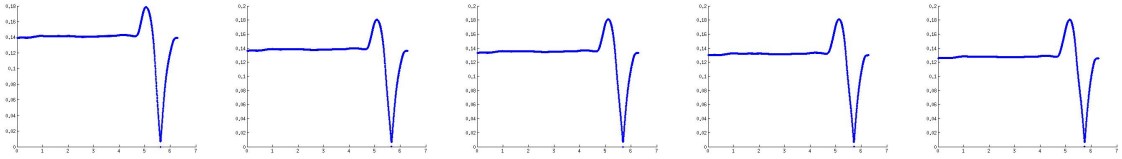
(a) Point 21 (b) Point 22 (c) Point 23 (d) Point 24 (e) Point 25

Figure 6.38: 1 Tape: Distance Function Based at Points 21-25



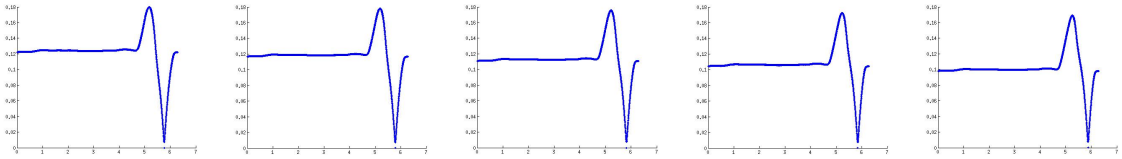
(a) Point 26 (b) Point 27 (c) Point 28 (d) Point 29 (e) Point 30

Figure 6.39: 1 Tape: Distance Function Based at Points 26-30



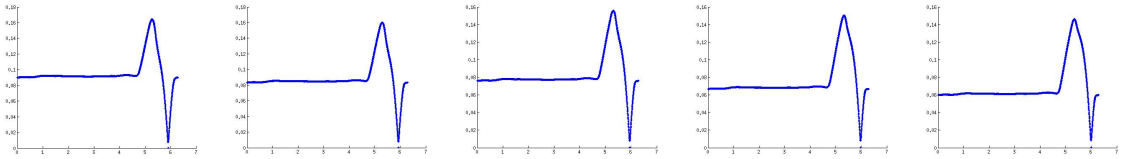
(a) Point 31 (b) Point 32 (c) Point 33 (d) Point 34 (e) Point 35

Figure 6.40: 1 Tape: Distance Function Based at Points 31-35



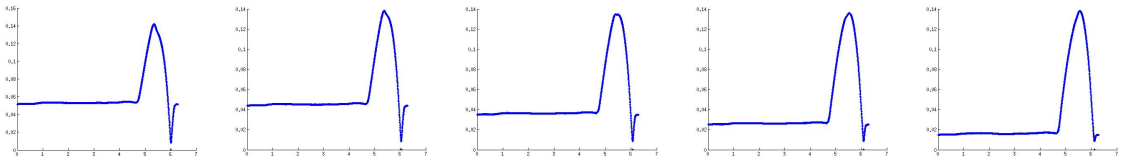
(a) Point 36 (b) Point 37 (c) Point 38 (d) Point 39 (e) Point 40

Figure 6.41: 1 Tape: Distance Function Based at Points 36-40



(a) Point 41 (b) Point 42 (c) Point 43 (d) Point 44 (e) Point 45

Figure 6.42: 1 Tape: Distance Function Based at Points 41-45



(a) Point 46 (b) Point 47 (c) Point 48 (d) Point 49 (e) Point 50

Figure 6.43: 1 Tape: Distance Function Based at Points 46-50

The figures with the results of the same experiment for the data set of the punch bowl with two pieces of tape are shown in Figures 6.45 to 6.54, with the final configuration shown in Figure 6.44. Figures 6.56 to 6.65 and Figure 6.55 show the results for the data set of the punch bowl with four pieces of tape (two vertical strips, one horizontal, and one $\times$). For introductions to these two data sets, see Section 5.2. In both data sets, we see similar behavior, in that the distance function based at a point in a region with tape visible appears to be continuous, and when no tape is visible, the distance function is discontinuous.

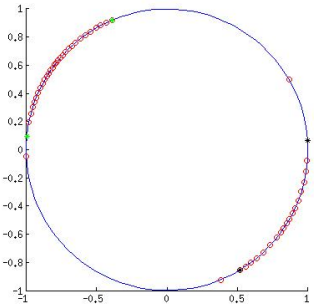
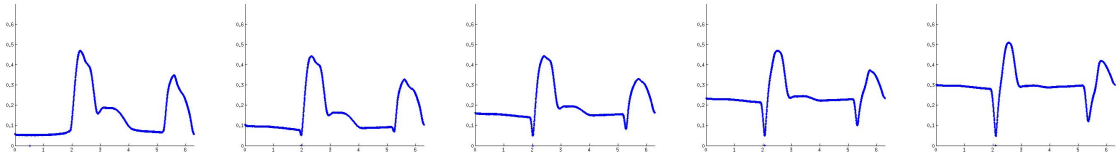
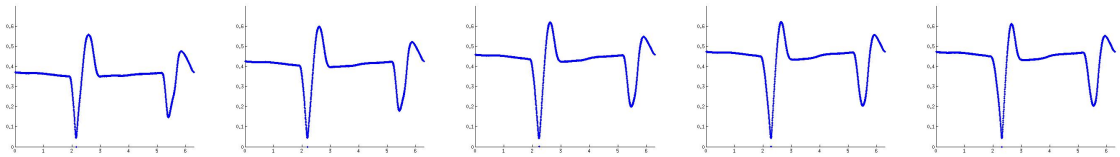


Figure 6.44: Final Configuration, 2 Tapes: 50 Points



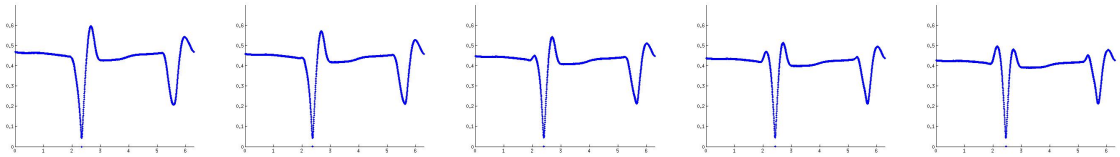
(a) Point 1 (b) Point 2 (c) Point 3 (d) Point 4 (e) Point 5

Figure 6.45: 2 Tapes: Distance Function Based at Points 1-5



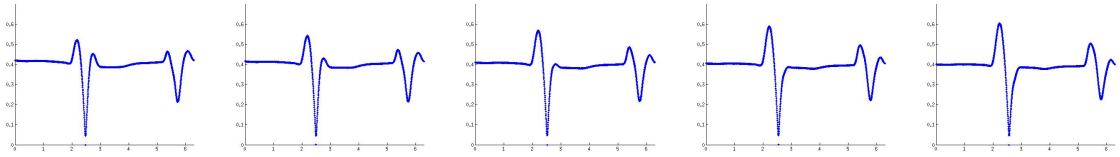
(a) Point 6 (b) Point 7 (c) Point 8 (d) Point 9 (e) Point 10

Figure 6.46: 2 Tapes: Distance Function Based at Points 6-10



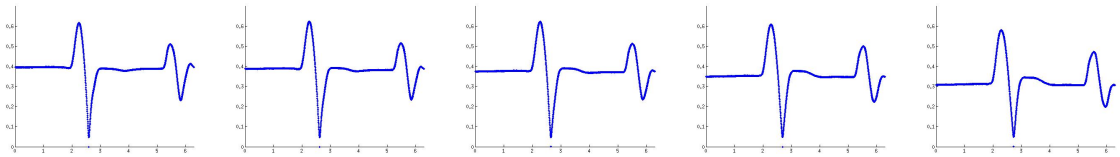
(a) Point 11 (b) Point 12 (c) Point 13 (d) Point 14 (e) Point 15

Figure 6.47: 2 Tapes: Distance Function Based at Points 11-15



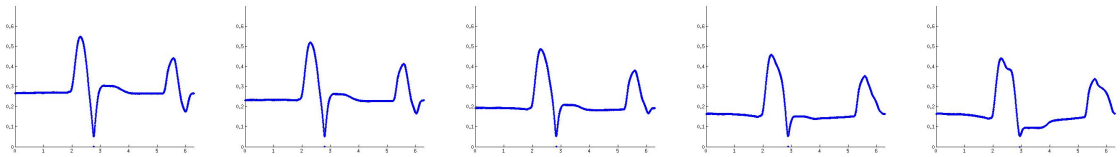
(a) Point 16 (b) Point 17 (c) Point 18 (d) Point 19 (e) Point 20

Figure 6.48: 2 Tapes: Distance Function Based at Points 16-20



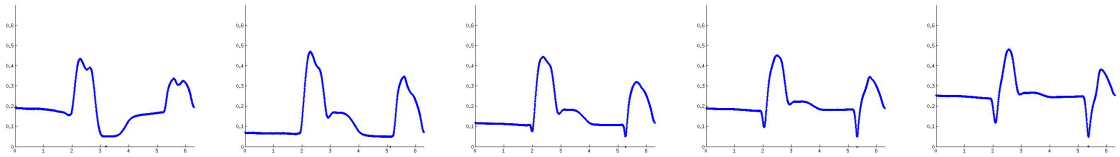
(a) Point 21 (b) Point 22 (c) Point 23 (d) Point 24 (e) Point 25

Figure 6.49: 2 Tapes: Distance Function Based at Points 21-25



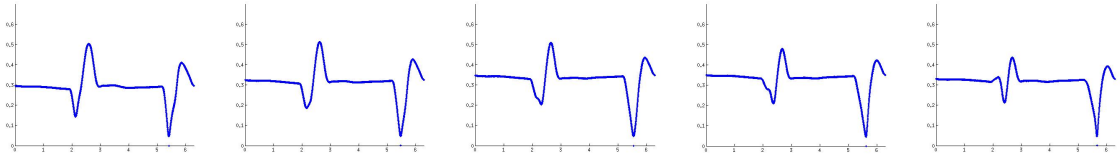
(a) Point 26 (b) Point 27 (c) Point 28 (d) Point 29 (e) Point 30

Figure 6.50: 2 Tapes: Distance Function Based at Points 26-30



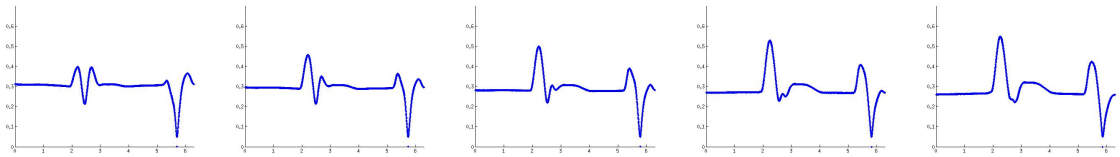
(a) Point 31 (b) Point 32 (c) Point 33 (d) Point 34 (e) Point 35

Figure 6.51: 2 Tapes: Distance Function Based at Points 31-35



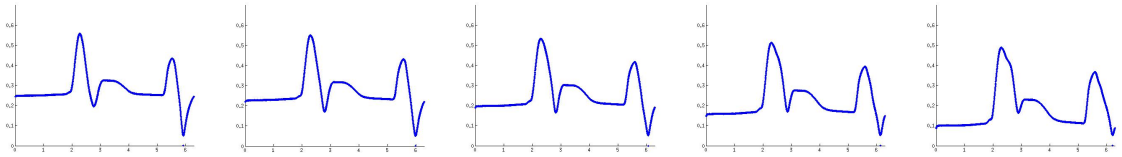
(a) Point 36 (b) Point 37 (c) Point 38 (d) Point 39 (e) Point 40

Figure 6.52: 2 Tapes: Distance Function Based at Points 36-40



(a) Point 41 (b) Point 42 (c) Point 43 (d) Point 44 (e) Point 45

Figure 6.53: 2 Tapes: Distance Function Based at Points 41-45



(a) Point 46 (b) Point 47 (c) Point 48 (d) Point 49 (e) Point 50

Figure 6.54: 2 Tapes: Distance Function Based at Points 46-50

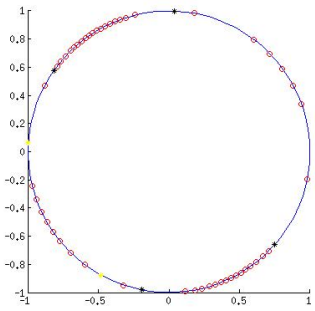
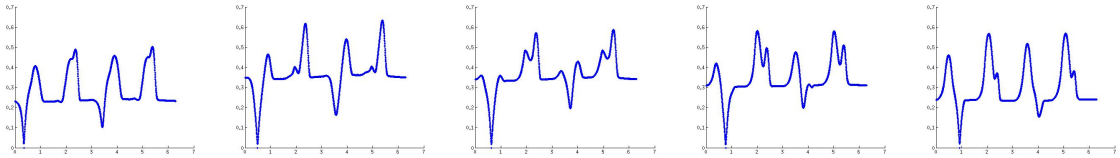
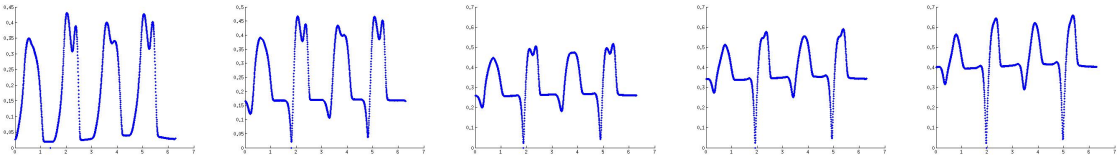


Figure 6.55: Final Configuration, 2 Horiz., 1 Vert., 1 × Tape: 50 Points



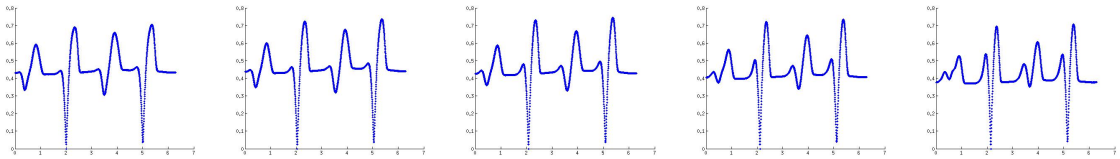
(a) Point 1 (b) Point 2 (c) Point 3 (d) Point 4 (e) Point 5

Figure 6.56: 4 Tapes: Distance Function Based at Points 1-5



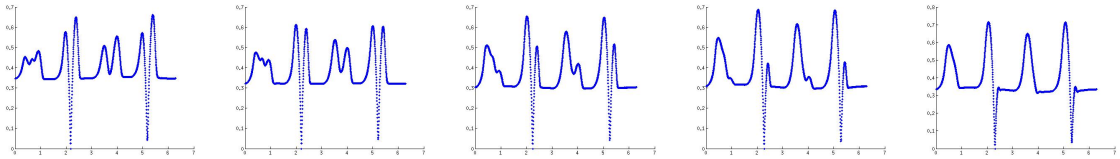
(a) Point 6 (b) Point 7 (c) Point 8 (d) Point 9 (e) Point 10

Figure 6.57: 4 Tapes: Distance Function Based at Points 6-10



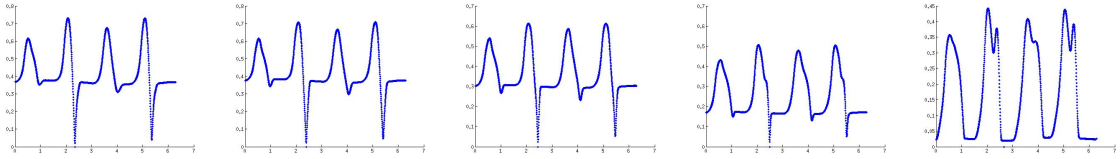
(a) Point 11 (b) Point 12 (c) Point 13 (d) Point 14 (e) Point 15

Figure 6.58: 4 Tapes: Distance Function Based at Points 11-15



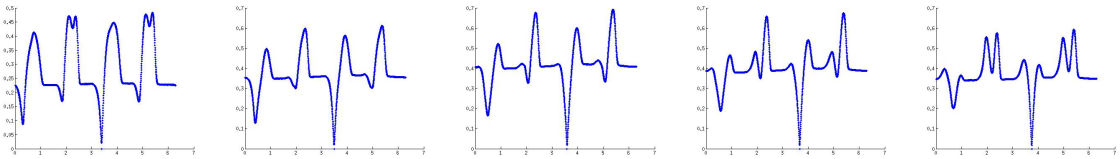
(a) Point 16 (b) Point 17 (c) Point 18 (d) Point 19 (e) Point 20

Figure 6.59: 4 Tapes: Distance Function Based at Points 16-20



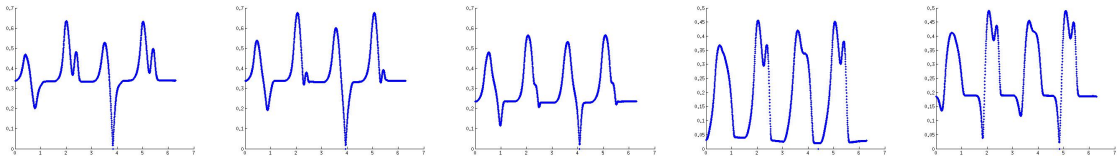
(a) Point 21 (b) Point 22 (c) Point 23 (d) Point 24 (e) Point 25

Figure 6.60: 4 Tapes: Distance Function Based at Points 21-25



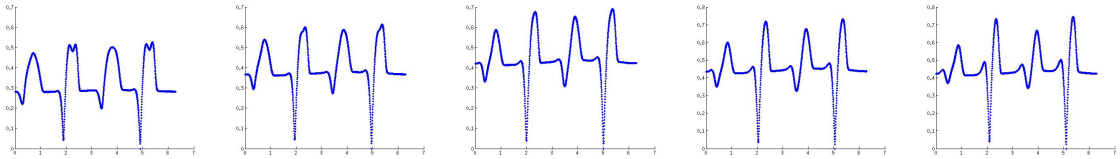
(a) Point 26 (b) Point 27 (c) Point 28 (d) Point 29 (e) Point 30

Figure 6.61: 4 Tapes: Distance Function Based at Points 26-30



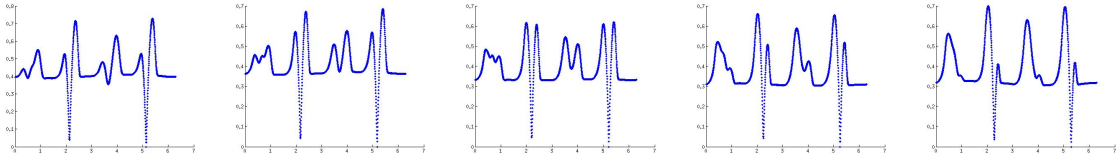
(a) Point 31 (b) Point 32 (c) Point 33 (d) Point 34 (e) Point 35

Figure 6.62: 4 Tapes: Distance Function Based at Points 31-35



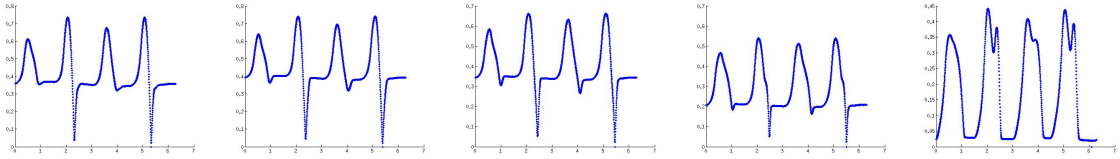
(a) Point 36 (b) Point 37 (c) Point 38 (d) Point 39 (e) Point 40

Figure 6.63: 4 Tapes: Distance Function Based at Points 36-40



(a) Point 41 (b) Point 42 (c) Point 43 (d) Point 44 (e) Point 45

Figure 6.64: 4 Tapes: Distance Function Based at Points 41-45



(a) Point 46 (b) Point 47 (c) Point 48 (d) Point 49 (e) Point 50

Figure 6.65: 4 Tapes: Distance Function Based at Points 46-50

Chapter 7

A Measure of Distortion

7.1 Introduction

Any configuration of points on S^1, along with associated vector spaces, which is used as a representation for a vector bundle, admits some loss of information. In this chapter, we develop a function that can be used to measure the amount of distortion introduced by using a given vector bundle approximation. We will call this function a *distortion function.*

We define a distortion function in the following way. Fix a set of allowable camera locations $L = \{\ell_i\}_{i=1}^n$ on S^1, an associated data bundle where each $x_i \in S^1$ has attached vector space U_{x_i}, and a choice of distance measure ∂ on the Grassmannian. The distortion function is a function of a configuration C of points on S^1. For each element ℓ_i of L, calculate the distance of the associated vector space U_{ℓ_i} to the vector space U_{x_j}, for each x_j in C. That is, for a fixed $\ell_i \in L$, find $\partial(U_{\ell_i}, U_{x_j})$ for each $x_j \in C$. Associate to each $\ell_i \in L$ a value m_i, defined to be the minimum distance achieved by any U_{x_j}. The output of the distortion function for a given configuration C is the sum of all the m_i's.

That is, the distortion function is a function of a configuration $C = \{x_j\}_{j=1}^r$ and

is defined to be

$$\mathcal{D}(C) := \sum_{\ell_i \in L} \min_{x_j \in C} \partial\left(U_{\ell_i}, U_{x_j}\right).$$

Note that the distortion function is positive by definition. The distortion function is an approximation of the integral

$$\int \min_{x_j \in C} \partial\left(U_\theta, U_{x_j}\right) d\theta.$$

7.2 Results

7.2.1 Red Punch Bowl Data Set

This data set is described in Section 5.2 and consists of pictures of a red punch bowl, with no distinguishing features from any single perspective. We expect that the data set of images of the red punch bowl has a distortion approximately the same for a random set of points, an evenly spaced set of points, and a set of points that is the result of running the Nearest Neighbor Dispersion Algorithm. We present here the results for each of these configurations.

We calculate the distortion for 15 different random sets of 25 points on S^1. The mean of these distortions is 23.6296, with a variance of 0.0114. The best configuration, in the sense that it yields the smallest distortion, has a distortion of 23.5028. The fact that the variance of the distortion is small across random sets of points suggests that it is unlikely that there exists a special configuration with very little distortion. This seems reasonable given the nature of the set.

The distortion for an evenly spaced configuration of points is 23.4578.

Finally, we use the final configurations after five runs of the Nearest Neighbor Dispersion Algorithm on 25 randomly spaced points. The mean distortion from these five configurations is 23.4582, with a variance of 0.0037. The minimum distortion from these five configurations is 23.4040. This is only a slight improvement on the random and evenly spaced configurations.

7.2.2 Red Punch Bowl with One Tape Data Set

This data set consists of images of a red punch bowl with one horizontal piece of white tape on it. This distinguishes a region of S^1, in which the attached vector spaces have some significant distance between each other and between points outside the region. We should therefore expect that the Nearest Neighbor Dispersion Algorithm will lead to a configuration which represents the underlying vector bundle better than a random or evenly spaced configuration.

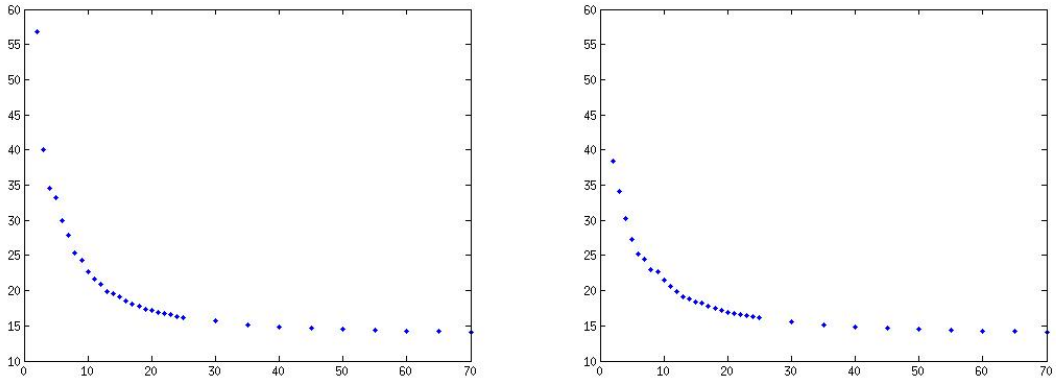
We again begin by calculating the distortion for 15 different random sets of 25 points on S^1. The mean distortion of this set is 27.3445, with a variance of 17.1226. The best configuration, in the sense that it yields the smallest distortion, has a distortion of 22.5814. The variance of the distortions for this set is significantly larger than that of the set of images of the red punch bowl with no tape. It is therefore likely that for this data set, there exist special configurations with relatively little distortion.

The distortion for an evenly spaced configuration of points is 22.9980, slightly larger than the distortion of the best random configuration.

On the other hand, after three runs of the Nearest Neighbor Dispersion Algorithm on 15 sets of 25 randomly distributed points, we get the following distortions: 16.2834, 16.1563, and 16.2101. Each of these configurations gives a significant improvement over the random and evenly spaced configurations. Notice that the variance is also small (0.0041), suggesting that the algorithm will give consistently good results for this data set.

Let us now consider other numbers of points. We find configurations using the Nearest Neighbor Dispersion Algorithm for varying numbers of points, from two to 70. The output of the distortion function has relatively large variance for small numbers of points. We therefore calculate the distortion for 15 different runs of the Nearest Neighbor Dispersion Algorithm for two to 24 points. The variance is small for large numbers of points, so we calculate the distortion for only three runs of the Nearest

Neighbor Dispersion Algorithm for every 5th number of points from 25 to 70 points. In Figure 7.1, we show the results of these calculations. The $x-$axis represents the number of points in a configuration C. Figure 7.1a shows the average distortion over all runs of the algorithm, while Figure 7.1b shows the minimum distortion over all runs of the algorithm.



(a) Distortion Function Based on Average

(b) Distortion Function Based on Min

Figure 7.1: 1 Tape: Distortion Function

The variance of the distortion for the 15 runs of the algorithm on 2 to 24 points is shown in the following table.

Number of Points	2	3	4	5	6
Distortion Variance	1577.4652	33.5681	20.4302	26.0622	12.1847
Number of Points	7	8	9	10	11
Distortion Variance	5.9277	2.7740	1.8415	0.8430	0.9650
Number of Points	12	13	14	15	16
Distortion Variance	0.3420	0.2129	0.1806	0.1464	0.0254
Number of Points	17	18	19	20	21
Distortion Variance	0.0223	0.0184	0.0332	0.0090	0.0332
Number of Points	22	23	24		
Distortion Variance	0.0061	0.0214	0.0033		

The variance of the distortion for the 3 runs of the algorithm on 25 to 70 points is shown in the following table.

Number of Points	25	30	35	40	45
Distortion Variance	0.0041	0.0019	0.0001	0.0000	0.0010
Number of Points	50	55	60	65	70
Distortion Variance	0.0023	0.0009	0.0020	0.0001	0.0005

7.2.3 Red Punch Bowl with Two Tapes Data Set

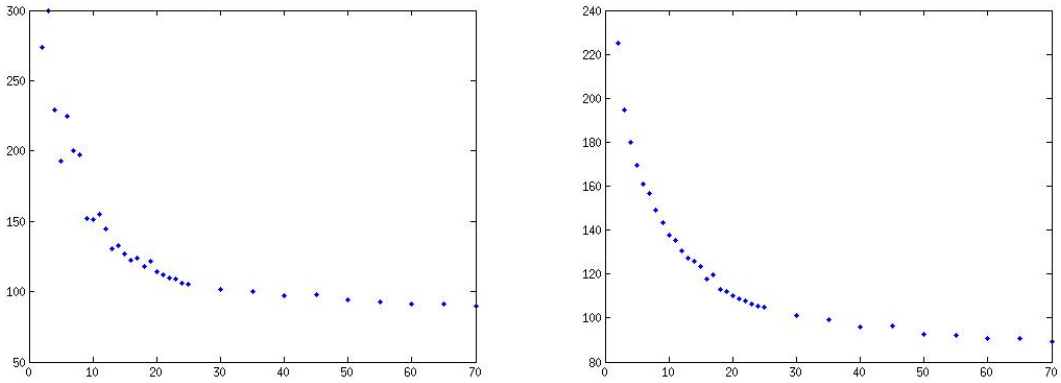
This data set is described in Section 5.2 and consists of images of a red punch bowl with one horizontal piece of white tape on one side and one pair of white tape pieces that form an $\times$ on the opposite side. Hence there are two distinguished regions of S^1 in which the attached vector spaces have significant distance between each other and between points outside the region. We therefore expect that with this set, as with the red punch bowl with one piece of tape data set, the Nearest Neighbor Dispersion Algorithm will lead to a configuration which represents the underlying vector bundle better than a random or evenly spaced configuration.

We calculate the distortion for 15 different random sets of 25 points on S^1. The mean distortion is 141.8079, with a variance of 211.0433. The best configuration, in the sense that it yields the smallest distortion, has a distortion of 123.2293. Note that the variance of the distortions is very large. It is therefore probable that there exist special configurations with relatively little distortion.

The distortion for an evenly spaced configuration of points is 127.7814, slightly larger than the distortion of the best random configuration.

After three runs of the Nearest Neighbor Dispersion Algorithm on 15 sets of 25 randomly distributed points, we get the following distortions: 104.7388, 105.5406, and 105.5080. Each of these configurations is an improvement over the random and evenly spaced configurations. Notice that the variance is also small (0.2059), suggesting that the algorithm will give consistently good results for this data set.

We find configurations using the Nearest Neighbor Dispersion Algorithm for varying numbers of points, from two to 70. As with the one tape data set, the output of the distortion function has relatively large variance for small numbers of points. The variance is consistently small for sufficiently large numbers of points after the algorithm has been run. We therefore calculate the distortion for 15 different runs of the Nearest Neighbor Dispersion Algorithm for two to 24 points. We calculate the distortion for three runs of the Nearest Neighbor Dispersion Algorithm for every 5th number of points from 25 to 70 points. In Figure 7.2, we show the results of these calculations. Figure 7.2a shows the average distortion over all runs of the algorithm at a given number of points, while Figure 7.2b shows the minimum distortion over all runs of the algorithm. The $x-$axis again represents the number of points in a configuration.



(a) Distortion Function Based on Average

(b) Distortion Function Based on Min

Figure 7.2: 2 Tapes: Distortion Function

The variance of the distortion for the 15 runs of the algorithm on 2 to 24 points is shown in the following table.

Number of Points	2	3	4	5	6
Distortion Variance	5625.8623	6883.8176	1611.6328	1178.8943	1545.1464
Number of Points	7	8	9	10	11
Distortion Variance	1764.7899	1982.2719	506.4759	602.9781	994.5140
Number of Points	12	13	14	15	16
Distortion Variance	924.8896	1.6737	406.3475	36.2721	1.8548
Number of Points	17	18	19	20	21
Distortion Variance	130.7021	5.3107	230.0736	9.0631	9.6844
Number of Points	22	23	24		
Distortion Variance	7.9975	10.6192	0.1123		

The variance of the distortion for the 3 runs of the algorithm on 25 to 70 points is shown in the following table.

Number of Points	25	30	35	40	45
Distortion Variance	0.3214	0.3367	1.5156	0.3217	1.5245
Number of Points	50	55	60	65	70
Distortion Variance	2.1565	0.5407	0.1290	0.3654	0.1264

7.2.4 Punch Bowl with Four Tapes Data Set

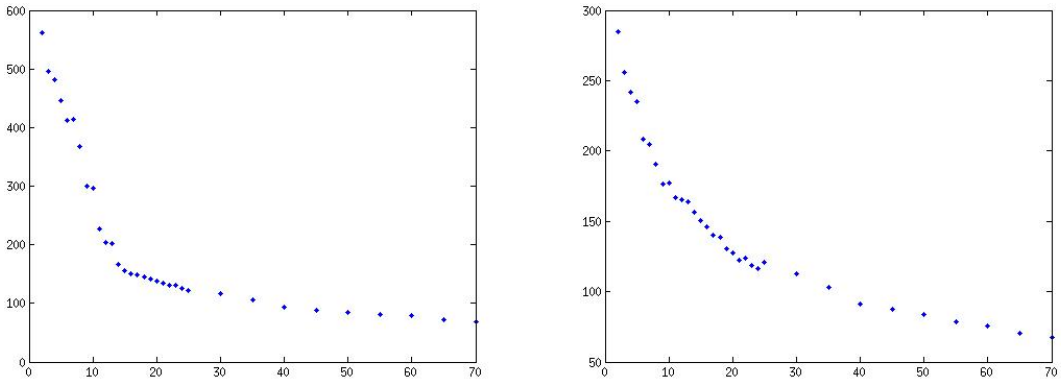
This data set is described in Section 5.2 and consists of images of a red punch bowl with tape on four sides. There is one horizontal piece of white tape on one side and one pair of white tape pieces that form an $\times$ on the opposite side. There is a vertical strip of tape on the two sides in between the horizontal and $\times$ tapes. There are therefore four distinguished regions of S^1 in which the attached vector spaces have significant distance between each other and between points outside the region. Hence, we expect that the Nearest Neighbor Dispersion Algorithm will lead to a configuration which represents the vector bundle better than a random or evenly spaced configuration.

We calculate the distortion for 15 different random sets of 25 points on S^1. The mean distortion is 158.5277, with a variance of 269.6836. The best configuration, in the sense that it yields the smallest distortion, has a distortion of 127.0842. The variance of the distortions is large; therefore, it is probable that there exist special configurations with relatively little distortion.

The distortion for an evenly spaced configuration of points is 160.2952, relatively large compared to the best random configuration but about as good as the mean random configuration distortion.

After three runs of the Nearest Neighbor Dispersion Algorithm on 15 sets of 25 randomly distributed points, we get the following distortions: 120.8192, 122.8273, and 122.8704. Each of these configurations is an improvement over the random and evenly spaced configurations. Also, the variance is small (1.3736), suggesting that the algorithm will give consistently good results for this data set.

We find configurations using the Nearest Neighbor Dispersion Algorithm for varying numbers of points, from two to 70. The output of the distortion function has relatively large variance for small numbers of points and is consistently small for sufficiently large numbers of points after the algorithm has been run. We therefore calculate the distortion for 15 different runs of the Nearest Neighbor Dispersion Algorithm for two to 24 points. We calculate the distortion for three runs of the Nearest Neighbor Dispersion Algorithm for every 5th number of points from 25 to 70 points. In Figure 7.3, we show the results of these calculations. Figure 7.3a shows the average distortion over all runs of the algorithm at a given number of points, while Figure 7.3b shows the minimum distortion over all runs of the algorithm.



(a) Distortion Function Based on Average

(b) Distortion Function Based on Min

Figure 7.3: 2 Horiz., 1 Vert., 1 $\times$ Tape: Distortion Function

The variance of the distortion for the 15 runs of the algorithm on 2 to 24 points is shown in the following table.

Number of Points	2	3	4	5	6
Distortion Variance	8634.4663	5486.4677	6697.6518	4623.8463	10856.0459
Number of Points	7	8	9	10	11
Distortion Variance	4460.3491	7211.6925	12186.9608	11350.4149	5813.1843
Number of Points	12	13	14	15	16
Distortion Variance	3588.6172	4319.2542	62.4439	38.8679	31.3581
Number of Points	17	18	19	20	21
Distortion Variance	46.2916	43.3193	47.0027	65.5488	47.5921
Number of Points	22	23	24		
Distortion Variance	58.9935	83.1293	31.2248		

The variance of the distortion for the 3 runs of the algorithm on 25 to 70 points is shown in the following table.

Number of Points	25	30	35	40	45
Distortion Variance	1.3736	27.3523	12.5073	5.4980	0.4629
Number of Points	50	55	60	65	70
Distortion Variance	0.0014	4.4310	15.1902	4.1607	1.4954

7.2.5 Volleyball Data Set

The volleyball data set consists of images of a red, white, and blue volleyball, with each view having some distinguishing feature. We use all three color filters and hence assume that we have a rank three vector bundle. We expect that a roughly uniform distribution of points will have a low distortion value, and that a random distribution will have slightly larger distortion.

Let us begin by fixing a distance function and comparing the distortion over random and evenly spaced sets of points to the distortion after the algorithm has been run. We calculate the distortion for 5 different random sets of 20 points on S^1

using the Fubini-Study metric. The mean of the distortions for this set is 1567.0857, with a variance of 908.8164. The best configuration, in the sense that it yields the smallest distortion, has a distortion of 1520.0779.

The distortion for an evenly spaced configuration of points is 1436.0169, a significant improvement over every randomly generated configuration.

Finally, we measure the final configurations after three runs of the Nearest Neighbor Dispersion Algorithm on 20 randomly spaced points. With the Fubini-Study metric and the usual parameters, we get configurations with distortions of 1512.7078, 1593.2825, and 1485.1701, which are approximately the same as the evenly spaced configuration and better than most of the random configurations.

In Figure 7.4, we show the distortion function for five runs of the algorithm with the Fubini-Study metric on random sets of two to 24 points. In the table below, we show the variance for the distortion corresponding to these configurations.

The variance of the distortion for the 5 runs of the algorithm on 2 to 24 points using the Fubini-Study metric is given below.

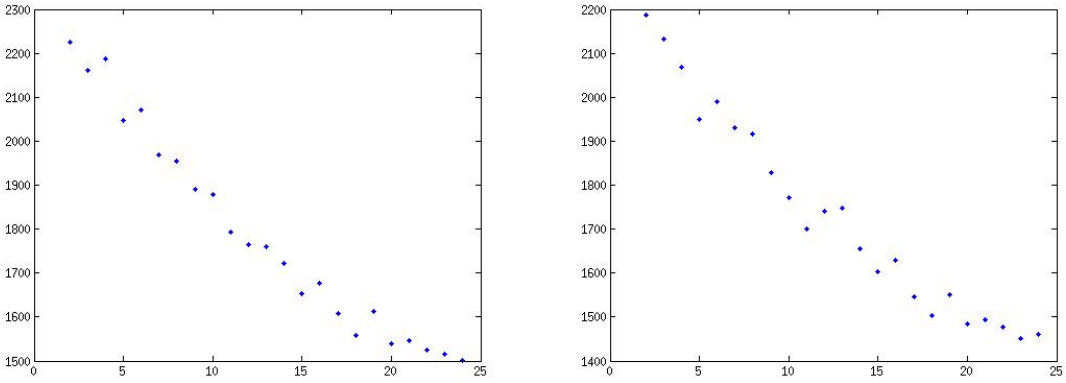
Number of Points	2	3	4	5	6
Distortion Variance	2730.3091	489.8366	8798.4427	7120.2832	12757.5389
Number of Points	7	8	9	10	11
Distortion Variance	1975.5630	2132.6311	3565.9060	7257.9882	3065.9247
Number of Points	12	13	14	15	16
Distortion Variance	346.8440	111.8989	3918.3117	1259.2306	1428.0483
Number of Points	17	18	19	20	21
Distortion Variance	3021.9938	2691.4559	1676.3971	1729.6772	3951.5757
Number of Points	22	23	24		
Distortion Variance	6473.4569	3358.6594	1083.1993		

An interesting aspect of the color filter data set is that it is a data set on which we can compare the various distance functions that we discuss in Section 2.6. We compute the distortion over five runs of the Nearest Neighbor Dispersion Algorithm

for each of the five different distance measures on each number of points from two to 24. The results of these runs are presented in Figures 7.4 to 7.8. A close observation of the scales on these figures reveals that the distance measure defined by the smallest principal angle consistently minimizes distortion far better than the four distance metrics. Note however, that the smallest principal angle distance function produces smaller values than the other distance metrics on other configurations as well. In the following table, we show the minimal distortion achieved for each of five random configurations, one evenly spaced configuration, and five runs of the Nearest Neighbor Dispersion Algorithm on a set of 20 points. For a fixed distance function, the Nearest Neighbor Dispersion Algorithm does not, in general, provide configurations that are significant improvements over uniform or random distributions of points. This is not unsurprising given the nature of the object.

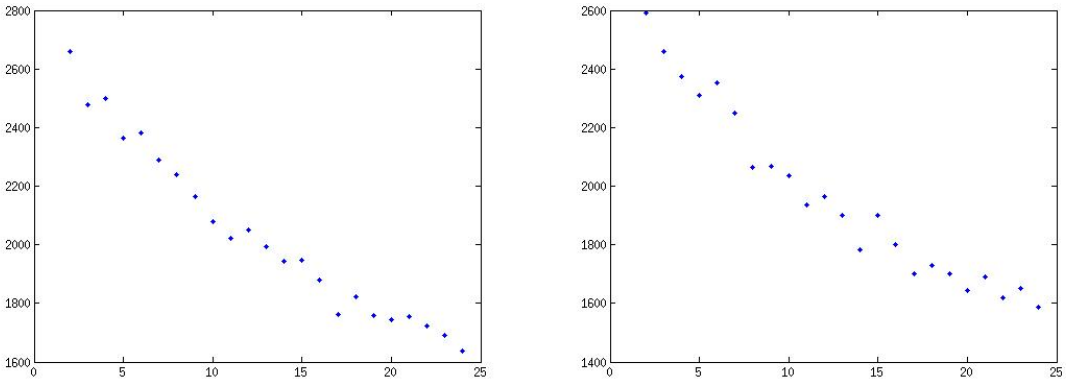
Best Distortion: Configurations of 20 Points, Volleyball Data

	Random	Uniform	NNDA
Fubini-Study metric	1520.0779	1436.0169	1485.1701
Geodesic metric	1634.2040	1511.3790	1644.0872
Chordal metric	1422.5835	1364.0296	1476.0362
Subspace metric	1140.3706	1064.5326	1104.2550
Smallest principal angle function	257.7373	252.3587	267.3751



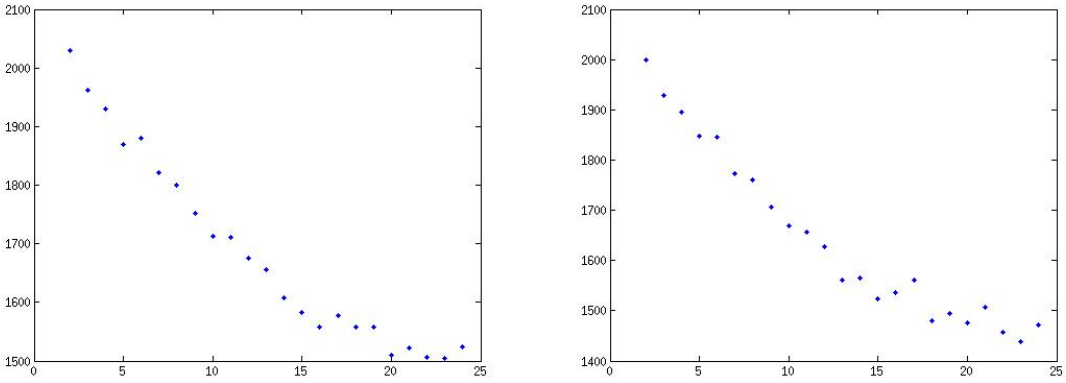
(a) Distortion Function Based on Average (b) Distortion Function Based on Min

Figure 7.4: Volleyball: Distortion Function, Fubini-Study Metric



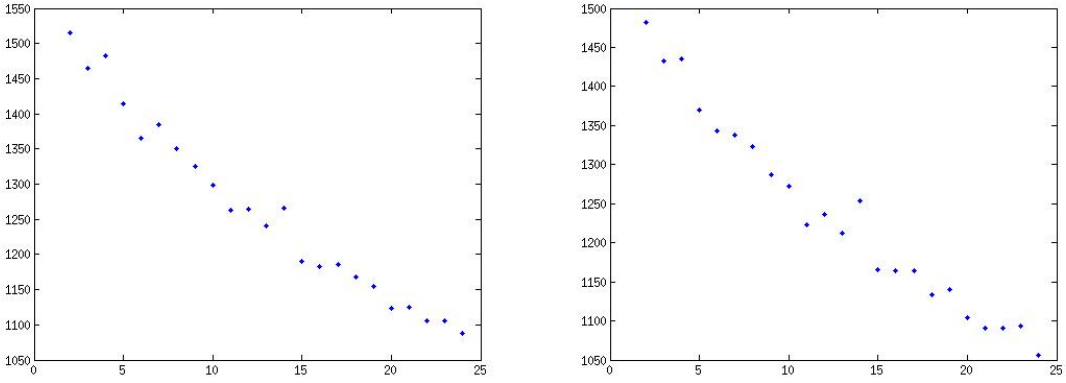
(a) Distortion Function Based on Average (b) Distortion Function Based on Min

Figure 7.5: Volleyball: Distortion Function, Geodesic Metric



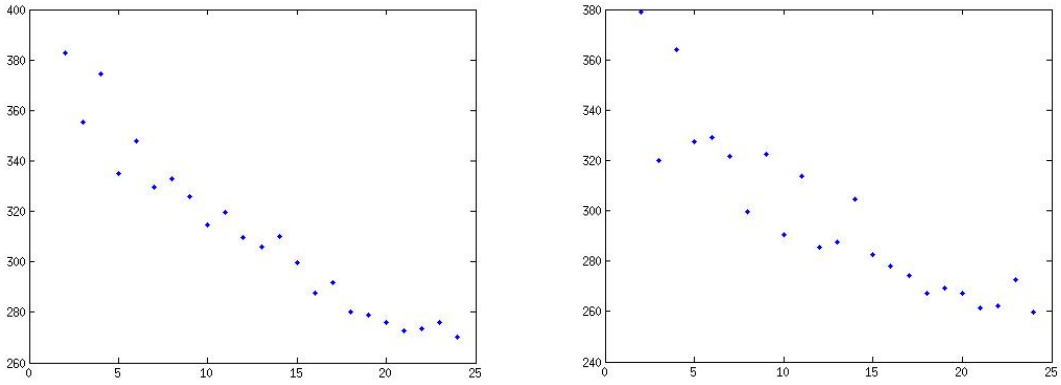
(a) Distortion Function Based on Average (b) Distortion Function Based on Min

Figure 7.6: Volleyball: Distortion Function, Chordal Metric



(a) Distortion Function Based on Average (b) Distortion Function Based on Min

Figure 7.7: Volleyball: Distortion Function, Subspace Metric



(a) Distortion Function Based on Average

(b) Distortion Function Based on Min

Figure 7.8: Volleyball: Distortion Function, Smallest Principal Angle Distance Function

Chapter 8

Artificially Rotating Data

8.1 Projections and Eigenpictures

8.1.1 Introduction

By capturing pictures of an object on a record player, we are, in effect, allowing the camera to move around the equator of a sphere with the object in a fixed position in the center of the sphere. In this section, we discuss one way of modeling a different situation: the camera is in a fixed position and is allowed to rotate. That is, the camera rotates along an axis between itself and the center of the object.

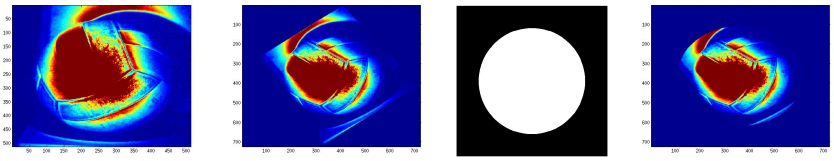
8.1.2 Methods

We begin with one picture of an object. In the experiment described in this section, we work with a data set of images of a red, white, and blue volleyball. Images are taken at a rate of approximately 220 pictures per second and at a resolution of 512×512 pixels. We present the results pertaining to pictures taken in the red filter. We attempt to align the center of the picture with the center of the volleyball, so that the rotation of the camera simulates rotation of the object.

We wish to rotate a picture, I, by every integer degree about its center. However,

most integer degree rotations result in an object which is no longer a matrix. For example, in Figure 8.1b, we see an image rotated by 38 degrees; it is no longer rectangular and its entries do not align with the entries of a rectangular matrix of similar size. We therefore embed each rotation of I in the center of a sufficiently large matrix of zeros (with a resolution of 512×512, the smallest size necessary to fit all possible integer degree rotations is 724×724). We will use $\tilde{I}_\theta$ to denote an embedding of a rotation of I by θ degrees in the counterclockwise direction. Since it is the case that most entries of $\tilde{I}_\theta$ corresponding to a location in the original image I cannot be filled in directly by entries of I, we must use interpolation to completely determine $\tilde{I}_\theta$.

We now have each integer degree rotation of I embedded in a larger matrix, but we have introduced a great deal of statistical variance by embedding the rotated I's in matrices of zeros. We therefore cover each $\tilde{I}_\theta$ with a mask, Z. We define Z to be a matrix of ones and zeros, where an entry of Z is defined to be zero if there exists at least one θ for which $\tilde{I}_\theta$ has a value of zero due to the original embedding. All other values of Z are defined to be one. Now we can take the componentwise product of Z with each $\tilde{I}_\theta$ and obtain a masked rotation of I, called $\tilde{I}_\theta^Z$. This process is seen in Figure 8.1. Note that the coloring is scaled, and therefore zeros do not appear black.



(a) Red Filter (b) Rotated 38° (c) Mask (d) Rotated,Masked

Figure 8.1: Rotation and Masking of Volleyball A

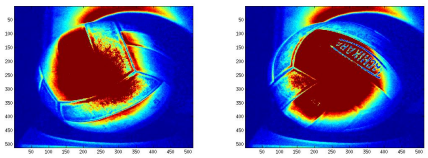
We flatten each $\tilde{I}_\theta^Z$ by recording entries in a column vector in order from left to right, top to bottom. We store only those entries which lie inside of the mask. That is, we store only the entries which correspond to an entry of 1 in Z. Now we can

concatenate these vectors in order of degree of rotation in a data matrix, X. When the resolution is 512×512, this results in a matrix X of size $204,699 \times 360$.

We determine the first several principal components of the data using the singular value decomposition as explained in Chapter 2. Let U, Σ, V be given by the singular value decomposition so that $X = U\Sigma V^\top$. We project X into two and three dimensions using columns 2-3 and 2-4 of U, respectively. As before, we discard the first column of U in order to mean-subtract the data.

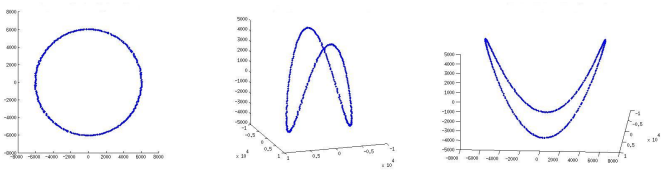
8.1.3 Results

We present the results for two different original pictures, Volleyball A and Volleyball B, shown in Figure 8.2. In Figures 8.3 and 8.4, we show the projection of X into two dimensions and into three dimensions for Volleyball A and B, respectively. In both cases, we get a very good approximation of a circle as the image of the projection into two dimensions. However, the circle resulting from Volleyball A is traced out once, but in the case of Volleyball B, the circle is traced out twice. Most frequently, the result of the projection is similar to that of Volleyball A.



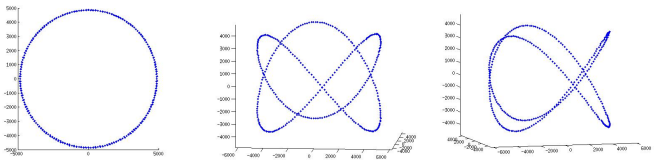
(a) Volleyball A (b) Volleyball B

Figure 8.2: Original Volleyballs



(a) 2-D (b) 3-D, View 1 (c) 3-D, View 2

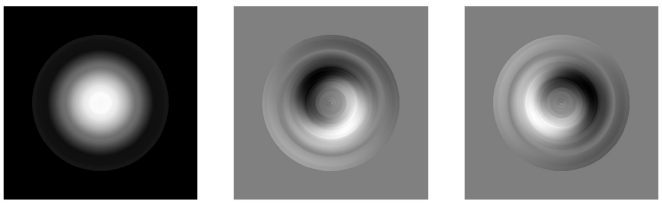
Figure 8.3: Projection of Volleyball A into Two and Three Dimensions



(a) 2-D (b) 3-D, View 1 (c) 3-D, View 2

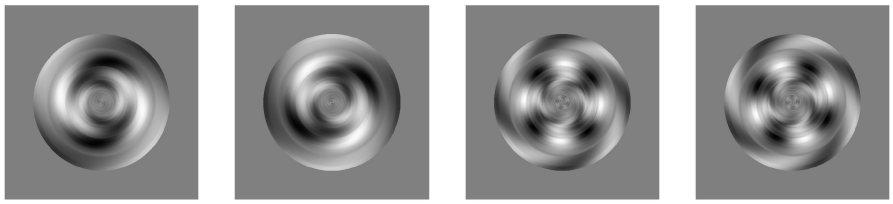
Figure 8.4: Projection of Volleyball B into Two and Three Dimensions

A visualisation of the first seven eigenpictures, or unflattened principal components, for the two volleyballs can be seen in Figures 8.5 through 8.8. Each figure is obtained by unflattening the given column vector of U as part of a masked matrix. Note that aside from the eigenpicture corresponding to the mean, the eigenpictures appear to come in pairs of similar energy. Again, coloring has been scaled, so zeros do not appear black.



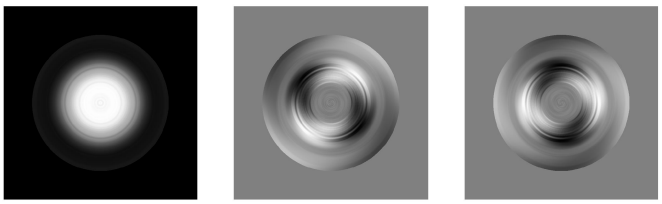
(a) 1: Mean (b) 2 (c) 3

Figure 8.5: Volleyball A: Eigenpictures One Through Three



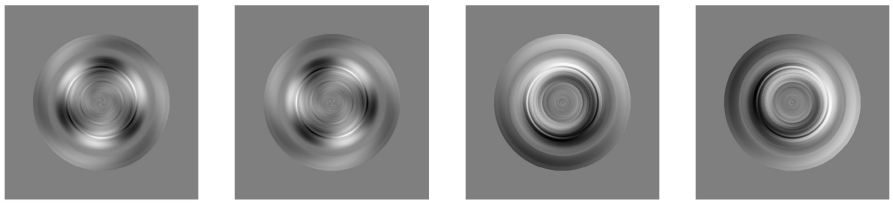
(a) 4 (b) 5: Unused (c) 6: Unused (d) 7: Unused

Figure 8.6: Volleyball A: Eigenpictures Four Through Seven



(a) 1: Mean (b) 2 (c) 3

Figure 8.7: Volleyball B: Eigenpictures One Through Three



(a) 4 (b) 5: Unused (c) 6: Unused (d) 7: Unused

Figure 8.8: Volleyball B: Eigenpictures Four Through Seven

Notice that in the case of Volleyball A, the eigenpictures are ordered by frequency of relative extrema. This is not the case with the eigenpictures for Volleyball B. We conjecture that the times when the projection into two dimensions traces a circle more than once coincide with the eigenpicture pairs which are not ordered by frequency of relative extrema.

In the projections into $\mathbb{R}^2$ which trace the circle once, the angle of rotation θ is preserved. In those which trace the circle more than once, the angle is preserved up to an integer multiple corresponding to the number of times the circle is traced out. Therefore, this projection gives us a method for recovering the angle (up to a multiple) between two pictures chosen randomly from the set.

8.2 Artificial Rotation as a Means of Intersecting Data Sets

8.2.1 Introduction

In this section, we combine artificial rotation with multiple data sets in order to find locations where the data sets match. Consider the following situation. Fix an object at the center of a sphere. Allow the camera to travel along a great circle while capturing images of the object. Call this great circle $\mathcal{C}_1$. Now move the camera and allow it to travel along a great circle, $\mathcal{C}_2$, with $\mathcal{C}_2 \neq \mathcal{C}_1$, again capturing images as it travels. Since we have two distinct great circles on a sphere, there must be two points of intersection. In this section, we develop an algorithm to find these two points.

In the collection of real data, we note that images are not allowed to be collected at every possible location on a given great circle $\mathcal{C}_i$ but instead are collected at a finite number of locations. Therefore, we seek an algorithm that finds two points that get as close as possible to the two points of intersection of $\mathcal{C}_1$ and $\mathcal{C}_2$.

Notice that the images in the data sets at the points of intersection will be the

same up to a rotation by an angle equal to the angle of incidence between the great circles. Given this observation, let us consider the situation from a different point of view. If we allow the camera to be placed anywhere on the sphere and to rotate along the axis between itself and the object, then the space of all camera locations and orientations is the special orthogonal group $SO(3)$. Image collection on $\mathcal{C}_i$ creates a map f from a surface into pixel space. Note that the surface is two-dimensional because at each point of a great circle $\mathcal{C}_i$, which is one-dimensional, we attach a circle given by all rotations of the camera from that particular perspective. Hence, each set of camera locations and orientations along $\mathcal{C}_i$ used in the collection of a data set is a sample from a two-dimensional surface inside $SO(3)$. If we take $f(\mathcal{C}_1) \cap f(\mathcal{C}_2)$, the intersection of the two data sets in pixel space, we expect a one-dimensional object, namely two circles. These two circles correspond to two images from each original data set along with all of their artificial rotations. We can exploit this fact in the creation of our algorithm by working not just with the images from the data sets, but also with their artificial rotations.

8.2.2 Methods

Throughout this section, we will focus on a data set of images of a volleyball collected at 1000 pictures per second using all three color filters at a resolution of 256×256. In order to create a smaller data set on which to run the algorithm, we reduce this set in three ways: we take every 5^{th} picture, we combine the color filter images to produce grayscale images, and we reduce to a resolution of 64×64 by averaging 4×4 blocks of pixels. An image from this data set is shown in Figure 8.9. We give some notation and then we state the general algorithm for finding the two points of intersection. We explain each step in more detail after the outline of the algorithm.

Fix a data set corresponding to images collected along the great circle $\mathcal{C}_i$. For each picture, we artificially rotate by each integer degree, mask, and flatten to obtain a vector. We will denote by I_j^i the vector corresponding to the j^{th} image in the i^{th} data

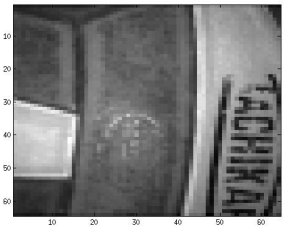


Figure 8.9: Original Volleyball, 64×64 Resolution

set.

Because we are working with rotations of an image, we would like to use the discrete Fourier transform. Let M be a matrix of size $m_1 \times m_2$, where m_1 is the number of entries inside the mask, and where each column represents an image at a particular stage of rotation. We generate a vector using the Fourier transform in the following way. We begin by creating a vector w, of size $1 \times m_1$. The vector w has entries equal to zero or one, with ones in those locations that correspond to the right half of the center row and the right half of the row one below center of an image before it is flattened. If each column of M is a flattened image, then the product wM is a vector of size $1 \times m_2$, whose entries carry information about the two half-rows in the right center of the mask of each image. We then compute the discrete Fourier transform and retain the first $\left\lfloor \frac{m_2}{2} \right\rfloor + 1$ coefficients. We denote the vector containing these coefficients by $\mathcal{F}(M)$. Note that retaining only the first $\left\lfloor \frac{m_2}{2} \right\rfloor + 1$ coefficients results in no loss of information because Fourier coefficients come in complex conjugate pairs. Depending on the step of the algorithm, we use different portions of the vector $\mathcal{F}(M)$. When we use a subset of the entries in $\mathcal{F}(M)$, we will denote the vector being used by $\mathcal{F}_S(M)$, where S is a list of the indices of the entries in $\mathcal{F}(M)$ to be used. We can now state the algorithm.

Algorithm 8.1. ***(Outline)***

Input: *Two sets of images collected along two great circles,* $\mathcal{C}_1$ *and* $\mathcal{C}_2$*, respectively, a choice of unitarily invariant function* ∂ *on the appropriate Grassmannian.*

Output: *The two locations at which the data sets agree: at images* $I^1_{min(R_1)}$ *and* $I^1_{min(R_2)}$ *in* $\mathcal{C}_1$ *and at* $I^2_{ind(I^1_{min(R_1)})}$ *and* $I^2_{ind(I^1_{min(R_2)})}$ *in* $\mathcal{C}_2$.

Algorithm:

1. *Narrow the search to two regions in* $\mathcal{C}_1$ *in which the match can occur. Call these regions* R_1 *and* R_2.

2. *Find the two images in* $\mathcal{C}_1$ *at which the best match occurs. Simultaneously, find the images in* $\mathcal{C}_2$ *at which the best match is achieved. This part of the algorithm is achieved by the following steps:*

   (a) *To each image* I^1_j *of* $\mathcal{C}_1$, *associate a minimum distance to* $\mathcal{C}_2$. *Denote by* $ind(I^1_j)$, *the index of the image in* $\mathcal{C}_2$ *at which the minimum distance is achieved.*

   (b) *Find the image in* $\mathcal{C}_1|_{R_1}$ *that achieves the smallest distance to* $\mathcal{C}_2$. *Call this image* $I^1_{min(R_1)}$. *Similarly, find* $I^1_{min(R_2)}$, *the image in* $\mathcal{C}_1|_{R_2}$ *that achieves the smallest distance to* $\mathcal{C}_2$.

Step one of the algorithm requires narrowing the search to two regions in the first data set of images. That is, we find two ranges of images, which correspond to two regions of the circle of camera locations $\mathcal{C}_1$. This is done in the following way. Fix an image vector I^1_j in $\mathcal{C}_1$. Create a matrix $M(I^1_j)$ of size $m_1 \times 360$ by setting each column equal to I^1_j. We will use $\mathcal{F}(M(I^1_j))$ to compare images in $\mathcal{C}_1$ to rotations of images in $\mathcal{C}_2$. For each I^2_k, create a matrix of rotations $N(I^2_k)$ of size $m_1 \times 360$ by setting each column r equal to a flattened, masked rotation of the image I^2_k by r degrees, for r from 0 to 359. Then compute $\mathcal{F}(N(I^2_k))$. Now we define the distance d from I^1_j to the set $\mathcal{C}_2$ to be

$$d = \min\{d_k\}_{k \in \mathcal{C}_2},$$

where

$$d_k = \left|\left|\left|\mathcal{F}(M(I^1_j))\right| - \left|\mathcal{F}(N(I^2_k))\right|\right|\right|.$$

Letting j range over the image vectors in the data set on $\mathcal{C}_1$, we obtain a vector V whose j^{th} entry quantifies the closeness of the j^{th} point on $\mathcal{C}_1$ to the entire set $\mathcal{C}_2$. If the two data sets match well, we should see two regions R_1 and R_2 of V in which the distances between $\mathcal{C}_1$ and $\mathcal{C}_2$ are small relative to the other regions. These regions should also correspond to regions of the circle that are approximately 180 degrees apart since the intersection of two distinct great circles must occur at antipodal points.

Step two of the algorithm gives a means of determining which image in the region $\mathcal{C}_1|_{R_i}$ achieves the best match with an image in $\mathcal{C}_2$. The process is similar to the process in step one with two differences. First, we create $M(I_j^1)$ using rotations by every integer degree of I_j^1, instead of by repetitions of I_j^1. Second, we normalize the vectors $\mathcal{F}(M(I_j^1))$ and $\mathcal{F}(N(I_k^2))$ before computing the vector V. Once we have V, we find the locations that give a min in the regions R_1 and R_2. For each of these two locations, we also know the index k of the image in $\mathcal{C}_2$ which yields the closest point. Hence, we have found the two pairs of images where the data sets intersect.

8.2.3 Results

We show the result of step one of the algorithm using only the first Fourier coefficient in Figure 8.10. That is, we compute V using $\mathcal{F}_S(\cdot)$, where $S = \{1\}$. We get similar results using a larger set of coefficients, but we choose to use only the first coefficient because it gives very good results. Note that there are two very distinct regions R_1 and R_2 in which a match between data sets can occur. Also notice that these regions correspond to locations which are approximately 180 degrees apart.

In Figure 8.11, we show the vector V computed using $S = \{5, \ldots, 60\}$. That is, we use the 5^{th} to the 60^{th} normalized Fourier coefficients of the rotations of the images. Again, the choice of which Fourier coefficients to use is somewhat arbitrary. We try a number of different choices of S, and this particular choice seems to yield the best results.

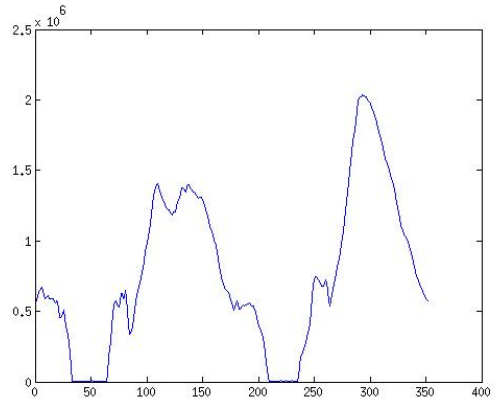


Figure 8.10: Step 1 - Plot of V : Min Distances from Points in $\mathcal{C}_1$ to the Set $\mathcal{C}_2$

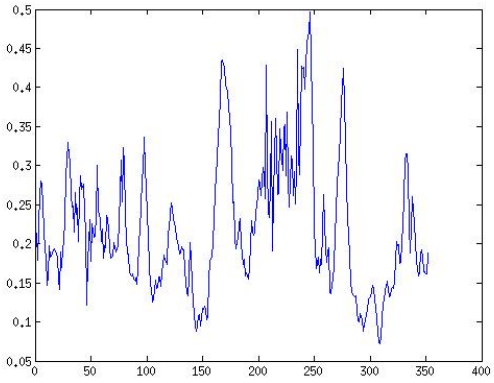
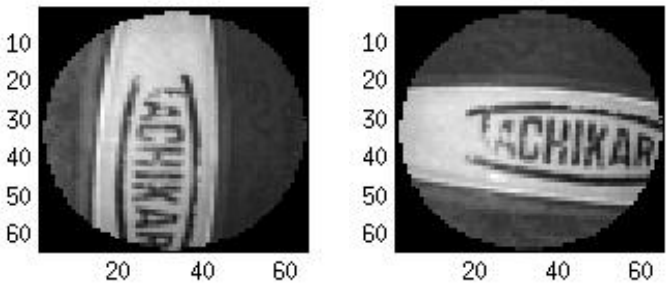


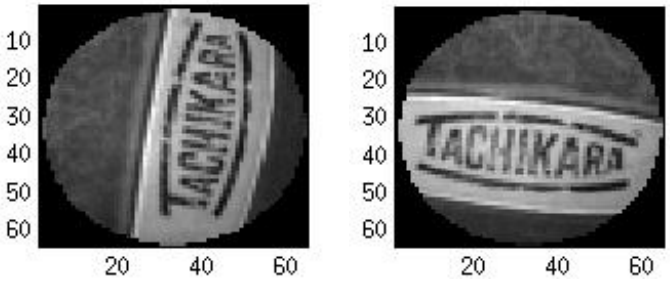
Figure 8.11: Step 2 - Plot of V : Min Distances from Points in $\mathcal{C}_1$ to the Set $\mathcal{C}_2$

In Figures 8.12 and 8.13, we show the original images from $\mathcal{C}_1$ and $\mathcal{C}_2$ that are given as the final output of the algorithm. In Figure 8.12, we see the match obtained in region R_1 of $\mathcal{C}_1$. Figure 8.13 shows the match in region R_2 of $\mathcal{C}_1$. Visual inspection of these results suggests that the algorithm was successful in finding the two points where the data sets agree.



(a) Min in $\mathcal{C}_1|_{R_1}$ (b) Corresponding Image in $\mathcal{C}_2$

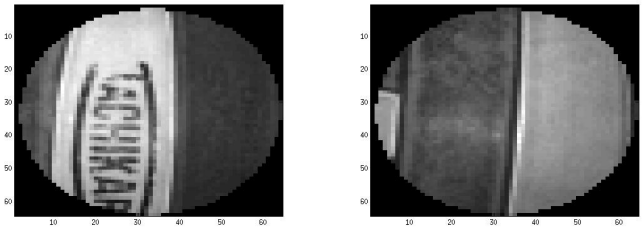
Figure 8.12: Intersection in R_1



(a) Min in $\mathcal{C}_1|_{R_2}$ (b) Corresponding Image in $\mathcal{C}_2$

Figure 8.13: Intersection in R_2

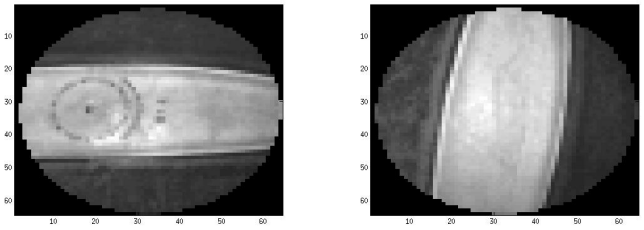
The volleyball data set gives a good demonstration of why both of the two steps of the algorithm described above are necessary. In Figure 8.14, we show the match determined by the minimum of the vector V that is the output at the end of step one. It shows that this step finds images which share some features but are not necessarily rotations of each other.



(a) Min in $\mathcal{C}_1|_{R_1}$ (b) Corresponding Image in $\mathcal{C}_2$

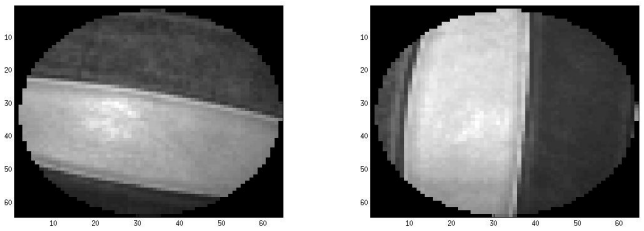
Figure 8.14: Intersection in R_1 : Step One Data Only

In Figures 8.15 and 8.16, we see the pairs of images that correspond to the two deepest minima achieved in the vector V that is the output at the end of step two. In other words, these are the two pairs that are found if there is no restriction to two regions R_1 and R_2, a priori. Notice that the connections between images in the two sets found using this method are quite strong. Unfortunately, they are due to symmetries of the object as opposed to the intersection of the two great circles. Hence, step one of the algorithm is necessary.



(a) Min in $\mathcal{C}_1$ (b) Corresponding Image in $\mathcal{C}_2$

Figure 8.15: Intersection: Step Two Data Only



(a) Second Best Min in $\mathcal{C}_1$ (b) Corresponding Image in $\mathcal{C}_2$

Figure 8.16: Intersection: Step Two Data Only

Conclusion

Pattern analysis is a field that continues to grow and change as technology and algorithms improve. Whenever data is collected, noise is introduced; we devote one chapter of this paper to this topic. The main contributions of this paper are the following: we apply Principal Component Analysis as a means of visualizing data, we present two new algorithms, and we define a function to measure the success of a data bundle approximation.

We find in Chapter 4 that noise can be introduced from the camera itself. Also, when data is collected under lighting that is powered by an alternating current, we find that a sufficiently high rate of image capture results in unintentional variation in the illumination conditions.

The main topic of study in this paper is data sets of images of a rotating object. Such data sets can be considered as sets that are invariant under an action of the special orthogonal group, $SO(2)$. We apply Principal Component Analysis to several such data sets in Chapter 5 in order to obtain visual and low-dimensional representations of the data. We find that the projection via Principal Component Analysis results in a closed loop, as expected. However, the nature of the curve varies significantly with changes in the object of study. In particular, while we have a map from S^1 into pixel space, it is not necessarily the case that the image of the projection will approximate a curve that is homeomorphic to S^1.

In Chapter 6, we consider the problem of approximating a data bundle over S^1 using a fixed number of points. The data sets being studied in this problem are

again thought of as sets that are invariant under an action of the special orthogonal group, $SO(2)$. We present an algorithm to find distributions of points on S^1 that are close to being (locally) optimal distributions in the sense that the associated subspace representations of the object are distributed with maximal distance between neighbors. We refer to this algorithm as the Nearest Neighbor Dispersion Algorithm.

In Chapter 7, we define a function that measures the success of an approximation of a data bundle. We apply this function to configurations obtained from an application of the Nearest Neighbor Dispersion Algorithm as well as to uniform and random configurations. This gives us a means of determining if and when the Nearest Neighbor Dispersion Algorithm improves upon alternative configurations. In addition, it gives a method for determining how many points in S^1 ought to be used once a distortion tolerance has been chosen.

Finally, in Chapter 8, we present an algorithm to find the approximate intersection of two data sets. The context for this problem is that two sets of images are collected along two distinct great circles on a sphere surrounding an object. By adding images to the data sets that correspond to rotation of the camera, we make the set of locations and orientations of the camera equal to $SO(3)$. Image collection therefore creates a map from $SO(3)$ into pixel space. Thus, the intersection of the data sets in pixel space is the intersection of the image of this map when applied to two two-dimensional objects in $SO(3)$. We exploit this fact in order to create an algorithm to find the two pairs of images that represent the intersection of the great circles. The visual results of the algorithm when applied to a data set of images of a volleyball suggest that the algorithm is successful in this case. We hope in the future to apply it to a variety of data sets to determine its robustness and to improve upon its efficiency.

In future projects, we would like to expand this work to be applied to data sets that are invariant under other groups. In particular, we hope to consider actions of the Euclidean group and/or the special Euclidean group. We also intend to improve upon the efficiency of the two algorithms. There are many possibilities for the def-

inition of the distortion function defined in Chapter 7. We would like to compare various definitions of a distortion function for the amount of information obtained via evaluation. Finally, in the setting of approximating data bundles, we would like to consider other algorithms and other base spaces, such as a higher dimensional sphere or a torus.

Bibliography

[1] Afriat, S. N. Orthogonal and oblique projectors and the characteristics of pairs of vector spaces. *Proceedings of the Cambridge Philosophical Society 53* (1957), 800–816.

[2] Axler, S. *Linear Algebra Done Right*, second ed., vol. 1. Springer-Verlag, 1997.

[3] Barg, A., and Nogin, D. Bounds on packings of spheres in the grassmann manifold. *IEEE Trans. Information Theory 48*, 9 (2002), 2450–2454.

[4] Basri, R., and Jacobs, D. Lambertian reflectance and linear subspaces. *PAMI 25*, 2 (2003), 218–233.

[5] Baumberg, A. Reliable feature matching across widely separated views. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 1* (2000), 1774.

[6] Belhumeur, P. N., and Kriegman, D. J. What is the set of images of an object under all possible illumination conditions? *International Journal of Computer Vision 28*, 3 (1998), 245–260.

[7] Beveridge, J., Draper, B., Chang, J.-M., Kirby, M., Kley, H., and Peterson, C. Principal angles separate subject illumination spaces in ydb and cmu-pie. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 2 (2009), 351–356.

[8] Bjorck, A., and Golub, G. H. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation 27*, 123 (1973), 579–594.

[9] Cangelosi, R., and Goriely, A. Component retention in principal component analysis with application to cdna microarray data. *Biology Direct 2*, 2 (2007).

[10] Chang, J.-M., Kirby, M., Kley, H., Peterson, C., Draper, B., and Beveridge, J. R. *LNCS 4844*. Springer Berlin / Heidelberg, 2007, ch. Recognition of Digital Images of the Human Face at Ultra Low Resolution Via Illumination Spaces, pp. 733–743.

[11] Chang, Y., Hu, C., Feris, R., and Turk, M. Manifold based analysis of facial expression. *Image and Vision Computing 24*, 6 (2006), 605–614.

[12] Conway, J., Hardin, R., and Sloane, N. Packing lines, planes, etc.: Packings in grassmannian spaces. *Experimental Mathematics 5* (1996), 139–159.

[13] Cox, D., Little, J., and O'Shea, D. *Using Algebraic Geometry*, vol. 185 of *Graduate Texts in Mathematics*. Springer-Verlag New York, Inc., 1998.

[14] Drmac, Z. On principal angles between subspaces of euclidean space. *SIAM Journal on Matrix Analysis and Applications 22* (2000), 173–194.

[15] Dummit, D. S., and Foote, R. M. *Abstract Algebra*. John Wiley and Sons, Inc., 2004.

[16] Edelman, A., Arias, T., and Smith, S. T. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl. 20* (1998), 303–353.

[17] Eisenbud, D. *The Geometry of Syzygies: A Second Course in Commutative Algebra and Algebraic Geometry*. Springer Science + Business Media, Inc., 2005.

[18] Eisenbud, D., and Harris, J. *The Geometry of Schemes*, vol. 197 of *Graduate Texts in Mathematics*. Springer-Verlag New York, Inc., 2000.

[19] Epstein, R., P.W., H., and Yuille, A. 5±2 eigenimages suffice: An empirical investigation of low-dimensional lighting models. *IEEE Workshop on Physics-Based Modeling in Computer Vision* (1995).

[20] Gibson, C. *Elementary Geometry of Algebraic Curves: An Undergraduate Introduction*. Cambridge University Press, 1998.

[21] Golub, G. H., and Van Loan, C. F. *Matrix Computations*, third ed. Johns Hopkins University Press, 1996.

[22] Golub, G. H., and Zha, H. Perturbation analysis of the canonical subspaces. *Linear Algebra and its Applications 210* (1994), 3–28.

[23] Griffiths, P., and Harris, J. *Principles of Algebraic Geometry*. Wiley & Sons, 1978.

[24] Hartshorne, R. *Deformation Theory*, vol. 257 of *Graduate Texts in Mathematics*. Springer New York Dordrecht Heidelberg London, 2010.

[25] Hassett, B. *Introduction to Algebraic Geometry*. Cambridge University Press, 2007.

[26] Horn, R. A., and Johnson, C. R. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[27] Hotelling, H. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology 24* (1933), 417–441.

[28] Jolliffe, I. *Principal Component Analysis*. Springer, 1986.

[29] Kara, S., Güven, A., and İçer, S. Classification of macular and optic nerve disease by principal component analysis. *Computers in Biology and Medicine 37*, 6 (2007), 836–841.

[30] Kim, T.-K., Arandjelovic, O., and Cipolla, R. Boosted manifold principal angles for image set-based recognition. *Pattern Recognition 40*, 9 (2007), 2475–2484.

[31] Kirby, M. *Geometric Data Analysis*. John Wiley & Sons, Inc., 2001.

[32] Li, Q., Zhang, Y., and Li, C.-K. Unitarily invariant metrics on the grassmannian space. *Electronic and Computer Engineering Journal* (2004).

[33] Liu, Z.-Y., Chiu, K.-C., and Xu, L. Improved system for object detection and star/galaxy classification via local subspace analysis. *Neural Networks 16*, 3-4 (2003), 437–451.

[34] Markoe, A. *Analytic Tomography*. Cambridge University Press, 2006.

[35] Matas, J., Burianek, J., and Kittler, J. *In Proc. BMVC*. 2000, ch. Object Recognition using the Invariant Pixel-Set Signature, pp. 606–615.

[36] Miao, J., and Ben-Israel, A. On principal angles between subspaces in $\mathbb{R}^n$. *Linear Algebra and its Applications 171* (1992), 81–98.

[37] Milnor, J. *Topology from the Differentiable Viewpoint*, revised ed. Princeton University Press, 1997.

[38] Okonek, C., Schneider, M., and Spindler, H. *Vector Bundles on Complex Projective Spaces*. Birkhäuser Boston, 1980.

[39] Pearson, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine 2* (1901), 559–572.

[40] Potier, J. L. *Lectures on Vector Bundles*. Cambridge University Press, 1997.

[41] Raychaudhuri, S., Stuart, J. M., and Altman, R. B. Principal components analysis to summarize microarray experiments: application to sporulation time series. *Pacific Symposium on Biocomputing* (2000), 455–466.

[42] Riklin-Raviv, T., and Shashua, A. The quotient image: Class-based rerendering and recognition with varying illuminations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23* (2001), 129–139.

[43] Schaffalitzky, F., and Zisserman, A. *ECCV 2002, LNCS 2350*. Springer Berlin Heidelberg, 2002, ch. Multi-view Matching for Unordered Image Sets, or "How Do I Organize My Holiday Snaps?", pp. 414–431.

[44] Schenck, H. *Computational Algebraic Geometry*, vol. 58 of *London Mathematical Society Student Texts*. Cambridge University Press, 2003.

[45] Shafarevich, I. R. *Basic Algebraic Geometry*, vol. 1-2. Springer-Verlag, 1994.

[46] Smith, K., Kahanpää, L., Kekäläinen, P., and Traves, W. *An Invitation to Algebraic Geometry*. Springer-Verlag New York, Inc., 2000.

[47] Steenrod, N. *The Topology of Fibre Bundles*, vol. 14 of *Princeton Mathematical Series*. Princeton University Press, 1951.

[48] Stewart, G., and Sun, J.-g. *Matrix Perturbation Theory*. Boston: Academic Press, 1990.

[49] Trefethen, L. N., and Bau, III, D. *Numerical Linear Algebra*. SIAM, 1997.

[50] Vailaya, A., Jain, A. K., and Zhang, H. J. On image classification: city images vs. landscapes. *Pattern Recognition 31*, 12 (1998), 1921–1935.

[51] Von Neumann, J. Some matrix-inequalities and metrization of matrix-space. *Tomsk Univ. Rev. 1* (1937), 286–300.

[52] Wang, T., and Shi, P. Kernel grassmannian distances and discriminant analysis for face recognition from image sets. *Pattern Recognition Letters 30*, 13 (2009), 1161–1165.

[53] Wei, M., and De Pierro, A. R. Perturbation analysis of the canonical subspaces. *Linear Algebra and its Applications 279* (1998), 135–151.

Appendix A

Matlab Code for Implementation of Algorithms

A.1 Nearest Neighbor Dispersion Algorithm – Local

A.1.1 Main Program

```
% INPUTS:
% ColsofU -- a matrix of size
%   (#pixels)x(rank of vector bundle)x(#possible locations)
%   each choice of 3rd coordinate gives a vector space for
%   that location on S^1.
% n -- # of camera locations to distribute.
% mchoice -- choice of distance measure.
%   Options are 1, 2, 3, 4, or 5, corresponding to
%   Fubini-Study, Geodesic, Chordal, Subspace, and
%   Smallest principal angle, respectively.
% numsteps -- # of iterations to go through PER STEPSIZE.
```

```
%    Usually 5000.
% vbrank -- rank of vector bundle.
% initialstepsize -- initial jump size. It represents the
%    number of spaces a point can jump in the first set of
%    steps. Spaces have size 2*pi/numpics. Usually start with
%    initialstepsize=30.

% OUTPUTS:
% thetasfinal -- angles determining locations of points in
%    final configuration.
% Points -- complex numbers giving point locations on unit
%    circle in final configuration.

numpics=size(ColsofU,3); %number of pictures in the data set
%    determines the number of possible camera locations in the
%    configuration.

% Put n points on the sphere in a random distribution:
thetasinds=randperm(numpics);
thetasinds=sort(thetasinds(1:n));
dt=2*pi/numpics;
thetas=thetasinds*dt;

% thetasinds is an nx1 vector of random locations on the
% circle. The locations are given by indices in [1,numpics],
% each of which corresponds to an angle: index*2pi/numpics.

Points=exp(1i*thetas); %nx1 vector of complex numbers on S^1.
```

```
OriginalPoints=Points;

for k=[initialstepsize:-5:10 9:-1:1]
    stepsize=k;
    for j=1:numsteps
%       Pick a point at random:
        ptind=floor(rand*(n-0.000001))+1;
%          random index -- an integer in [1,n]
        ind=thetasinds(ptind);
%           ind is the location of the randomly chosen point.

        Q=ColsofU(:,:,ind); %space corresponding to ind.

%       Find indices of two neighbors of ind:
        if ptind==1
            near1=n;
        else
            near1=ptind-1;
        end
        if ptind==n
            near2=1;
        else
            near2=ptind+1;
        end
        near1=thetasinds(near1);
        near2=thetasinds(near2);

        Q1=ColsofU(:,:,near1);
```

```
        Q2=ColsofU(:,:,near2);

        S1=svd(Q1'*Q);
        S2=svd(Q2'*Q);
%       S1 and S2 contain the principal angles that we will
%       use to measure the distance between the vector space
%       at ind and those at the neighboring points.

        distance1=basicdistance(S1,mchoice);
        distance2=basicdistance(S2,mchoice);
%           basicdistance function uses one of five choices
%           for distance measure to calculate distance based
%           on principal angles.

        [mdist,indexmin]=min([distance1 distance2]);
        if indexmin==1
            indmin=near1;
            otherind=near2;
        else
            indmin=near2;
            otherind=near1;
        end

        thetasinds(ptind)=findnewindex(ind,indmin,otherind,...
            ColsofU,stepsize,numpics,mchoice);
%           findnewindex function determines what the new
%           location of ind should be.
```

```
        thetasinds=sort(thetasinds);
    end
end
thetasfinal=thetasinds*dt;
Points=exp(1i*thetasfinal);
t=linspace(0,2*pi);
figure;hold on;
plot(OriginalPoints,'+')
plot(exp(1i*t))
axis square
plot(Points,'ro');
```

A.1.2 Basic Distance Function

```
function distance=basicdistance(S,mchoice)

% INPUTS:
% S -- a matrix of singular values.
% mchoice -- a choice of distance measure.

% OUTPUT:
% distance -- the evaluation of the distance measure
%   with inputs given by the principal angles on the
%   diagonal of S.

% Occasionally, numerical calculations result in singular
% values that are slightly larger than one, which leads to
% imaginary distances. We force such singular values to be
% equal to one.
```

```
nangles=size(S,1);
for j=1:nangles
    if S(j)>1 && abs(S(j)-1)<10^(-4)
        S(j)=1;
    end
%   If the singular values are significantly larger than one,
%   we break the program and debug.
    if S(j)>1
        disp('One or more singular value is > 1.')
        return
    end
end

if mchoice==1
    temp=1;
else
    temp=zeros(1,nangles);
end
for j=1:nangles
    if mchoice==1
        temp=temp*S(j);
    elseif mchoice==2
        temp(j)=acos(S(j));
    elseif mchoice == 3
        temp(j)=sin(acos(S(j)));
    elseif mchoice==4
        temp(j)=sin(acos(S(j)));
    end
```

```
end

if mchoice==1 %Fubini-study
    distance=acos(temp);
elseif mchoice==2 %Geodesic
    distance=sqrt(sum(temp.^2));
elseif mchoice==3 %Chordal
    distance=sqrt(sum(temp.^2));
elseif mchoice==4 %Subspace
    distance=max(temp);
else %smallest principal angle only
    distance=acos(S(1));
end
```

A.1.3 Find New Index Function

```
function newthetasind=findnewindex(ind,indmin,otherind,...
    ColsofU,stepsize,numpics,mchoice)

% INPUTS:
% ind -- index in [1,numpics] of point to be moved.
% indmin -- index of closest neighboring point.
% otherind -- index of farthest neighboring point.
% ColsofU -- matrix of vector spaces attached at possible
%   point locations.
% stepsize -- jump size allowed for the given iteration.
% numpics -- # of pictures in the data set.
% mchoice -- choice of distance measure.
%   Options are 1, 2, 3, 4, or 5, corresponding to
```

```
%    Fubini-Study, Geodesic, Chordal, Subspace, and
%    Smallest principal angle, respectively.

% OUTPUT:
% newtheatsind -- new location of ind.
%    New location is determined by moving ind in both
%    possible directions, then determining which move yields a
%    greater distance with indmin. The two moves are as far as
%    possible in the given direction without jumping over or
%    landing on the nearest point.

Listofinds=sort([ind,indmin,otherind]);

%Cases in which the point has no space to move:
if ind==1
    if Listofinds(2)==2 && Listofinds(3)==numpics
        newthetasind=ind;
        return
    end
elseif ind==numpics
    if Listofinds(1)==1 && Listofinds(2)==numpics-1
        newthetasind=ind;
        return
    end
else
    if ind-1==Listofinds(1) && ind+1==Listofinds(3)
        newthetasind=ind;
        return
```

```
    end
end


if ind==Listofinds(1)
    newth1=ind+stepsize;
    while newth1>=Listofinds(2) %ind has jumped over another
        %point. Need to move it back.
        newth1=newth1-1;
    end
    newth2=ind-stepsize;
    if newth2<=0 %Need index to wraparound.
        newth2=newth2+numpics;
        while newth2<=Listofinds(3) %ind has jumped.
            newth2=newth2+1;
        end
        if newth2==numpics+1
            newth2=1;
        end
    end %If no wraparound, then newth2 cannot have jumped.
elseif ind==Listofinds(3)
    newth1=ind+stepsize;
    if newth1>numpics
        newth1=newth1-numpics;
        while newth1>=Listofinds(1)
            newth1=newth1-1;
        end
        if newth1==0
            newth1=numpics;
```

```
        end
    end %If no wraparound, then newth1 cannot have jumped.
    newth2=ind-stepsize;
    while newth2<=Listofinds(2)
        newth2=newth2+1;
    end
else %ind==Listofinds(2)
    newth1=ind+stepsize;
    while newth1>=Listofinds(3)
        newth1=newth1-1;
    end
    newth2=ind-stepsize;
    while newth2<=Listofinds(1)
        newth2=newth2+1;
    end
end

%If there's only one choice for which way to go, pick that:
if newth1==ind
    newthetasind=newth2;
    return
end
if newth2==ind
    newthetasind=newth1;
    return
end

Q1=ColsofU(:,:,newth1);
```

```
Q2=ColsofU(:,:,newth2);
Qindmin=ColsofU(:,:,indmin);
S1=svd(Q1'*Qindmin);
S2=svd(Q2'*Qindmin);
distance1=basicdistance(S1,mchoice);
distance2=basicdistance(S2,mchoice);

[mdist,indmdist]=min([distance1,distance2]);

if indmdist==1
    newthetasind=newth2;
else
    newthetasind=newth1;
end
```

A.2 Intersection of Data Sets

A.2.1 Step 1

```
% INPUTS:
% DMa -- a data matrix, each of whose columns is the first
%    181 Fourier coefficients for one image from data set a.
% DMb -- a data matrix, each of whose columns is the first
%    181 Fourier coefficients generated from all integer
%    degree rotations of an image from data set b.
% range2use -- Fourier coefficients to use for each image.
%    usually equal to 1.

ndatapts1=size(DMa,2);
```

```
ndatapts2=size(DMb,2);

Minloc=zeros(1,ndatapts1);
MinDists=zeros(1,ndatapts1);
ComparisonDists=zeros(ndatapts1,ndatapts2);

for j=1:ndatapts1
    for k=1:ndatapts2
        ComparisonDists(j,k)=norm(abs(DMa(range2use,j))-...
            abs(DMb(range2use,k)));
    end
    [MinDists(j) Minloc(j)]=min(ComparisonDists(j,:));
end

figure;plot(MinDists)
```

A.2.2 Step 2

```
% INPUTS:
% Ca -- data set a. Ca is a 4-D matrix of images collected
%   along great circle C1.
% Cb -- data set b. Cb is a 4-D matrix of images collected
%   along great circle C2 (distinct from C1).
% DMa -- a data matrix, each of whose columns is the first
%   181 Fourier coefficients generated from all integer
%   degree rotations of an image from data set a.
% DMb -- a data matrix, each of whose columns is the first
%   181 Fourier coefficients generated from all integer
%   degree rotations of an image from data set b.
```

```
% range2use -- Fourier coefficients to use for each image.
%   usually equal to 5:60.
% rangeformin1 -- determined from step 1 of the algorithm.
%   In this example, it is 30:75.
% rangeformin2 -- determined from step 1 of the algorithm.
%   In this example, it is 200:240.

ndatapts1=size(DMa,2);
ndatapts2=size(DMb,2);

Minloc=zeros(1,ndatapts1);
MinDists=zeros(1,ndatapts1);
ComparisonDists=zeros(ndatapts1,ndatapts2);

% Normalize Fourier coefficients to be used:
for j=1:ndatapts1
    DMa(range2use,j)=abs(DMa(range2use,j))./ ...
        norm(abs(DMa(range2use,j)));
end
for j=1:ndatapts2
    DMb(range2use,j)=abs(DMb(range2use,j))./ ...
        norm(abs(DMb(range2use,j)));
end

% Calculate min distance from each image in data set a to
%   data set b.
for j=1:ndatapts1
    for k=1:ndatapts2
```

```
            ComparisonDists(j,k)=norm(abs(DMa(range2use,j))-...
                abs(DMb(range2use,k)));
        end
        [MinDists(j) Minloc(j)]=min(ComparisonDists(j,:));
end

% Find minima in a allowed regions:
[a,b]=min(MinDists(rangeformin1));
[c,d]=min(MinDists(rangeformin2));

% Plot two pairs of images corresponding to intersection
%   of data sets: (plotpics is a function created for
%   this purpose.
plotpics(b+rangeformin1(1)-1,Minloc,Ca,Cb,Mask)
plotpics(d+rangeformin2(1)-1,Minloc,Ca,Cb,Mask)
```