Investigating the Auto-Focusing problem using Locality Preserving Projections (LPP): Can LPP be used to determine the best focused image from a sequence of incrementally focused images?

by **Richard Muscat**

7th May 2012

Supervisor: **Professor Ing. Kenneth P Camilleri**

Submitted as part of the requirements for the award of the Degree in Computing and Information Systems of the University of London.

UNIVERSITY OF LONDON

BSC IN COMPUTING AND RELATED SUBJECTS FOR

EXTERNAL STUDENTS CIS320 PROJECT SUBMISSION FORM

A copy of this form (or a typed or computer-generated version) must be completed by each student and attached to each project report that is submitted to the University.

The report must be submitted in time to be received by the University before 15 May in the year of the examination.

Full name:Richard Muscat

(*as it appears on your Registration Form*)

Student number:040333821

Project title: Investigating the Auto-Focusing problem using Locality Preserving Projections (LPP): Can LPP be used to determine the best focused image from a sequence of incrementally focused images?

DECLARATION

I declare that:

- I understand what is meant by plagiarism.
- I understand the implications of plagiarism.
- This project report is all my own work and I have acknowledged any use of the published or unpublished works of other people.

Signature …….. **Date** 14 Jan 2012

Acknowledgement

I would like to thank all those who had supported me in some way throughout the course of the project.

First of all, I would like to thank my supervisor Professor Ing. Kenneth P. Camilleri for his guidance and support throughout all stages of the project.

I would also like to thank my wife for her continuous support.

Finally I would like to thank my parents for their support and encouragement throughout the years.

Contents

Abstract

A digital camera system provides image information and therefore auto-focusing is a classic problem in such system. Image data is high dimensional. Locality Preserving Projections (LPP) reduces the data dimensionality while preserving the local similarity in data. Since a focusing sequence is a sequence of similar images, LPP may be used to preserve some important information related to the focusing problem. This dissertation investigates in what way can LPP represent the auto-focusing problem and whether such approach can give a new insight to such problem. This study proceeds to discuss a novel method of how can the focused image be identified from applying LPP on difference in pixel intensity across an edge. Results showed that when LPP is applied on edge strength data it yields a consistent representation of the focused image.

CHAPTER 1

Terms of Reference

1. Terms of Reference

1.1 Objectives

As described in the PDF (refer to Appendix E) the main question that this project is set to answer is whether *Locality Preserving Projections (LPP)* (He & Niyogi, 2003) can aid in addressing the auto-focusing problem. Images in a focusing sequence have a certain locality in the data because they are sequentially achieved through the focusing process. Then there is the question of what is the underlying dimensionality factor of such data which has a strong local colony. Since LPP algorithm has the aspect of representing data on a low dimensional space while preserving local data similarity, could this approach have a new insight into the auto-focusing problem?

The first step in this investigation was to review literature on auto-focusing to place the study in context. The next step was to get familiar with LPP as to proceed with the investigation. As it was pre-specified in the PDF, the original objectives were:

- How does LPP behave on focusing image sequences?
- What is the effect of image content on LPP-transformed data?
- How does LPP behave on difference in pixels across edges?

Moreover, throughout the course of investigation further objectives evolved. These include:

- In what way can the focused image be identified in a low-dimensional representation resulting from LPP?
- What pre-processing would be required?
- What is the sufficient dimensionality to represent such problem?
- Can LPP be generalized?

The investigation progressed systematically in addressing the above questions.

The key milestones in the project were more or less achieved as originally planned in the PDF. However, as listed above, further objectives evolved and had generated the need to add some further key milestones in the work plan. These included:

- Automate detection of focused image (February)
- Automate detection and selection of edges (beginning of March)
- Experiment on generalizability of LPP on pixel difference across edges (end of March/beginning of April)

1.2 Structure of the report

The report comprises five main chapters. This chapter introduces the report and provides a high-level overview of this study. Chapter 2 includes a literature review on passive auto-focusing techniques and background theory on LPP, which lead to experiments methodology. The latter part involves designed algorithms and the *Experimental Design* throughout the investigation. Chapter 3 reports results and analysis of each experiment. Moreover there is an Appendix illustrating the *Focusing Image Sequences* used and another Appendix providing further detailed *results*. MATLAB code and processed datasets in MATLAB format are available in the compact disc attached. There is also an Appendix describing the contents in the disc and how these relate to the results obtained. Another Appendix provides MATLAB code listing. Results are further discussed in the Chapter 4. Finally the *Conclusion* in Chapter 5 goes through this study and provides recommendations for future work.

CHAPTER 2

Methods

2. Methods

2.1 Literature Review

Auto-focusing is one of the most important aspects of a digital still camera. In active auto-focusing the distance of an object is determined by directing a beam of energy towards the subject. The detection sensor receives the reflected beam and the system adjusts the lens position accordingly. While active focusing relies on more hardware to determine distance, passive focusing determines such distance using passive computer analysis. This study is concerned with the analysis of images under different focusing conditions and is thus based on a passive approach. Therefore related work on passive autofocusing methods is reviewed here to place our study in the context of autofocusing research.

According to Subbarao et al (1993), most passive autofocus techniques define a focus measure. Typically, as the lens moves from one end towards the other such focus measure gradually increases, reaches a maximum at the focused lens position and then gradually decreases. The problem is to find the position of the lens which has the maximum focus.

Depth From Focus (DFF) and *Depth From Defocus* (DFD) are auto-focusing technologies related with obtaining depth information by controlling camera parameters. DFF computes the distances to points in a scene by acquiring many images using different focusing parameters. Then the best focusing parameters are searched by using a focus measure. Prior research on DFF was concerned with the development and evaluation of different focus measures. Subbarao et al (1993) provided a theory for evaluating several focus measures based on the *Optical Transfer Function* (OTF). According to Ens and Lawrence (1993) the calculation of DFF involves deconvolving the defocus operator from the image and modelling it. Many techniques for determining the defocus operator use inverse filtering. Ens and

Lawrence (1993) present a general matrix method using regularization to eliminate some fundamental problems with inverse filtering such as inaccuracies in finding the frequency domain representation, windowing and border effects. Xiong & Shafer (1993) state that the main cause of inaccuracy in focusing is the local maxima problem (mostly due to noise). They address such problem by proposing a search algorithm which is a combination of Fibonacci search and curve fitting. In view of the various focus measures developed Subbarao & Tyan (1998) described a method to select an optimal focus measure for auto-focusing and DFF. They define namely, Auto-focusing Uncertainty Measure (AUM) and Auto-focusing Root Mean Squared Error (ARMS) to estimate the noise sensitivity of various focus measures. Since camera parameters are changed frequently and multiple images need to be acquired, DFF methods are generally time-consuming.

In *Depth from Defocus* (DFD) methods, the depth is calculated by estimating the degree of image blur. In contrast to DFF, DFD requires only two to four images to be obtained and processed. DFD approaches may be classified as frequency domain, spatial domain, and statistical approaches.

Pentland (1987) compared two images, one was acquired by a pinhole camera whereas the other one was acquired by using a wide aperture camera. An inverse filtering method was then used to recover depth. Subbarao & Wei (1992) presented a new method for finding depth from image defocus and rapid autofocusing of a camera. This method is based on computing only one-dimensional Fourier coefficients, and hence it gains more computational efficiency. Xiong & Shafer (1993) proposed an iterative blur estimation method to reduce the window effect. Xian & Subbarao (2005) states that frequency domain approaches require more computation when compared to spatial domain approaches. Statistical methods generally require even higher computational costs as they generally involve optimization operation.

According to Xian & Subbarao (2005), spatial domain approaches preserve the spatial correspondence information. Consequently they are more suitable for applications such as continuous focusing and object tracking. Subbarao (1991) defined a spatial-domain convolution/deconvolution transform for n-dimensional signals. Subbarao & Surya (1994) proposed a novel method called STM for estimating the distance to objects in spatial domain. There are two varations of STM. STM1 changes the lens postion whereas STM2 only changes the aperture diameter.

The image-based auto-focusing techniques reviewed above involved computing depth information and searching the focused image position. A sequence of incrementally focused images contains data similarity, and the best focused image has its position in such sequence. Applying Locality Preserving Projections (LPP) would reduce the dimensionality while preserving the locality of such data. This study is mainly concerned with whether the focused image position can be searched in a low-dimensional representation. The next section provides background theory of LPP.

2.2 Locality Preserving Projections

He (2005) states that Locality Preserving Projections (LPP) are "linear projective maps that arise by solving a variational problem that optimally preserves the neighbourhood structure of the data set."

2.2.1 Linear Dimensionality Reduction

Image data generally has a high dimensionality. Dimensionality reduction transforms data from high-dimensional to a meaningful low-dimensional representation. Such representation should ideally comprise the underlying information within the data. Working with low-dimensional data would therefore result in more efficient analysis and processing of such data. Two traditional and popular methods for linear

dimensionality reduction are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

2.2.1.1 The problem of linear dimensionality reduction

Given a set $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m$ in $\mathbf{R}^n$, we need to find a transformation matrix A that maps these high dimensional vectors $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m$ in $\mathbf{R}^n$ to a set of low dimensional points $\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_m$ in $\mathbf{R}^l$ ($l \ll n$), such that data point $\mathbf{y}_i$ represents $\mathbf{x}_i$, and $\mathbf{y}_i = A^T\mathbf{x}_i$

(Equation 1)

2.2.1.2 Traditional Linear Techniques for Dimensionality Reduction

Principal Component Analysis (PCA) seeks a low dimensional projection that best preserves variance in the high dimensional data. By minimizing the following objective function,

$$\min_{\mathrm{a}} \sum_{i=1}^{m} |\mathbf{x_i} - \mathbf{a}\mathbf{a}^{\mathrm{T}}\mathbf{x_i}|^2$$

The set of n-dimensional orthonormal principal vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_\mathrm{n}$ are found which represent the eigenvectors of the sample covariance matrix. Some criterion may be used to select the $l \ll n$ principal vector associated to l larger eigenvalues (He, 2005). The transformation matrix A^T (refer to Equation1) will contain these selected principal vectors in its rows.

Another popular linear projection technique is Linear Discriminant Analysis (LDA). LDA seeks a low dimensional projection that best preserves the class discriminatory information in the high dimensional data. If we consider a set of high dimensional vectors $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m$ belonging to h classes of that high dimensional data, then a

transformation is found by maximising the ratio of the between-class scatter, S_b, and the within-class scatter, S_w, that is,

$$\max_{\mathrm{a}} \frac{\mathbf{a}^{\mathrm{T}} S_b \mathbf{a}}{\mathbf{a}^{\mathrm{T}} S_w \mathbf{a}}$$

$$S_{\mathrm{b}} = \sum_{i=1}^{h} |\mathrm{C_i}| (\boldsymbol{\mu}^{\mathrm{i}} - \boldsymbol{\mu})(\boldsymbol{\mu}^{\mathrm{i}} - \boldsymbol{\mu})^{\mathrm{T}}$$

$$S_{\mathrm{w}} = \sum_{i=1}^{h} |\mathrm{C_i}| \mathrm{E}\left[(\mathbf{x}^{\mathrm{i}} - \boldsymbol{\mu})(\mathbf{x}^{\mathrm{i}} - \boldsymbol{\mu})^{\mathrm{T}} \right]$$

where $\boldsymbol{\mu}$ is the total sample mean vector, $|\mathrm{C_i}|$ is the number of samples in class $\mathrm{C_i}$, $\boldsymbol{\mu}^{\mathrm{i}}$ are the average vectors of $\mathrm{C_i}$, and $\mathbf{x}^{\mathrm{i}}$ are the sample vectors associated to $\mathrm{C_i}$. The selected vectors $\mathbf{a_i}$ are set in the rows of transformation matrix A^T (refer to Equation 1) (He, 2005).

2.2.1.3 Limitations of PCA and LDA

PCA is sensitive to outliers and unable to represent the topological structure of data. Moreover PCA has limited discriminating power. LDA is robust to discriminate between different classes of data. However LDA like PCA fails to preserve the local information that lies within data. LPP handles better such limitations. It is more capable to discriminate between different classes of data than PCA. Moreover, since LPP is concerned with preserving the local information that lies within data, it is less sensitive to outliers (He, 2005). The data within an auto-focusing sequence contains a strong topological structure. Therefore LPP is more likely to be capable to represent such problem.

2.2.2 Algorithms

2.2.2.1 Locality Preserving Projections

This subsection describes the algorithm of Locality Preserving Projections (LPP) as presented by He & Niyogi (2003).

With reference to the problem of linear dimensionality reduction described in section 2.2.1.1, the algorithm proceeds as follows:

Constructing adjacency graph

The adjacency graph G, has m nodes where each node g_i represents image vector $\mathbf{x}_i$. An edge is created between nodes *i* and *j* if $\mathbf{x}_i$ and $\mathbf{x}_j$ are *close*. For the majority of the experiments the *k nearest neighbours* [parameter $k \epsilon \boldsymbol{N}$] were selected as those being *close*. Experimental results were also obtained where the adjacency graph was constructed by creating an edge between consecutive images in a focusing sequence.

The weight matrix

Edges are weighted in a sparse symmetric $m \times m$ matrix W where W_{ij} has the weight of the edge connecting vertices *i* and *j* if and only if such edge exists. Otherwise W_{ij} is 0.Weighting for the edge joining vertices *i* and *j* is done either by heat kernel [parameter $t \epsilon \mathbf{R}$], that is:

$$W_{ij} = e - \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{t}$$

or through a Simple-minded approach such that

$$W_{ij} = 1$$

Computing the eigenvectors and eigenvalues

Finally the algorithm computes the eigenvectors and eigenvalues for the generalized eigenvector problem:

$$XLX^T\mathbf{a} = \lambda XDX^T\mathbf{a} \qquad \textbf{(}\text{Equation 2)}$$

such that D is a diagonal matrix where its entries are row or column sums of W (since W is symmetric), $D_{ii} = \sum_j W_{ji}$. $L = D - W$ is the Laplacian matrix. Column i of matrix X is $\mathbf{x}_i$.

The solutions to Equation 2, $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ are vectors ordered according to their eigenvalues $\lambda_1 < \cdots < \lambda_n$. The transformation matrix A^T (refer to Equation 1) will contain such vectors in its rows. The l most significant eigenvectors are retained neglecting the rest.

How LPP is applied on the auto-focusing problem

To represent the auto-focusing problem LPP is applied on the difference in pixel intensity across an edge. Such edge difference values are used as input data to the LPP algorithm. Consider a set of images in a focusing image sequence. The data for each image i is organised in a vector $\mathbf{x}_i$ where $\mathbf{x}$ is a vector containing the edge difference values representing the image, and i denotes its index as ordered in the focusing sequence. Therefore given a focusing sequence as set of high dimensional vectors $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m$ in $\mathbf{R}^n$, LPP transforms such data giving us a set of low dimensional data-points $\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_m$ in $\mathbf{R}^l$ such that low dimensional data-point $\mathbf{y}_i$ represents image i.

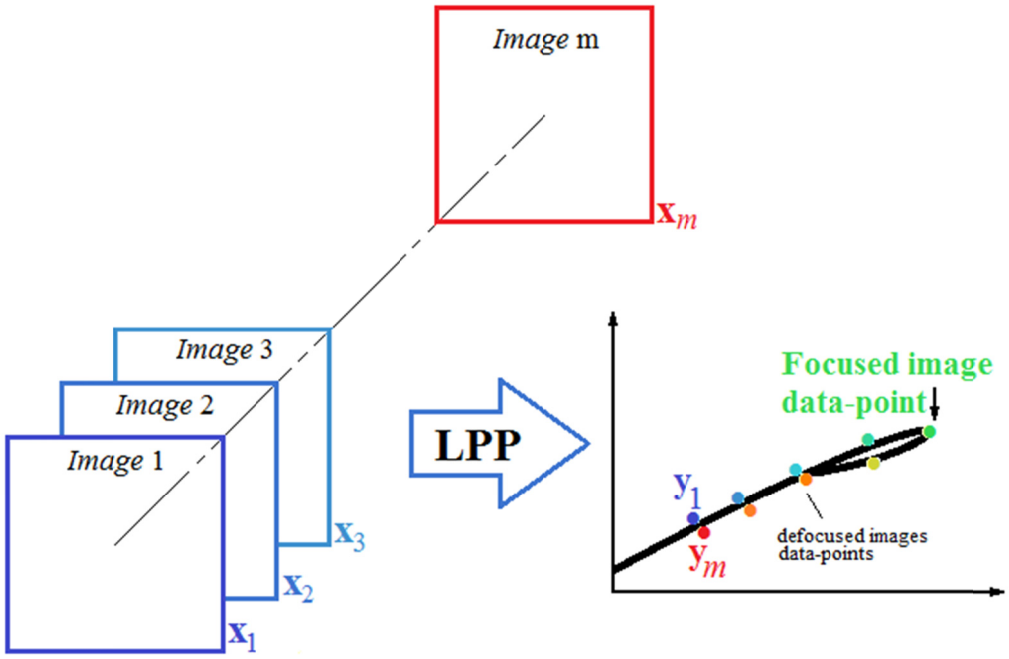


Figure 2.1 – The relation between images in a focusing sequence, input vectors and an expected two dimensional representation resulting from LPP.

When plotting these low dimensional data-points we expect to get a trajectory that starts from the first image data-point $\mathbf{y}_1$, moves towards the focused image data-point and at such point it changes direction and goes back until it reaches the last image data-point $\mathbf{y}_m$ (refer to Figure 2.1). This means that the defocused image data-points prior to focusing are somehow mapped close to those defocused image data-points subsequent to focusing. We expect this to happen because the defocused images prior to focusing are somehow similar to defocused images after focusing. Since LPP preserves local the local structure of data, it would map data-points representing similar data close to each other. Consequently a turning point occurs at the focused image data-point. Experimental results had in fact confirmed such expectations. (refer to section 3.3).

2.2.2.2 Pre-processing - Automating edge detection and selection

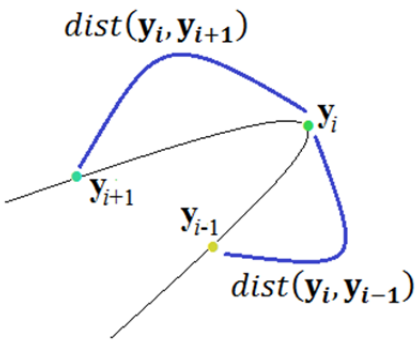
Since the high dimensional data input into LPP consists of edge difference values, some pre-processing is required. Therefore an algorithm to automate the selection of edges was designed. Given a focusing sequence of images in grayscale, this first involved computing the mean image out of such set of images. Then using the Sobel operator, edge detection was applied on the mean image. Ideally edge detection would be run on a focused image but at this stage such information is not known. Our scope is to select one strong edge. As to detect strong edges, and hence eliminate noise and many unwanted edges, a threshold value is specified when detecting edges through the Sobel operator. The command in MATLAB concerning edge detection returns a binary image as an output, with 1's where the function finds edges and 0's elsewhere. A search for the longest vertical edge within the binary image is conducted and such edge is selected. For the scope of this study, only vertical edges are being considered as to keep it simple. However what can be done on a vertical edge may be done on any edge, by applying the same principle. When the longest vertical edge has been selected, edge difference values are computed constituting a dataset to be processed by LPP. Such dataset is scaled to the range of [0,1] (by dividing all values by the maximum value).

Section 3.3.4 demonstrates the application of the algorithm described above on two focusing image sequences. The implementation in MATLAB is available in Appendix D.3 and in the compact disc attached (Refer to Appendix C)

2.2.2.3 Post-processing - Detection of focused image

Recall that a turning point occurs at the focused image data-point in LPP low dimensional representation. Based on the observations in section 3.3.3 a simple algorithm to detect the focused image from the output of LPP was designed. This consisted in two stages. In the first stage, the algorithm selects the data-points in which a change in direction is detected in either the first or second dimension of LPP.

In the second stage, from the selected data-points, it outputs the index *i* of the data-point $\mathbf{y}_i$ having the maximum total Manhattan distance *td*, such that:



$$td(\mathbf{y}_i) = dist(\mathbf{y}_i, \mathbf{y}_{i+1}) + dist(\mathbf{y}_i, \mathbf{y}_{i-1}) \quad \text{and}$$

$$dist(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{n} |\mathbf{a}_j - \mathbf{b}_j|$$

where: *j* is the dimension index;
n = 3 (thus considering Manhattan distance in 3-dimensions)

Figure 2.2 – Total Manhattan distance

The implementation in MATLAB is available in Appendix D.2 and in the compact disc attached (Refer to Appendix C)

2.3 Experimental Design

2.3.1 Behaviour of LPP on images

Throughout these experiments several approaches for applying LPP on image focusing sequences were analysed. This means that vector $\mathbf{x}_i$ comprises pixel intensity values for grayscale image *i*. The image index *i* is ordered sequentially according to the focusing sequence. Experiments mainly involved different methods for constructing the adjacency graph combined with the two ways of weighting the edges (heat kernel and Simple-minded approach). The methods for constructing the adjacency graph involved:

1. *k nearest neighbours* [parameter $k \epsilon \boldsymbol{N}$];
2. Creating an edge between consecutive images in the focusing sequence
3. Combination of 1 and 2 above.

While *k nearest neighbours* relies on the content within the data to construct the adjacency graph, the approach in 2 above imposes physical adjacency, that is, every image data-point is forced to be adjacent with the next.

2.3.2 Effect of image content on LPP-transformed data

2.3.2.1 Two-Dimensional LPP-transformed data

LPP will always discard something from the data. Hence there's always an error (difference) between original data and low-dimensional data. The amount of detail increases as the image focuses. Therefore it was interesting to analyse whether some form of relationship exists between focusing and the amount of error due to dimensionality reduction. This was investigated by applying LPP on images and reconstructing back the high-dimensional image data from the LPP low-dimensional data. One expects the focused image to have the highest error since a lot of detail resides there.

The following procedure was applied:

First LPP was applied on the data. Then using the first two dimensions of LPP, high dimensional data was reconstructed from the low dimensional data. This was achieved by the product of low dimensional data and the pseudo-inverse of the transformation matrix A. The root mean squared error (RMSE) was then calculated from the difference between the original data and the reconstructed data. Results consisted of plots of images versus RMSE on LPP-transformed data.

2.3.2.2 A range of dimensions for LPP-transformed data

The experiment in 2.3.2.1 was repeated for a range of low dimensions i.e. from 1 to 30. The scope was to analyse further what happens on the focused image and to observe the relationship between RMSE and the amount of dimensions considered

for LPP-transformed data. The expectation here was that the error is reduced gradually as more dimensions are considered.

2.3.3 Behaviour of LPP on difference in pixels across an edge

A focused image is an image which has a high contrast and therefore its edges are sharp. Edges become less sharp as the image starts to be defocused. Therefore the difference in pixels across an edge is at its maximum when an image is focused. Such difference starts decreasing as the image defocuses. Here we investigate the behaviour of LPP on pixel difference across the edge.

More specifically the intensity of each pixel at a position of $+i\delta$ from the edge position along the edge normal is subtracted from the intensity of the corresponding pixel at a position of $-i\delta$.

With reference to Figure 2.3 below, consider M pixels along an edge where each pixel position is z_q ; $1 \leq q \leq$ M

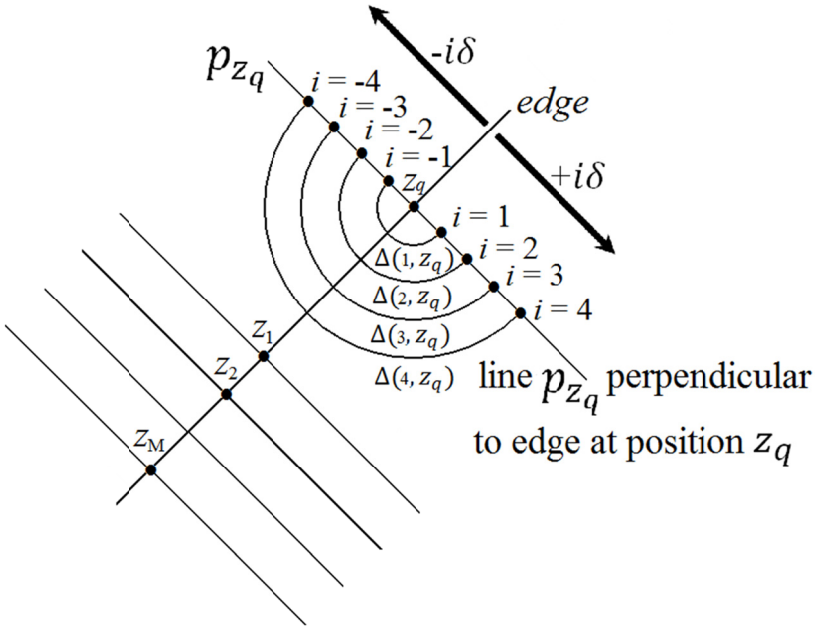


Figure 2.3 – Difference in pixels intensity across an edge

Let the perpendicular line to the edge through this position be p_{z_q}, and i be a position along p_{z_q} then:

$$\Delta(i, z_q) \triangleq |f\left(z_q + i\delta p_{z_q}\right) - f\left(z_q - i\delta p_{z_q}\right)|$$

where $f(z)$ is the image intensity in image f at position z;
δ is a small positive number.

2.3.3.1 LPP on edge difference values independently

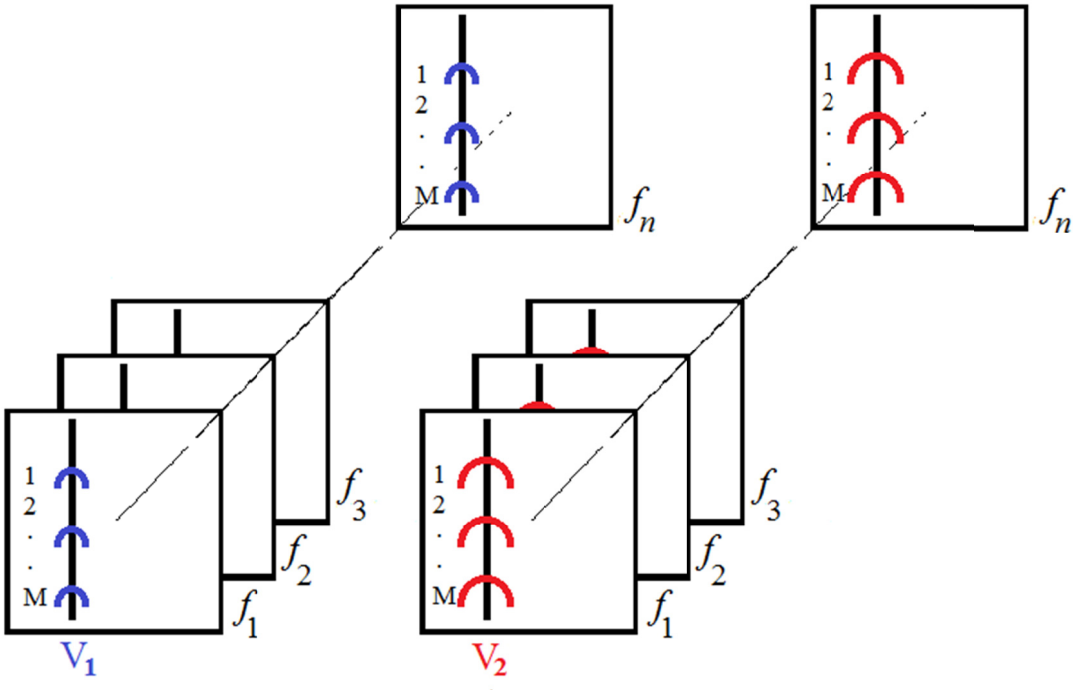


Figure 2.4 – Datasets V_1 and V_2 comprising edge difference values for i=1 and i=2.

Consider an image f_d from an image focusing sequence $f_1, f_2, \ldots, f_n$. For each image f_d, the edge difference values $\Delta_d(i, z_q)$ are calculated for a fixed i at M positions z_q along the edge. The vector data-point is denoted by $\mathbf{v}_{d,i} \in R^M$. Vector $\mathbf{v}_{d,i}$ for the entire image focusing sequence constitute the dataset $V_i \in R^{M \times n}$ to be processed by LPP. (refer to Figure2.4)

This is repeated for various values of *i* to analyse the effect of the intensity data from further away from the edge on the LPP performance.

2.3.3.2 LPP on edge difference values in aggregate

In this experiment the combined effect of pixels across an edge on LPP is analysed. Therefore, for a given image f_d in the focusing sequence $f_1, f_2, \ldots, f_n$, the data vector $\mathbf{v}_d \in R^{MB}$ obtained by stacking the vectors $\mathbf{v}_{d,i}$ for all i=1,2,…,B (refer to Figure 2.5 below)

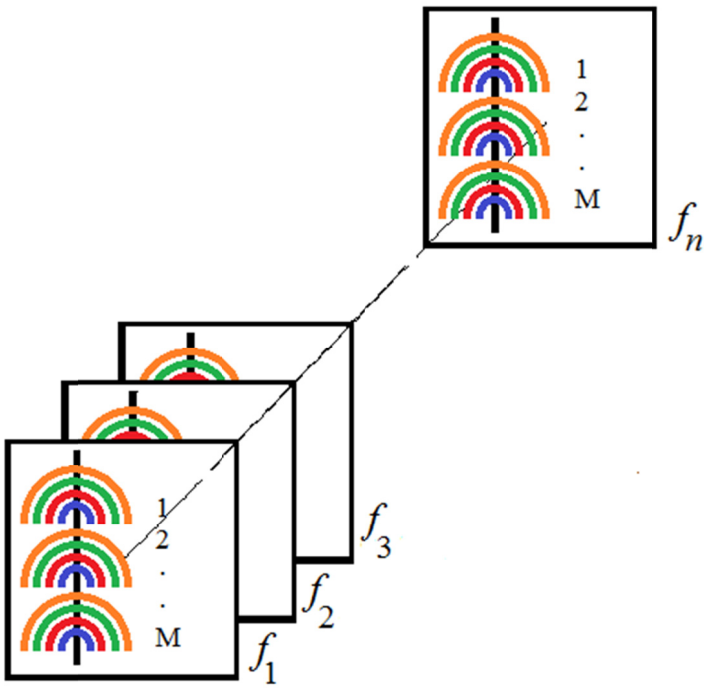


Figure 2.5 – Dataset comprising combined edge difference values for a focusing sequence

2.3.4 Generalization of LPP on difference in pixels across an edge

The scope of this experiment is to study whether the method of applying LPP on pixels across an edge (refer to section 2.3.3.2) could be generalized. This involved training LPP with one dataset and hence obtaining a transformation matrix *A* (refer to Equation 1). Matrix *A* was then used to transform test datasets in low-dimensional representation (refer to Equation 1) and therefore observe whether the correct focused image in test datasets could be detected.

2.3.4.1 Generalization of LPP across different edges in the same focusing sequence

For each focusing sequence, LPP was trained on one of the three edges. It was then tested on the other two edges in the same focusing sequence. This was cross-validated by training a second edge and testing the other two, training the third edge and testing the other two.

2.3.4.2 Generalization of LPP across different focusing sequences

LPP was trained on an edge from one focusing sequence and tested on the three edges of another focusing sequence. For cross-validation, this was repeated for both focusing sequences by training an edge at a time and see what happens with the three edges of the other sequence.

CHAPTER 3

Experimental Results and Analysis

3. Experimental Results and Analysis

This chapter provides results and analysis for experiments. The MATLAB code used in the experiments and pre-processed datasets are provided in the compact disc attached. Moreover disc contents and related instructions may be found in Appendix C. Appendix D contains MATLAB Code Listing.

3.1 Behaviour of LPP on images

3.1.1 Analysing several approaches for LPP on images

Four datasets were used for this experiment. These involved two focusing image sequences having two objects:

- Same distance and different object
- Different distance and same object

The aim was to analyse the resulting position of the focused image with respect to other images in the sequence and to compare results when having:

- Same distance - they should be focused concurrently; and
- Different distance - the first object is focused, and then starts defocusing followed by the second object becoming focused.

The two same distance focusing image sequences named *sd1* and *sd2* were made up of 74 images with image 65 being the focused image (refer to Appendix A.1). Images were resized and cropped to 259×180 and converted to grayscale.

The two different distance focusing image sequences named *near* and *far* were made up of 68 images with images 52 and 55 being the respective focused images (refer to Appendix A.2). Images were resized and cropped to 762×280 and converted to grayscale

Each focusing sequence was organized in a matrix such that each vector representing an image comprises pixel intensity values of such image. For this experiment vectors were normalized to unit norm.

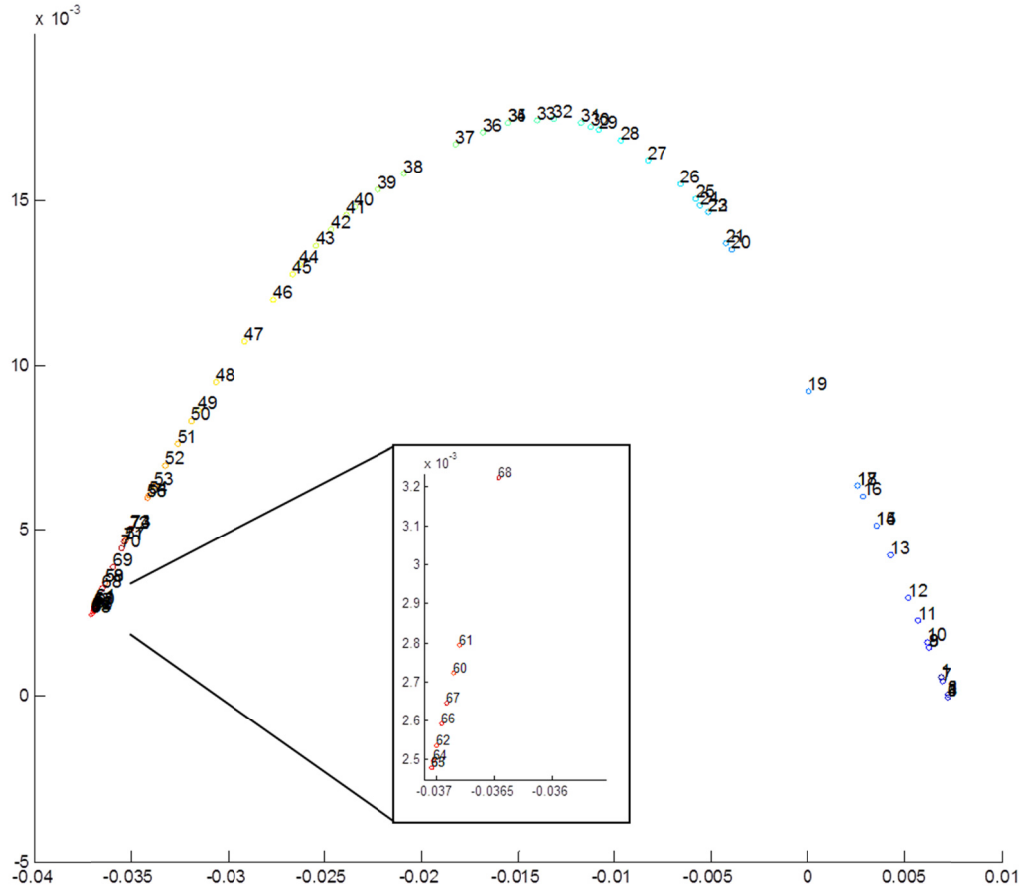


Figure 3.1 – A two-dimensional representation of sd1 dataset using *kNN(k=5)* for adjacency graph, and using *heat kernel (t=5)* for weight matrix. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

The plot above (Figure 3.1) resulted from applying LPP on sd1 dataset, constructing adjacency graph using *k nearest neighbours* [parameter k = 5] and weighting the edges through *heat kernel* [parameter t = 5]. Each data point represents an image. In this plot we obtained a trajectory in the shape of a parabola. It seems that defocused images were projected on the right side while the more focused images were projected on the left side of the plot. One may also note that data points representing

images with similar features were projected near to each other. Thus locality is somehow preserved. The best focused image data-point (image65) is projected at the bottom left followed by data-point for image 63. However this was not the case for all the datasets.

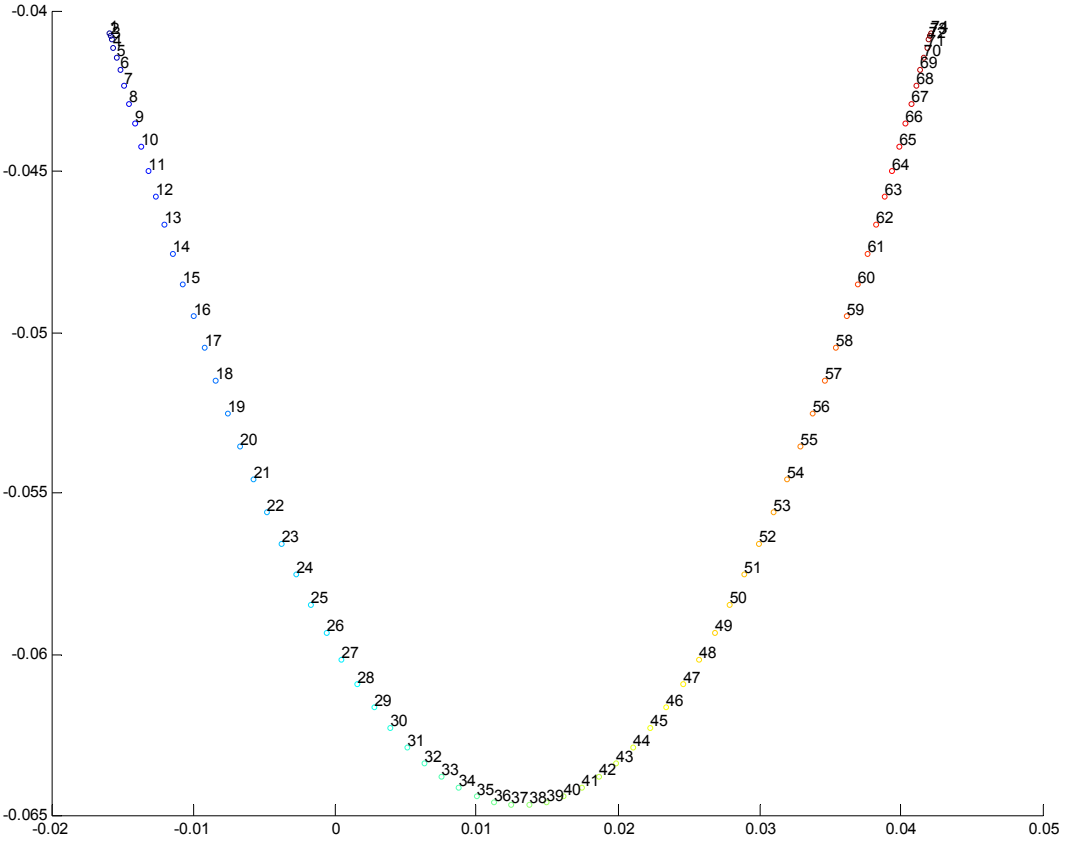


Figure 3.2 – A two-dimensional representation of sd1 dataset using physical adjacency for adjacency graph, and using *heat kernel (t=5)* for weight matrix. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

This plot in Figure 3.2 resulted by imposing physical adjacency on sd1. Here we are imposing physical adjacency such that image 1 should be adjacent with image2, image 2 with image 3 and so on. *Heat kernel* [parameter t = 5] is being used for the weighting the edges. This again resulted in a parabolic shape. However data-points

were projected consecutively starting from 1 at the top right, and ending with 74 at the top left. This is justified due to the fact that adjacency between consecutive vectors is forced.

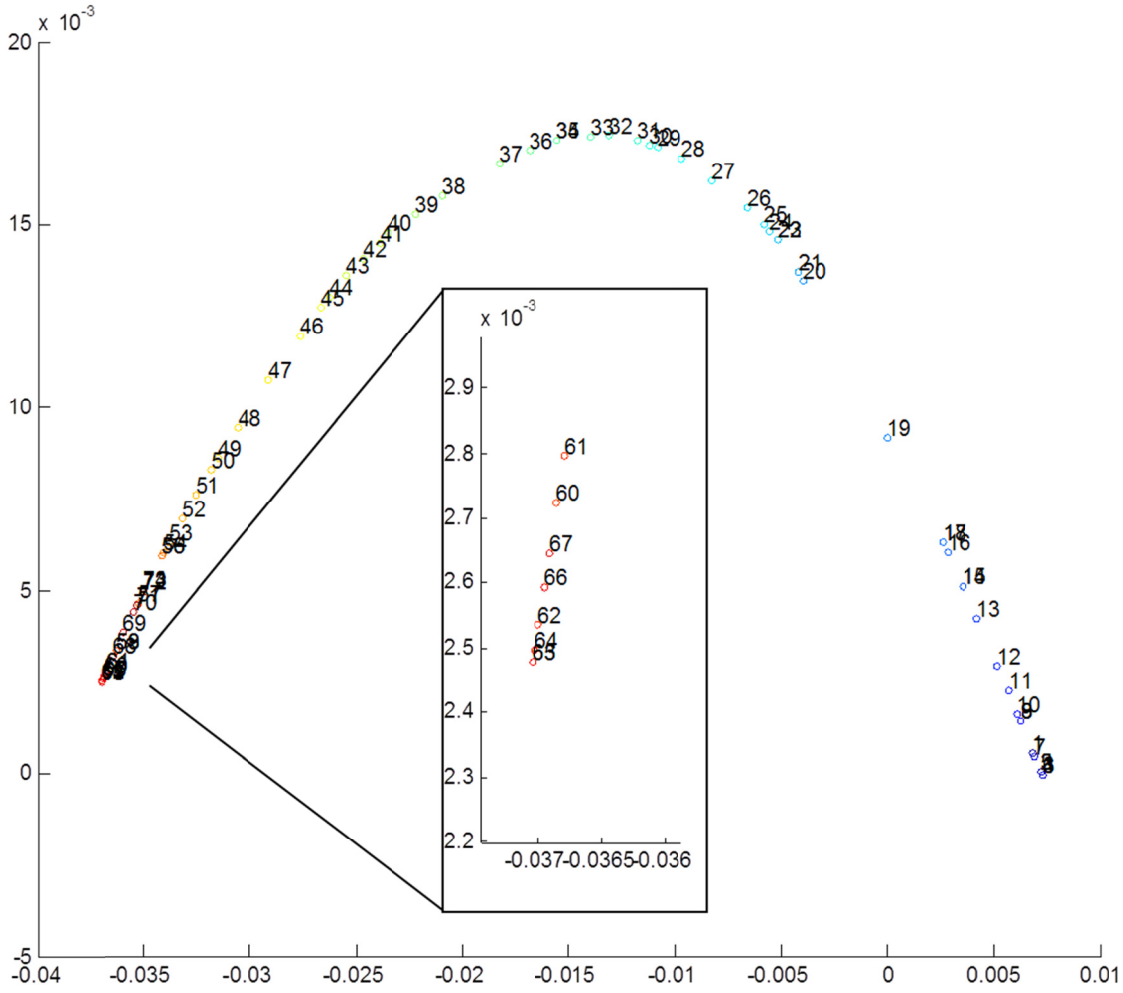


Figure 3.3 – A two-dimensional representation of sd1 dataset using the combination of *physical adjacency* and *kNN(k=5)* for adjacency graph, and using *Heat kernel (t=5)* for weight matrix. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

As shown in Figure 3.3, the combination of imposing physical adjacency with using *k nearest neighbours* [parameter $k = 5$] results in a trajectory which is quite similar to

when the latter approach on its own was used. Again edges are weighted by using *heat kernel* [parameter $t = 5$].

The three approaches above were repeated using a *simple-minded* approach for the weight matrix (for results refer to Appendix B.1). However there was no noticeable difference in plots between *heat kernel* and *simple-minded* approach.

The behaviour of LPP on the other datasets sd2, near and far (refer to Appendix) was quite similar to the example shown above. However the data-point representing the focused image was not in a consistent position such that it could be automatically identified.

3.1.2 Pre-processing data by scaling pixel values

These results were obtained by applying LPP [using parameters k nearest neighbours k=5 and heat kernel t=5] on sd1,sd2 (refer to Appendix A.1) and TV (refer to Appendix A.3) datasets. Datasets sd1 and sd2 were reduced to the more focused images, consisting of the last 18 images in the sequence (i.e. from image 57 to image 74). The TV dataset consisted of 22 images cropped to a size of 200×200 pixels and converted to grayscale. The first image was auto-focused and image 19 was the best manually focused image. Datasets were scaled to [0,1] (instead of normalized to unit norm). This was done by dividing all pixel values in the dataset by the largest pixel value.

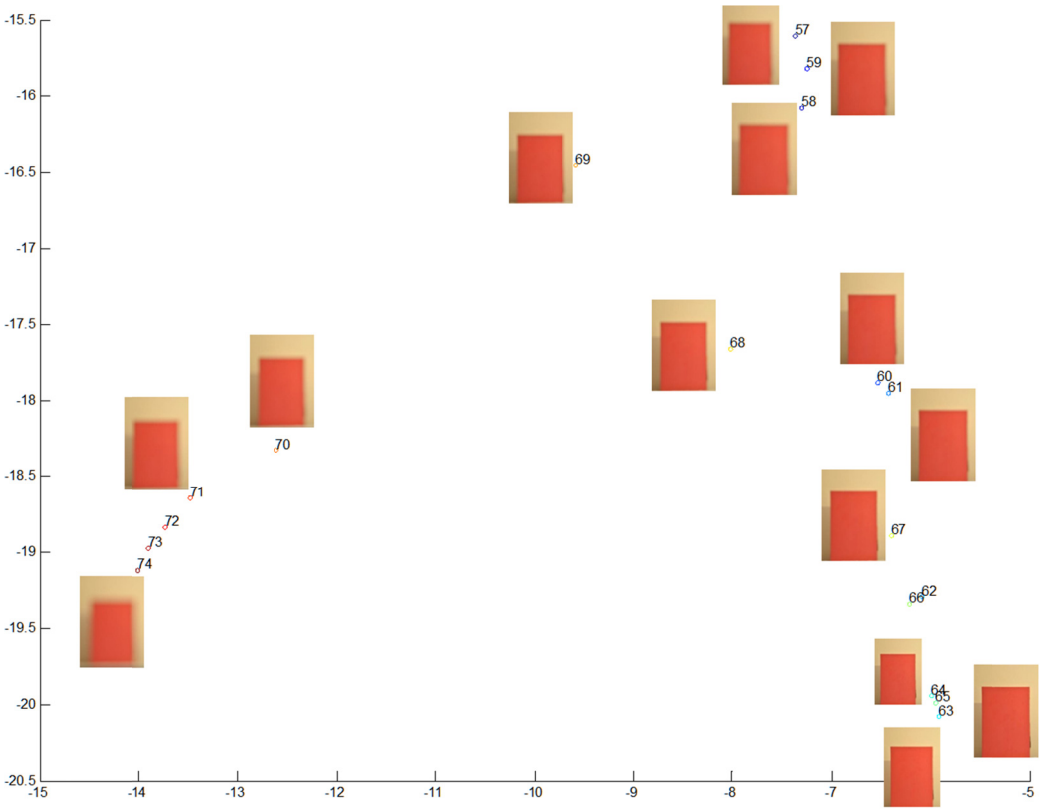


Figure 3.4 – A two-dimensional representation of sd1 with high-dimensional vectors scaled to [0, 1]. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension

Results show that in all three data sets the focused image is projected at some end. Also data points are clustered according to the features in the image they are representing. For instance, in the case of sd1(refer to **Figure 3.4**) the less focused images prior to focusing are projected on top comprising 57, 58 and 59. Then going down we find better focused images, and at the bottom right we find the best focused images 63,64 and 65. Going up again we find images which start to the defocus. The trajectory continues to move in left direction, up to image 69, then it turns down to the left up to image 74. In sd2 (refer to Figure 3.5) and TV(refer to Figure 3.6) we got similar patterns. For the TV dataset we have the better focused images to the left, having image 1 (which was autofocused) at the very left whereas in sd2 the better focused images are projected at the top, with the best focused image at the very top.

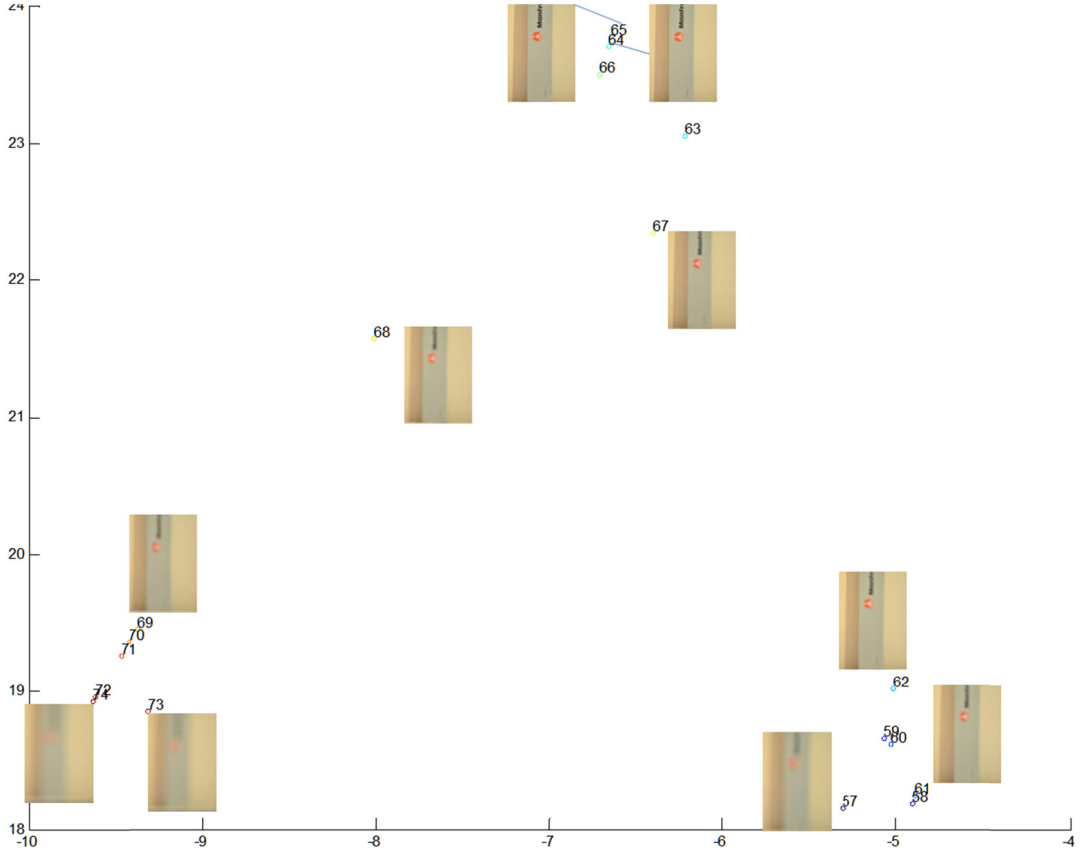


Figure 3.5 – A two-dimensional representation of sd2 with high-dimensional vectors scaled to [0, 1]. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension

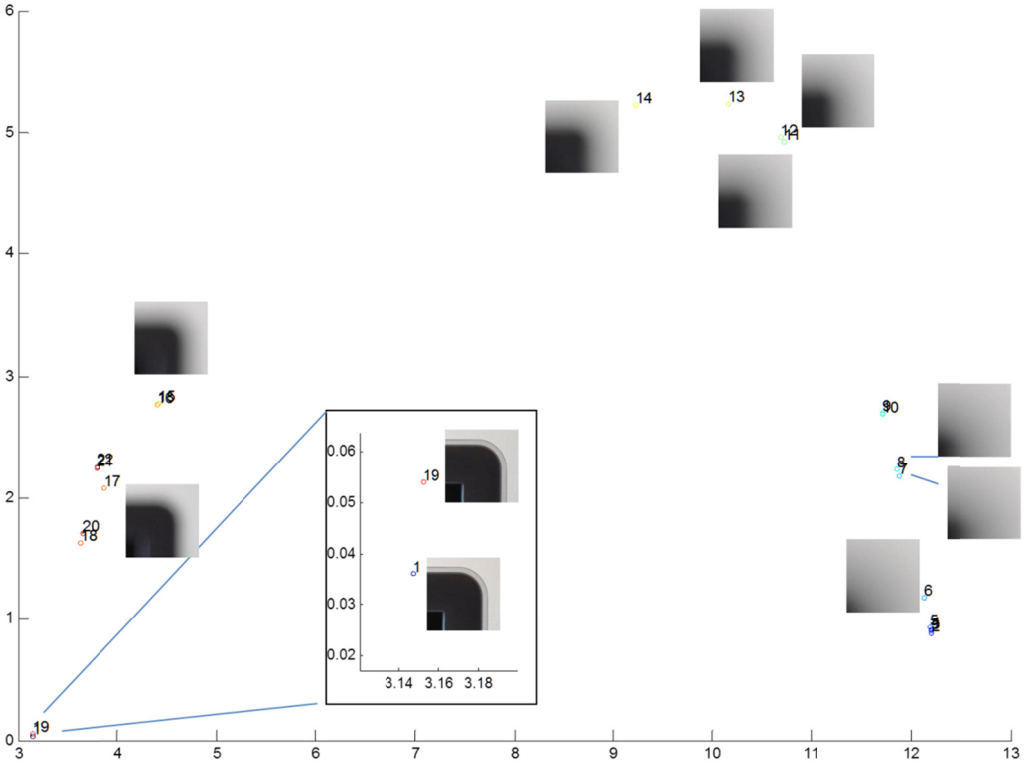


Figure 3.6 – A two-dimensional representation of TV with high-dimensional vectors scaled to [0, 1]. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension

Although results are more interesting than in section 3.1.1, the focused image data point is not consistently projected such that it could be automatically traced. While for sd2 the best focused image is projected at the very top and for TV at the very left, and for sd1 the best focused image (image 65) is positioned between 63 and 64.

3.2 Effect of image content on LPP-transformed data

3.2.1 Two-Dimensional LPP-transformed data

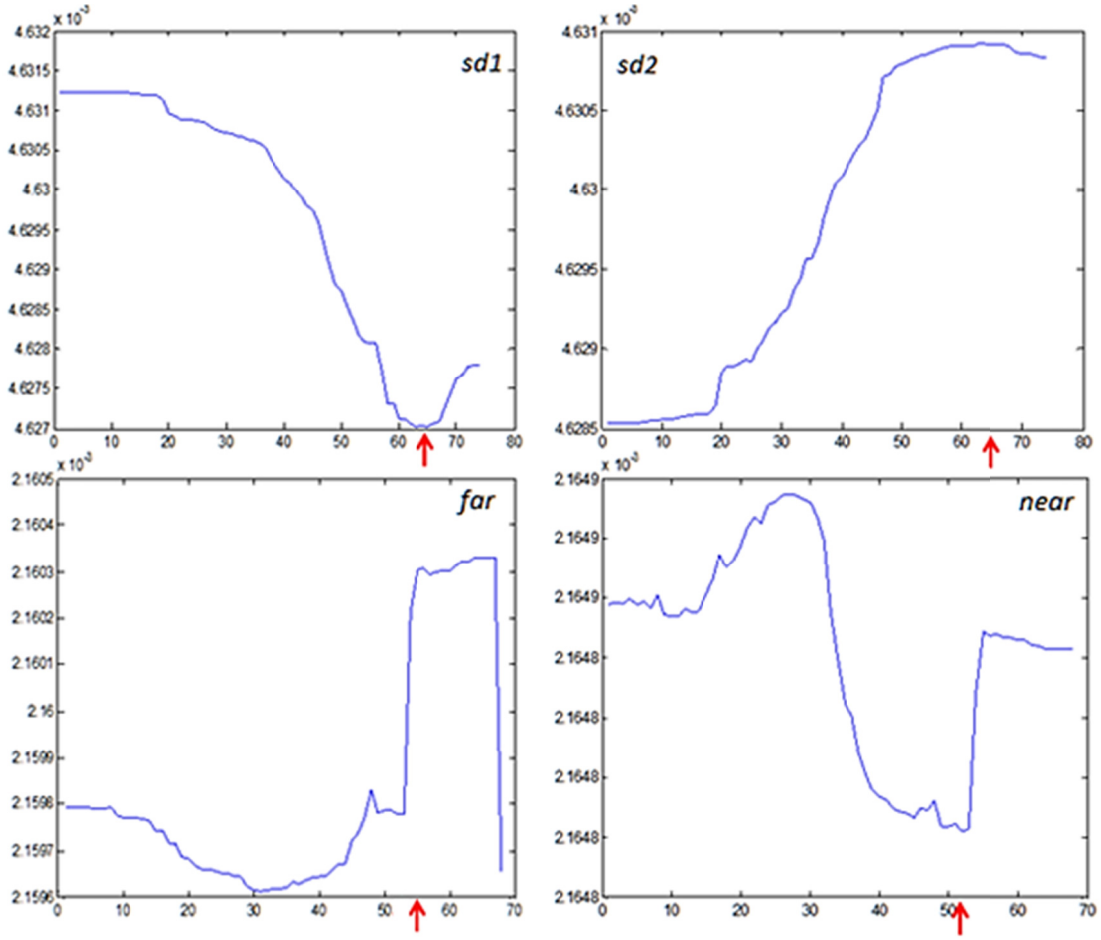


Figure 3.7 – Plots of focusing images sequence (x-axis) vs. RMSE for LPP (y-axis). The arrow denotes the focused image

This experiment was applied on the same four datasets as in section 3.1.1, that is *sd1, sd2, near* and *far*. The parameters used for LPP were *k nearest neighbours k=5 and heat kernel t=5*. These four plots show the root mean squared error for the four datasets. While expecting the highest error for the focused image, it seems that results obtained were quite inconsistent. For the focused image (image 65) in sd1 and sd2 dataset a minimum and a maximum error were obtained respectively. A

minimum error was also obtained for the near dataset (image 52) whereas image 55 in far dataset has a local maximum.

3.2.2 A range of dimensions for LPP-transformed data

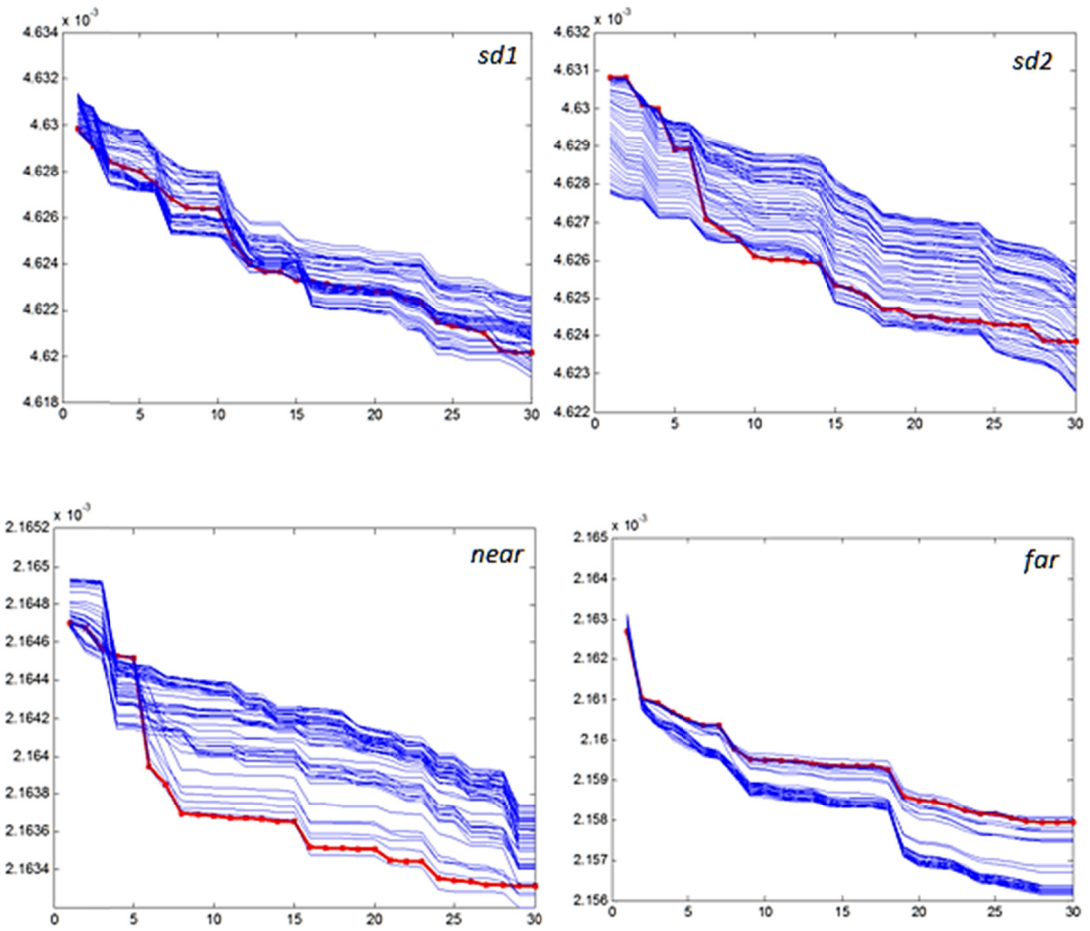


Figure 3.8– Plots of Low dimensions considered (x-axis) vs. RMSE (y-axis). The red plot represents the focused image.

For each of the four datasets every plot represents an image in the dataset while the red plot represents the focused image. These plots show how RMSE decrease when

more dimensions are considered. The red plots show that for all datasets with the exception of the last, the focused image had the lowest error for some dimensions. However due to inconsistency in results this approach cannot be generalized.

3.3 Behaviour of LPP on difference in pixels across the edge

Experiments in this section involved the following data:

- Same distance 1 (sd1) focusing sequence (refer to Appendix A.1.2) – the last 18 images (i.e. from image 57 to image 74)
- Boxes focusing sequence (refer to Appendix A.4)

The datasets prepared for the experiments in the sections below were scaled to [0,1] by dividing all values by the largest value. Moreover, the following parameters were used for LPP: *k nearest neighbours k=5 and heat kernel t=5*.

3.3.1 LPP on edge difference values independently

For each of the focusing sequences mentioned above (sd1 and Boxes) an edge was manually selected. Ten respective datasets were prepared, each with a different value of i ($1 \leq i \leq 10$), that is V_1, V_2, … up to V_{10} ($1 \leq q \leq 200$)

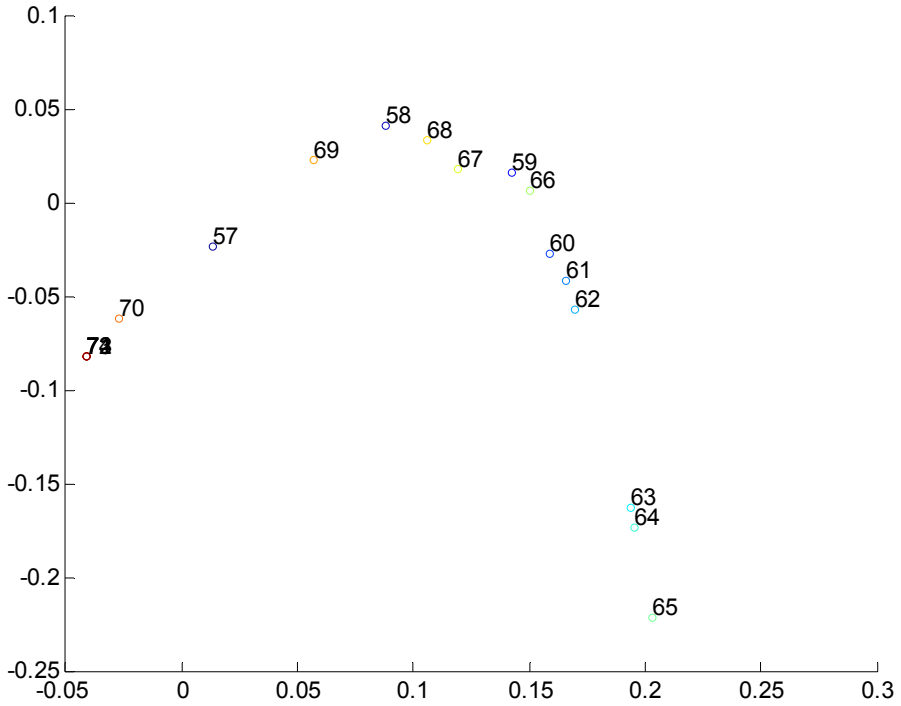


Figure 3.9 - A two-dimensional representation of edge difference values V_9 for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

Figure 3.9 above shows the result of applying LPP on $\mathbf{V_9}$ of sd1. The trajectory starts from data-point 57, and moves to the right towards 58, 59, down to data-point 65. After data-point 65 (focused image) the trajectory changes direction and gradually moves back up and to the left side until it reaches the data-point of image 74 (the last image). Applying LPP independently on the rest of the ten datasets mentioned above (each with a different value of i $[1 \leq i \leq 10]$) provided similar results (refer to Appendix B.2.1.1 for more results). Some results had shown that a change in direction of the trajectory occurred more than once. However it was generally noted that the distance between the focused image data-point and its immediate neighbours (the previous image and the next image) was larger than for any other data-point and its neighbours.

The low dimensions in LPP are ordered in descending order according to the magnitude of the eigenvalues. Thus the first dimension of LPP retains the most

information. Such significance decreases for each subsequent dimension. Figure 3.10 below shows typical resulting eigenvalues for this experiment. This shows that the first three dimensions contain a certain degree of information whereas the fourth and fifth dimensions are less important.

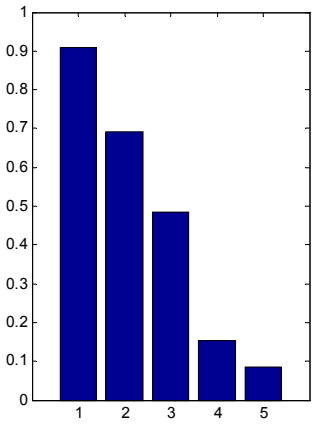


Figure 3.10 – Typical Eigenvalues – LPP on edge difference values independently

3.3.2 LPP on edge difference values in aggregate

The selection of edges was done manually by visually analysing the focused image and choosing approximate edges. As shown in Figure 3.11, for each of the two focusing sequences three edges were selected.

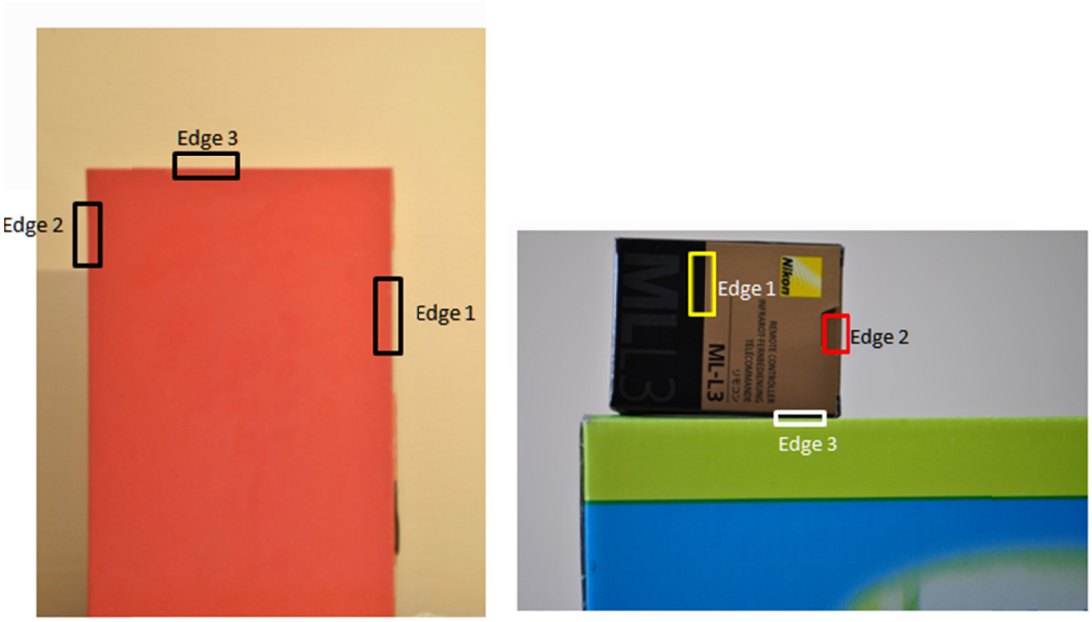


Figure 3.11 – Three manually selected edges on sd1(left) and boxes (right)

Four datasets for each focusing sequence were prepared as follows:

- A dataset V for each edge was prepared such that $1 \leq q \leq 200$ and $1 \leq i \leq 10$; and
- Another dataset V(comp) made up by stacking the three datasets above, V(Edge 1), V (Edge 2) and V (Edge 3)

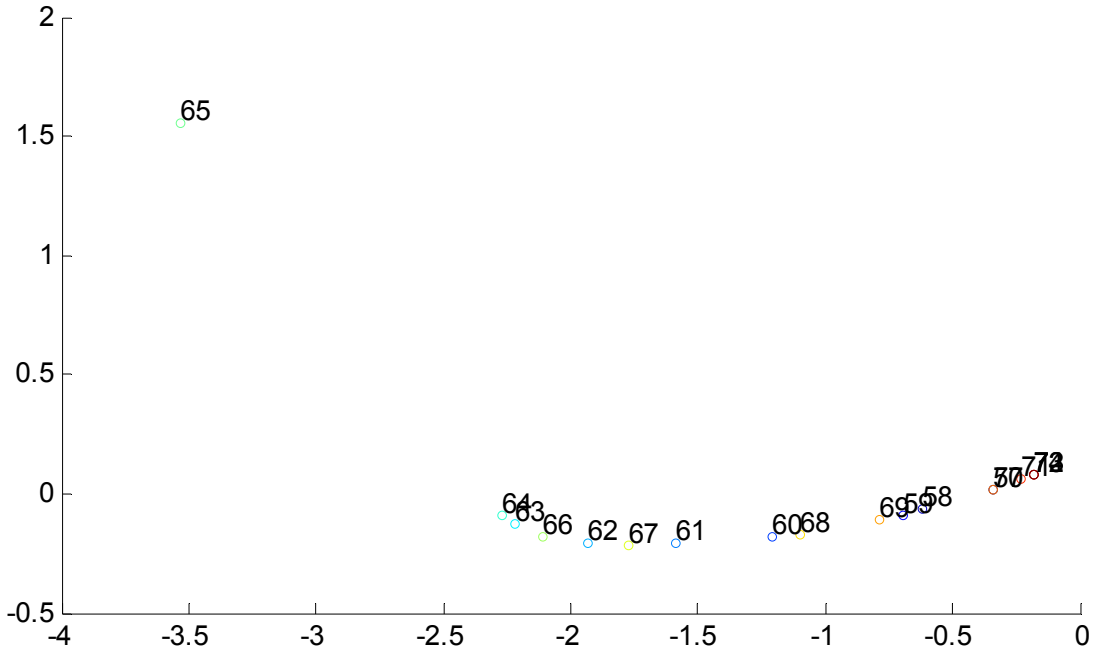


Figure 3.12 - A two-dimensional representation of edge difference values V(Edge1) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

Resulting plots are similar to those in section 3.3.1 in that the trajectory starts in a direction, continues to move in such direction until the focused image, and then it goes back. However, plots are smoother and stronger such that the focused image data-point stands out more clearly. For instance as can be seen in Figure 3.12 above, the trajectory moves smoothly from data-point 57 towards the next images in the sequence until the focused image (data-point 65 in the top left) and then smoothly goes back in the opposite direction. Such smoothness might be justified by the fact that more detailed information is fed into the LPP algorithm. On visually analysing plots such as Figure 3.12 (and other similar results in Appendix B.2.1.2) one can note that within just the first dimension (x-axis) there is already a clear representation of the information of such focusing image sequence and that the data-point representing

the focused image is more distant than its neighbouring data-points. For instance in Figure 3.12 there is relatively a long distance from 65 to 64 and 66.

The eigenvalues in Figure 3.13 below show that the first three dimensions contain a certain degree of information, especially the first two, whereas the fourth and fifth dimensions are less important.

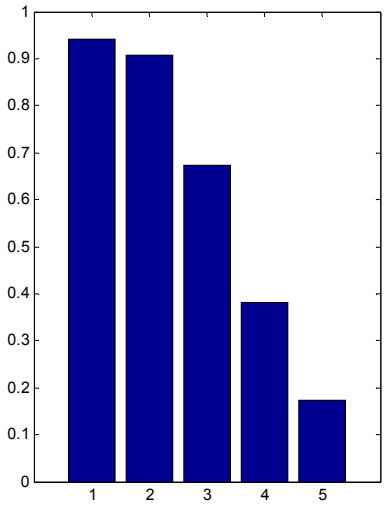


Figure 3.13 - Typical Eigenvalues – LPP on edge difference values in aggregate

3.3.3 Further Analysis

On visually analysing the results obtained in the two sections above, one can notice a common pattern in the way data is represented. As expected generally all plots resulted in a trajectory which moves from one data-point to the next until the data-point representing the focused image. At such point the trajectory changes its direction and moves back. This is justified by the fact that the image sequence starts

from a defocused image, it gradually gets in focus, and then after reaching maximum focus, it starts defocusing again. Hence LPP is preserving the intrinsic information of the data and presenting a representation of such problem in a low dimensional space. It is folding data-points subsequent to focusing on the data-points prior to focusing as these contain similar information. Also, another observation is that as shown in Figure 3.10 and Figure 3.13, the underlying information of data lies mostly in the first three dimensions of LPP. Results also shown that the distance between the data-point representing the focused image and its immediate neighbours was generally the longest when compared with the other data-points and their respective neighbours.

This has evolved into:

- Analysing occurrences of change in direction for the first dimension of LPP (refer to appendix B.2.2).
- Calculating and analysing total Euclidean distances between data-points and their respective immediate neighbours (excluding the first and the last). This was done in 1,2,3,4 and 5 dimensions (refer to appendix B.2.3).

A change in direction was detected for the focused image data-point in all the datasets that were analysed. There were also other data-points for which a change in direction was detected. However when analysing distances, there was the following outcome:

- Euclidean distances(Appendix B.2.3.2) for results in section 2.3.3.2 – the data-point representing the focused image is at the maximum Euclidean distance even if only the first dimension is considered
- Euclidean distances (Appendix B.2.3.1) for results in section 2.3.3.1 – when considering more dimensions, (this varied from 2 to 5 dimensions) the data-point representing the focused image is at the maximum Euclidean distance.

Finally, when analysing distances among those data-points in which a change in direction was detected (ignoring the rest of the dataset), the focused image data-point was found to be at a maximum distance when considering two to three dimensions. Based on the analysis in this section an algorithm for detecting the focused image from the output of LPP was designed (refer to section 2.2.2.3). Such algorithm was tested on all datasets prepared for sections 3.3.1 and 3.3.2 and was found to be successful.

3.3.4 Automating edge detection and selection

Edges in the previous experiments were selected manually. An algorithm to automate such process was designed (refer to section 2.2.2.2). This section shows the outcome of such algorithm when applied on both 'sd1' and 'Boxes' focusing image sequences.

As shown in Figure 3.14 for each focusing sequence the mean image was computed and the longest vertical edge out of the detected edges was selected. From the selected edge a dataset V was computed such that $1 \leq q \leq$ M and $1 \leq i \leq 10$;

where M = the length in pixels of the selected edge.

For sd1 M was 117 whereas for Boxes it was 83

LPP was then applied and the algorithm in section 2.2.2.3 was able to detect the correct focused image from the output of LPP.

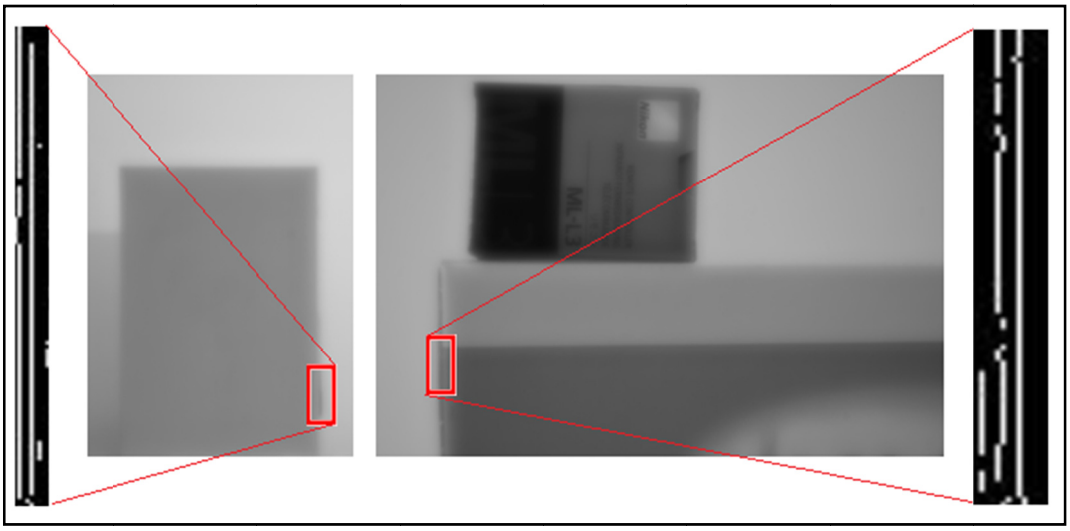


Figure 3.14 – Mean Images for sd1 and Boxes sequences and their respective selected edge.

The plots below show the results of LPP on the selected edges. In both cases the focused image data-point (65 for sd1 and 10 for boxes) clearly stands out. The trajectory changes direction at the data point representing the focused image and such data-point is projected at a distance from other data-points.

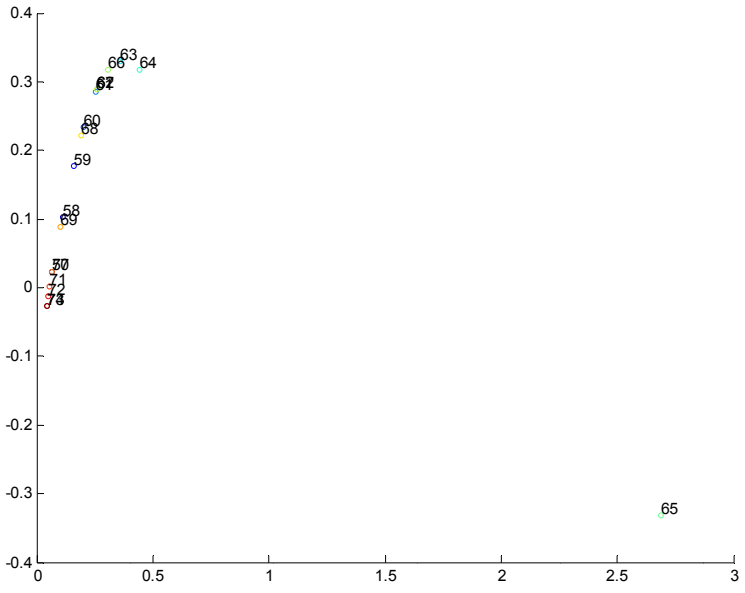


Figure 3.15 - A two-dimensional representation of edge difference values V(sd1) of the automatically selected edge for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

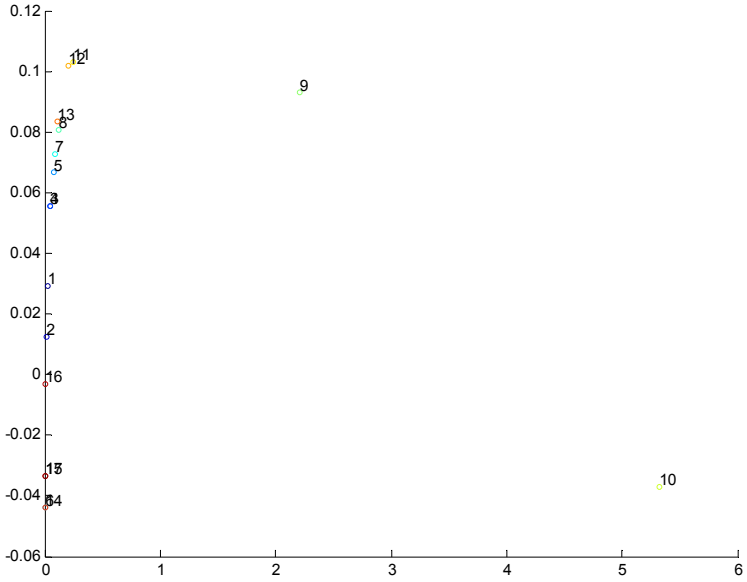


Figure 3.16 - A two-dimensional representation of edge difference values V(Boxes) of the automatically selected edge for Boxes data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

3.4 Generalization of LPP on difference in pixels across an edge

The datasets for each edge prepared in section 3.3.2 were used for this experiment. Again, the following parameters were used for LPP: *k nearest neighbours k=5 and heat kernel t=5*.

3.4.1 Generalization of LPP across different edges in the same focusing sequence

Results showed that when training LPP with on one edge dataset and use the resulting transformation matrix A to transform the other two edge datasets in the same focusing sequence, a correct result was obtained. The table below shows the results of applying the algorithm described in section 2.2.2.3 on the transformed data.

Training Dataset	Testing Dataset	Resulting Focused Image
SD1 - Edge1	SD1 - Edge2	65
	SD1 - Edge3	65
SD1 - Edge2	SD1 - Edge1	65
	SD1 - Edge3	65
SD1 - Edge3	SD1 - Edge1	65
	SD1 - Edge2	65
Boxes - Edge1	Boxes - Edge2	10
	Boxes - Edge3	10
Boxes - Edge2	Boxes - Edge1	10
	Boxes - Edge3	10
Boxes - Edge3	Boxes - Edge1	10
	Boxes - Edge2	10

Moreover the plots below show examples of such results. Such plots again show a consistent pattern such that the trajectory moves towards the focused image data-point (10 for 'Boxes' and 65 for 'sd1') and then at such point it changes direction and goes back. Again the distance between the focused image data-point and its immediate neighbours is larger than the distance between any data-point and its respective immediate neighbours. Further examples of such plots may be found in Appendix B.3.1

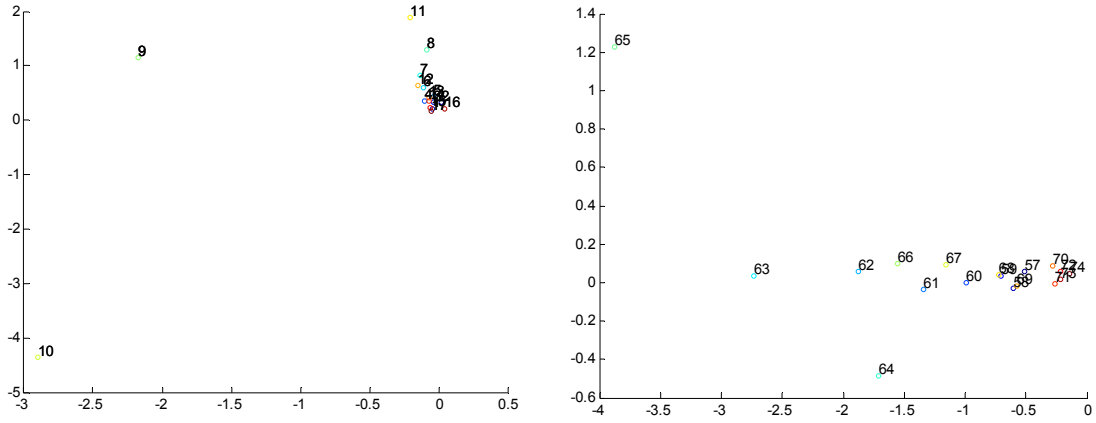


Figure 3.17 - Two-dimensional representations of testing sets V(Edge2) of Boxes (left plot) and V(Edge3) of sd1 (right plot) using the respective training sets V(Edge1) of Boxes and V(Edge1) of sd1. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

3.4.2 Generalization of LPP across different focusing sequences

Results showed that when training LPP with on one edge dataset and use the resulting transformation matrix A to transform three edge datasets from a completely different focusing sequence, a correct result was obtained. The table below shows the results of applying the algorithm described in section 2.3.4 on the transformed data.

Training Dataset	Testing Dataset	Resulting Focused Image
SD1 - Edge1	Boxes - Edge1	10
	Boxes - Edge2	10
	Boxes - Edge3	10
SD1 - Edge2	Boxes - Edge1	10
	Boxes - Edge2	10
	Boxes - Edge3	10
SD1 - Edge3	Boxes - Edge1	10
	Boxes - Edge2	10
	Boxes - Edge3	10
Boxes - Edge1	SD1 - Edge1	65
	SD1 - Edge2	65
	SD1 - Edge3	65
Boxes - Edge2	SD1 - Edge1	65
	SD1 - Edge2	65
	SD1 - Edge3	65
Boxes - Edge3	SD1 - Edge1	65
	SD1 - Edge2	65
	SD1 - Edge3	65

Figure 3.18 below again show the consistent pattern experienced in previous experiments. This time the experiment is done across focusing sequences. Further examples of such plots may be found in Appendix B.3.2

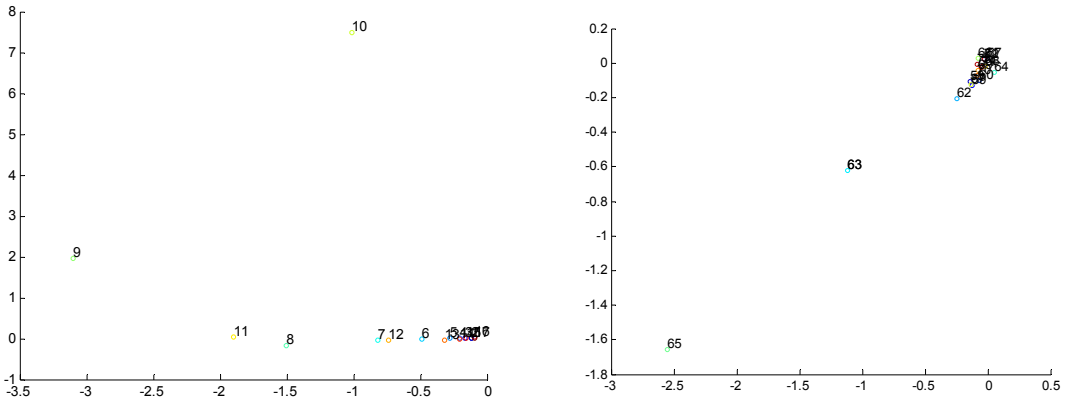


Figure 3.18 - Two-dimensional representations of testing sets V(Edge3) of Boxes (left plot) and V(Edge2) of sd1 (right plot) using the respective training sets V(Edge1) of sd1 andV(Edge3) of Boxes. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

CHAPTER 4

Discussion

4. Discussion

Throughout this investigation, several approaches were attempted in order to determine whether LPP can be used to identify the best focused image from a sequence of incrementally focused images. Many techniques reviewed in literature were composed of two stages, a focus measure and then a search algorithm whereas our approach is mostly concerned with representing the auto-focusing problem in a low-dimensional space. Most approaches in literature are based on depth computation. *Depth From Focus* is slow as it needs to acquire many images. *Depth From Defocus* is faster as needs only few images, however it has high computational costs due to inverse filtering. In our method we are also acquiring a focusing sequence that comprises many images. However the focused image is identified through a simple search in a low dimensional representation.

4.1 Behaviour of LPP on images

One attempt involved studying the behaviour of LPP on images. Here results shown that the two-dimensional plots were quite smooth and parabolic. Data-points representing images with similar features were projected near each other. This shows that LPP is able to preserve the intrinsic data within the images. However the problem was that the data-point representing the focused image was not projected in some consistent identifiable position. In section 3.1.2, datasets were scaled to the range [0, 1]. Such approach provided better results. For instance, the plot in Figure 3.6 shows that the best focused image (data-point 1) was projected at the very lower left. But this was not consistent throughout all datasets.

4.2 Effect of image content on LPP-transformed data

An alternative approach was investigated. This involved applying LPP on images and reconstructing back the high-dimensional image data from the LPP low-dimensional data, thus neglecting some eigenvectors. Then the root mean squared error was computed from the difference between the original and reconstructed data.

The highest error was expected for the focused image since it contains more detail. Yet results were not consistent and therefore such approach could not be generalized.

4.3 Behaviour of LPP on difference in pixels across the edge

It is known by experience that contrast is reduced when an image is defocused. In fact many techniques discussed in literature exploit the blurring aspect, and hence quantify the amount of defocus to achieve a focus measure. When an image is defocused, edges and lines become less defined and small details start being lost. Edges are similar in all focused images except for their contrast. Therefore there is a certain degree of consistency and image content independence when considering edge data. In view of all this, this study investigated the behaviour of LPP on difference in pixels across an edge. For the scope of the study, simple vertical and horizontal edges were considered. There might have been accuracy limitations since the selection of edges was done manually. However results showed a consistent pattern in that the trajectory moves towards the focused image data-point and at such point it turns its route back (refer to section 3.3). Moreover, in contrast to other data-points, the focused image data-point was generally distant from its immediate neighbours. Eigenvalues suggested that generally underlying information lies within the first three dimensions of LPP. In view of all this, an algorithm to search for the focused image data-point in the output from LPP was designed. This involved two-stages. The first stage selects those data-points where a change in direction is detected in either the first or second dimension. The second stage involved choosing from the selected data-points, that data-point which is at a maximum (Manhattan) distance from its immediate neighbours. The first three dimensions were considered for the magnitude of such distance.

In order to automate this process further an edge detection method was used to extract edges. As shown in section 3.3.4, using edge detection, selecting the longest vertical edge, and applying LPP on edge difference values provided successful results. The algorithm is simple in that only one vertical edge is finally selected. However the same principle would apply if more edges would be considered. This therefore indicates that the whole process from a focusing image sequence to the identification of the focused image could be automated.

4.4 Generalization of LPP on difference in pixels across an edge

Another important question in this investigation was whether such method could be generalized. Results specifically shown that this method is generalizable across different focusing sequences (refer to section 3.4.2). This provides more computational efficiency as LPP could be applied just once on some dataset. Then the resulting transformation matrix A is used every time a dataset needs to be transformed in low-dimensions. This provides more computational efficiency since there is no need to apply LPP every time.

CHAPTER 5

Conclusions and Recommendations for Future Work

5. Conclusions and Recommendations for Future Work

5.1 Summary and Achievements

The main goal of this project was to investigate whether LPP can be used to determine the best focused image from a sequence of incrementally focused images. While most techniques reviewed in literature define a focus measure, this study was concerned with finding a way to represent the auto-focusing problem in low-dimensional space. As to achieve this objective, three different approaches were investigated. The approaches of applying LPP on images and that of examining the effect of image content on LPP-transformed data failed to accomplish such objective.

The investigation showed that the most effective way of representing the auto-focusing problem is by applying LPP on the difference in pixel intensity across edges. Consistency in results led to the creation of a simple algorithm in which the focused image data-point could be identified. Also since edges were concerned, a way of automating the extraction and selection of an edge was proposed. Results also demonstrated that the problem may be represented in the first three dimensions of LPP. Moreover, another finding in this investigation was that such approach could be generalized across different focusing sequences. This would therefore provide more computational efficiency.

5.2 Future work

The findings of this study could evolve in investigating further aspects. The strategy proposed for finding and selecting edges might be further sophisticated and extended to select strong edges at various orientations. Consequently this might progress into studying if the approach revealed in this project might be used to determine depth. Consider for instance the case of having a limited depth of field, and having a focusing sequence with different objects situated at a different distance. Hence the

focusing of object A occurs before the focusing of object B. Using edge detection and applying LPP-transformation on edge difference values for A and B independently would result in identifying when A is focused and when B is focused. This could then be used to compute depth from A to B. An alternative objective to depth could be multi-focusing.

APPENDICES

Appendix A: Focusing Image Sequences

Focusing image sequences were shot with a digital SLR camera. The aperture was fully opened as to allow a narrow depth of field. After directing the camera to the object and composing the image, images were shot sequentially while manually slightly rotating the focusing ring. Images were resized/cropped as required.

A.1 Same distance different object

Figure A.1: Two different objects at the same distance

The two focusing image sequences in this section were formed by shooting a focusing sequence of two different objects (as shown in Figure A.1) at the same distance from the camera. The images were then cropped around the two objects. For these two datasets image 65 is the best focused image.

A.1.1 Same distance 1 – sd1

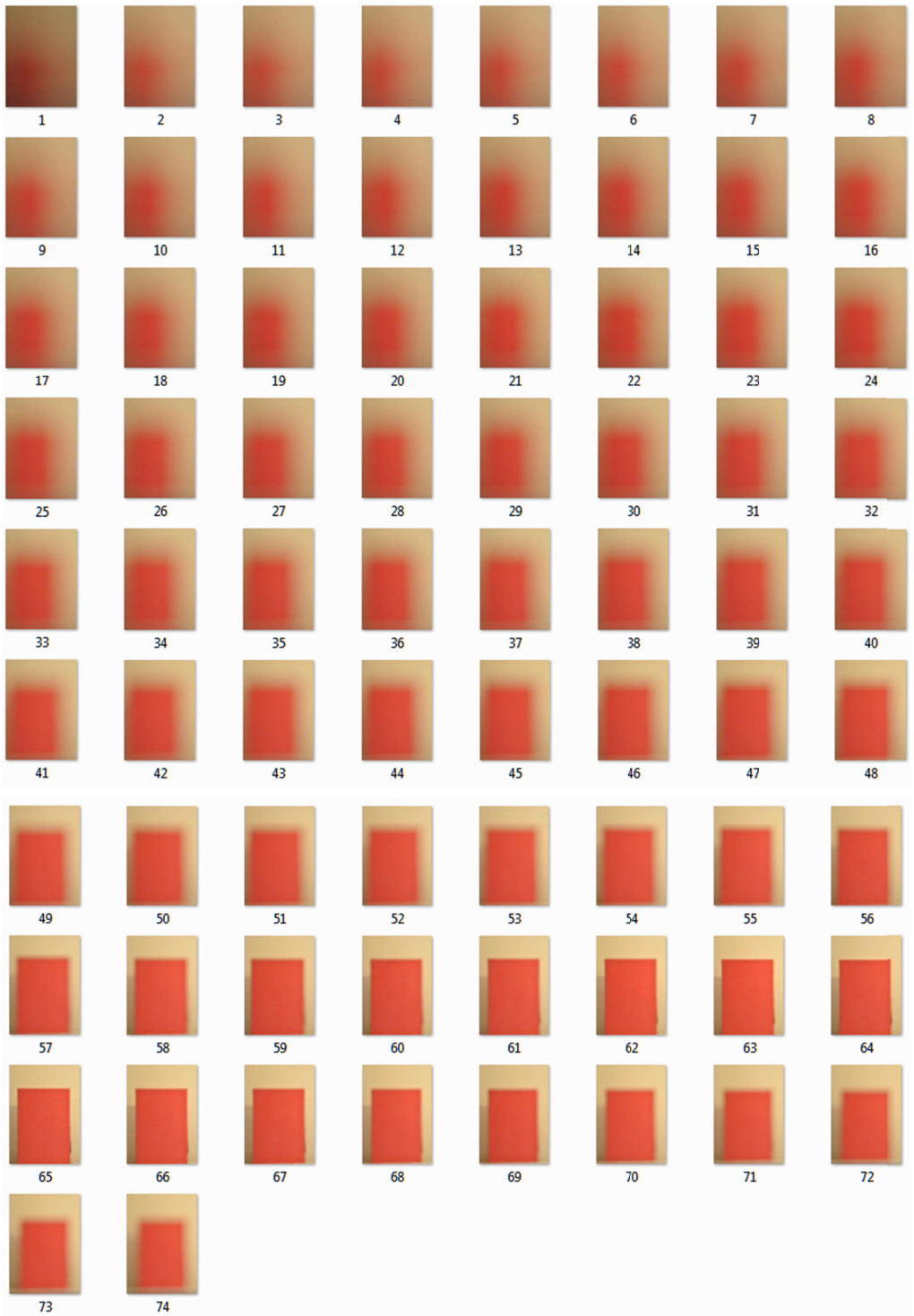


Figure A.2: sd1 focusing sequence

A.1.2 Same distance 2 – sd2

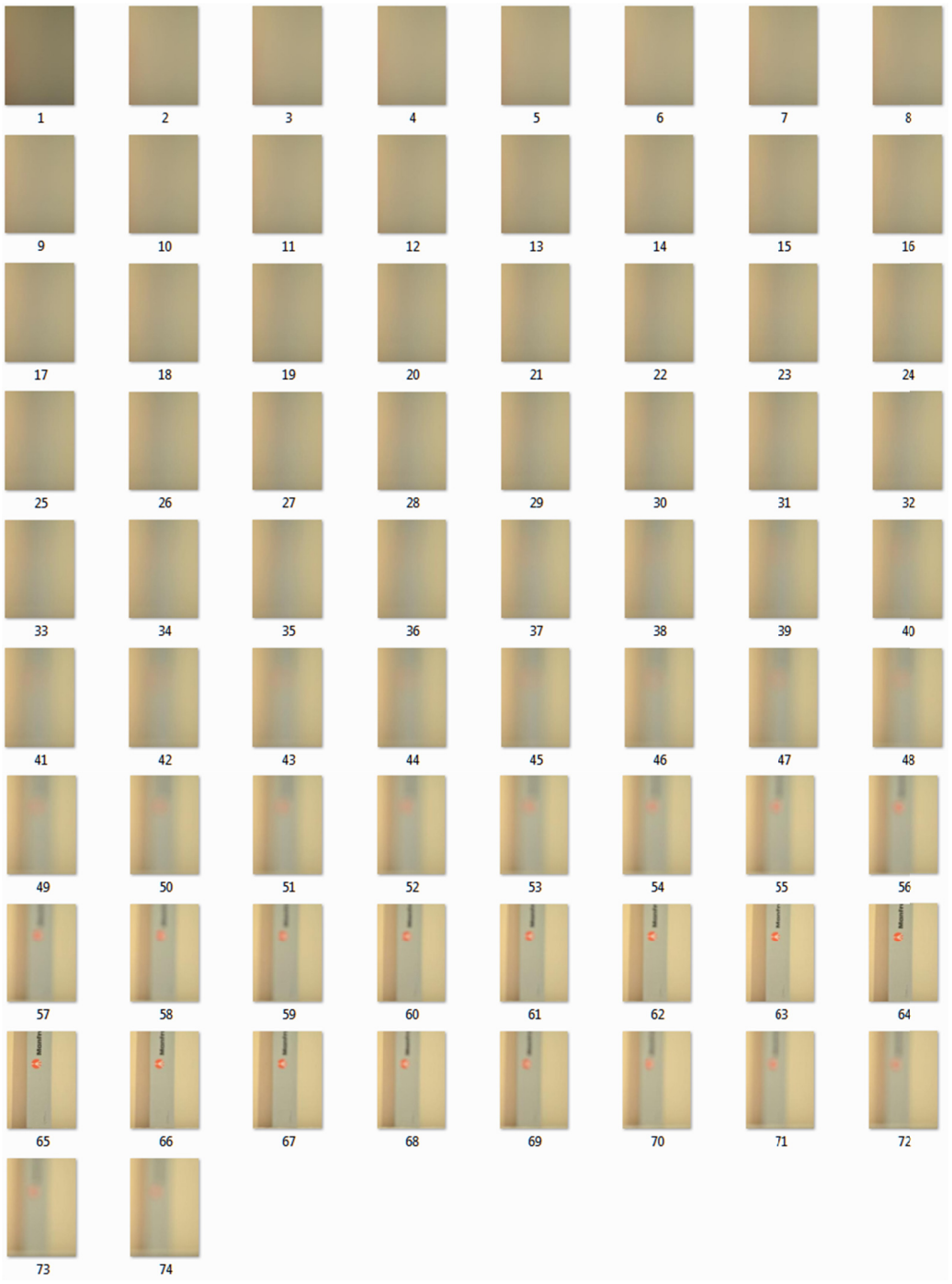


Figure A.3: sd2 focusing sequence

A.2 Different distance same object

Figure A.4: Two identical objects at a different distance

The two datasets in this section were formed by shooting a focusing image sequence of two identical objects (as shown in figure) at a different distance from the camera. This allowed for not having the objects focused at the same time. Consequently the nearer object was best focused at image 52 whereas the other object was focused at image 55. Images were then cropped around the two objects to form the respective datasets.

A.2.1 Near dataset

Figure A.5: near focusing sequence

A.2.2 Far dataset

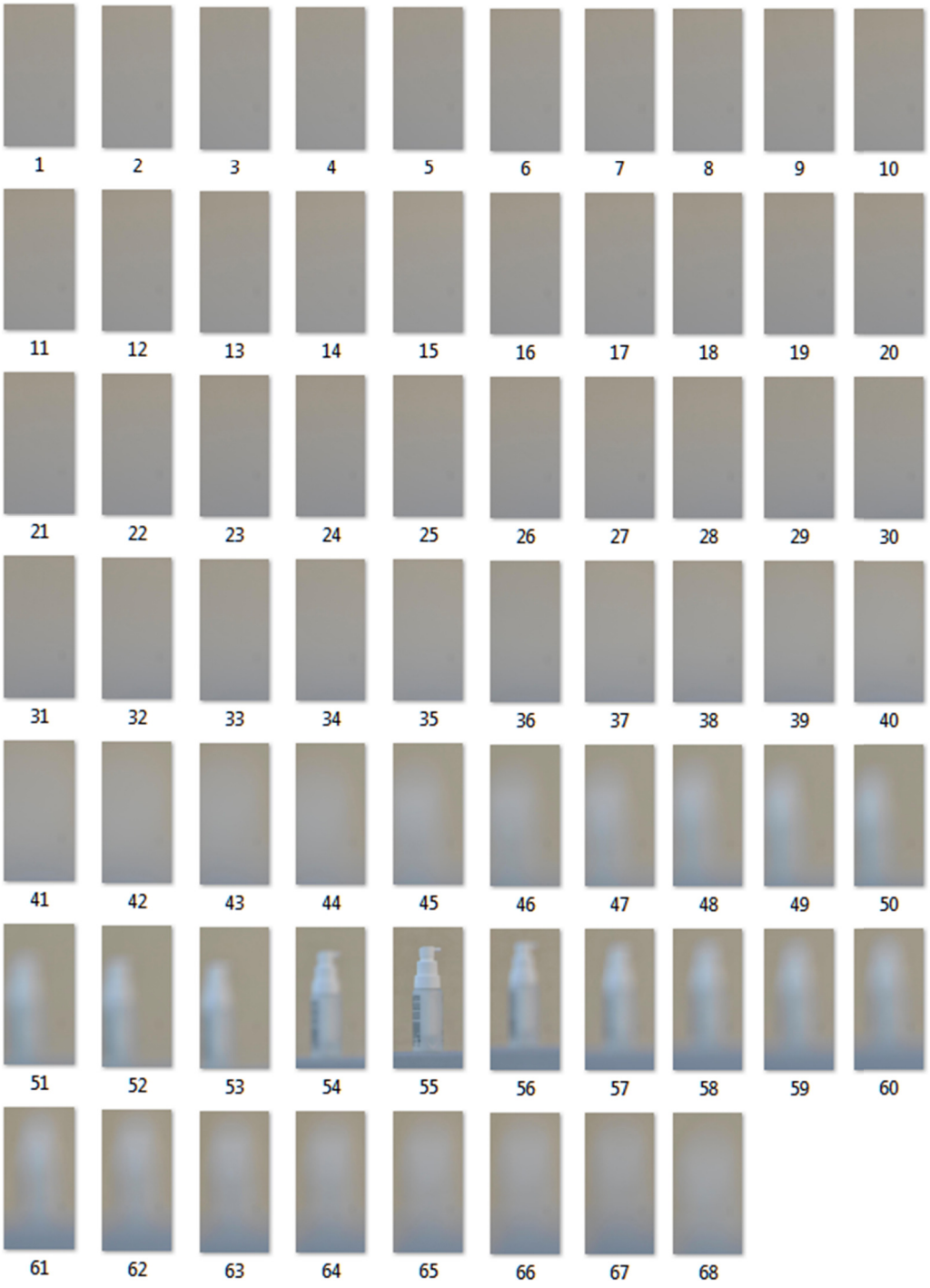


Figure A.6: far focusing sequence

A.3 TV

The first image for this focusing sequence was auto-focused whereas the other images were shot sequentially while manually varying focus as done for the previous datasets.

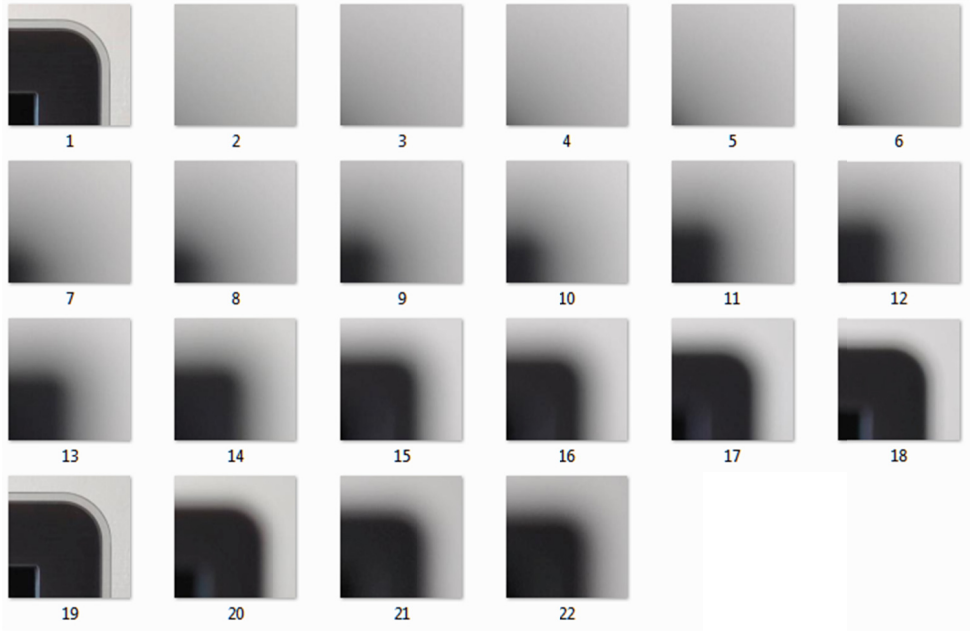


Figure A.7: TV focusing sequence

A.4 Boxes

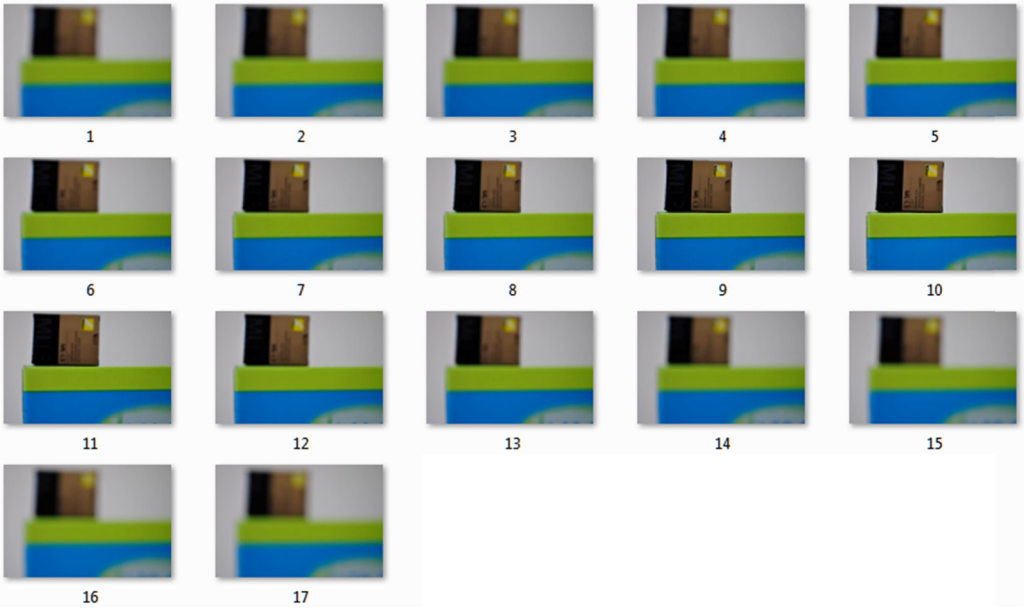


Figure A.8: Boxes focusing sequence

Appendix B: Further Results

B.1 Behaviour of LPP on images

B.1.1 'sd1' dataset – (simple-minded approach for weighting edges)

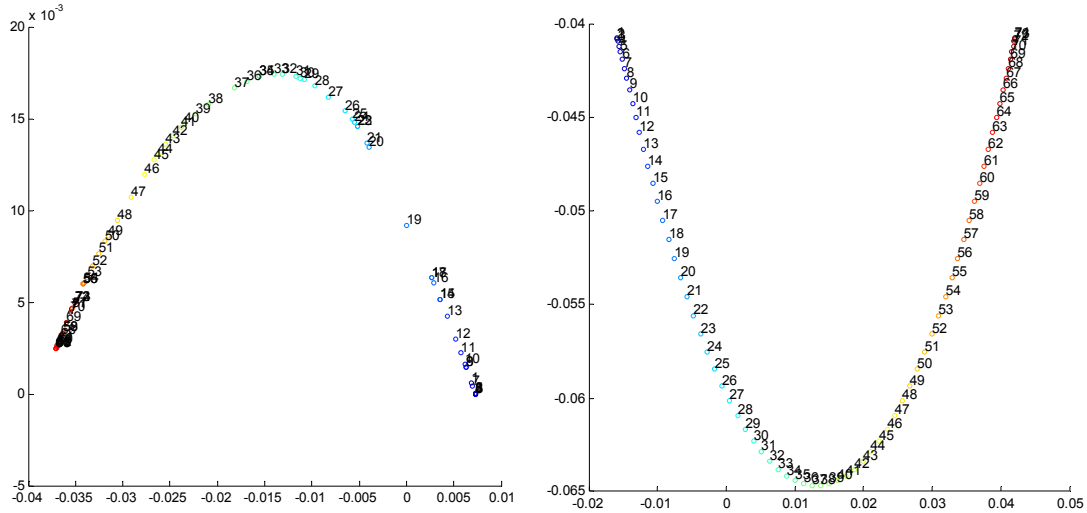


Figure B.1 – A two-dimensional representation of sd1 dataset. The adjacency graph was constructed using *KNN(K=5)* in the left plot and imposing *physical adjacency* in the right plot. Weighting of edges was done using simple-minded approach for both plots. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

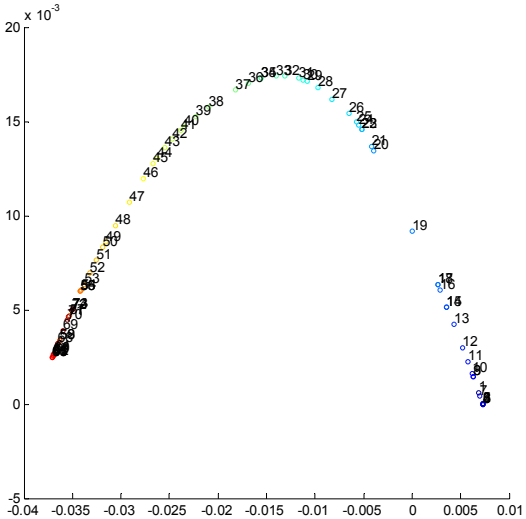


Figure B.2 – A two-dimensional representation of sd1 dataset. The adjacency graph was constructed using a combination KNN(K=5) and physical adjacency. Weighting of edges was done using simple-minded approach. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.1.2 'sd2' dataset

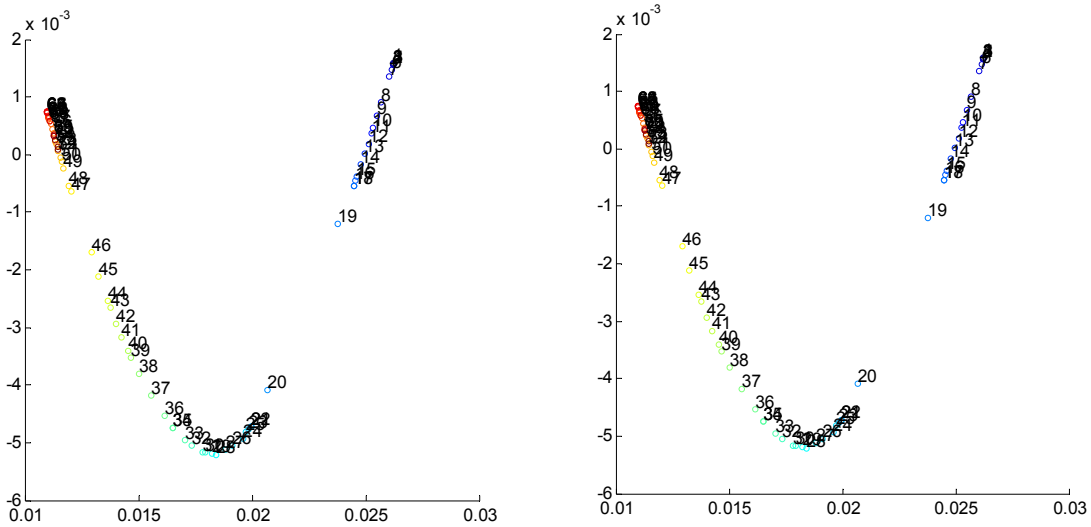


Figure B.3 – A two-dimensional representation of sd2 dataset. The adjacency graph was constructed using *KNN (K=5)* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

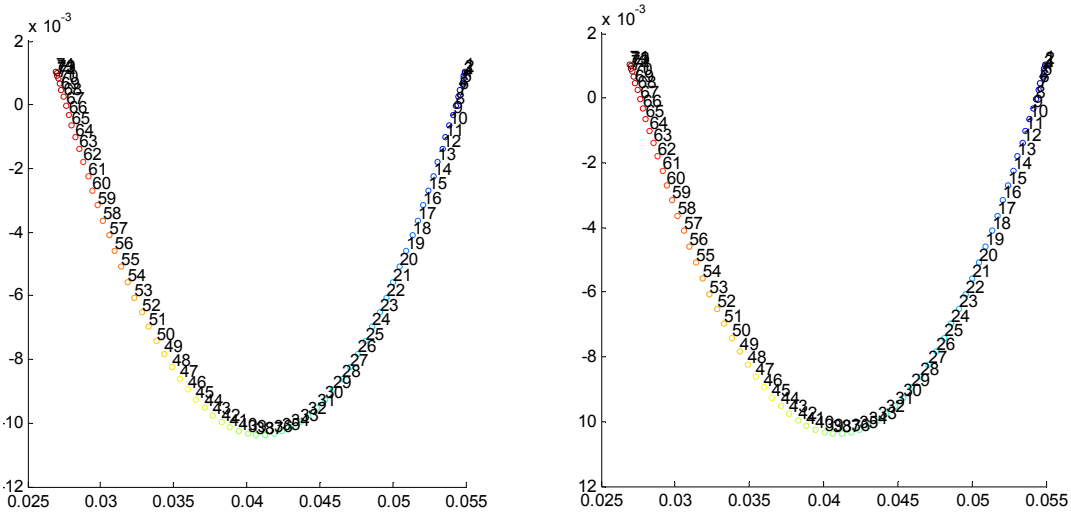


Figure B.4 – A two-dimensional representation of sd2 dataset. The adjacency graph was constructed using *physical adjacency* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

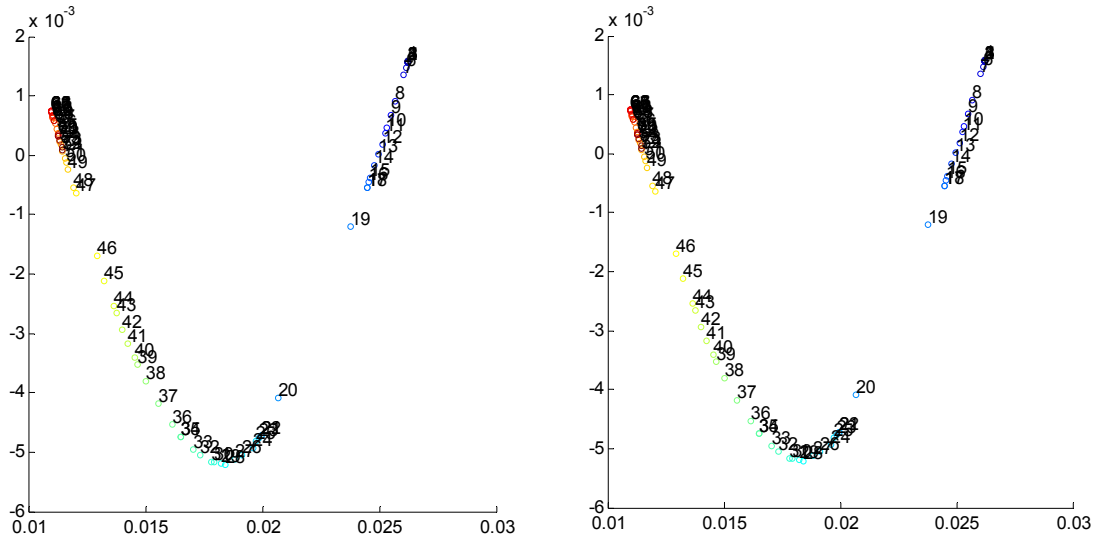


Figure B.5 – A two-dimensional representation of sd2 dataset. The adjacency graph was constructed using a combination KNN(K=5) and physical adjacency in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.1.3 'near' dataset

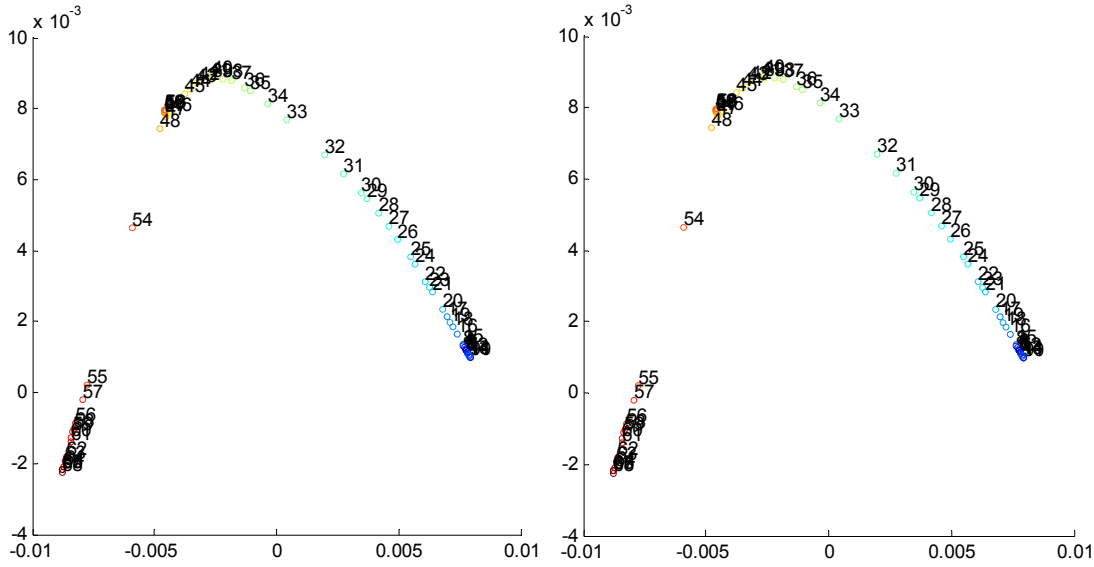


Figure B.6 – A two-dimensional representation of near dataset. The adjacency graph was constructed using *KNN (K=5)* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

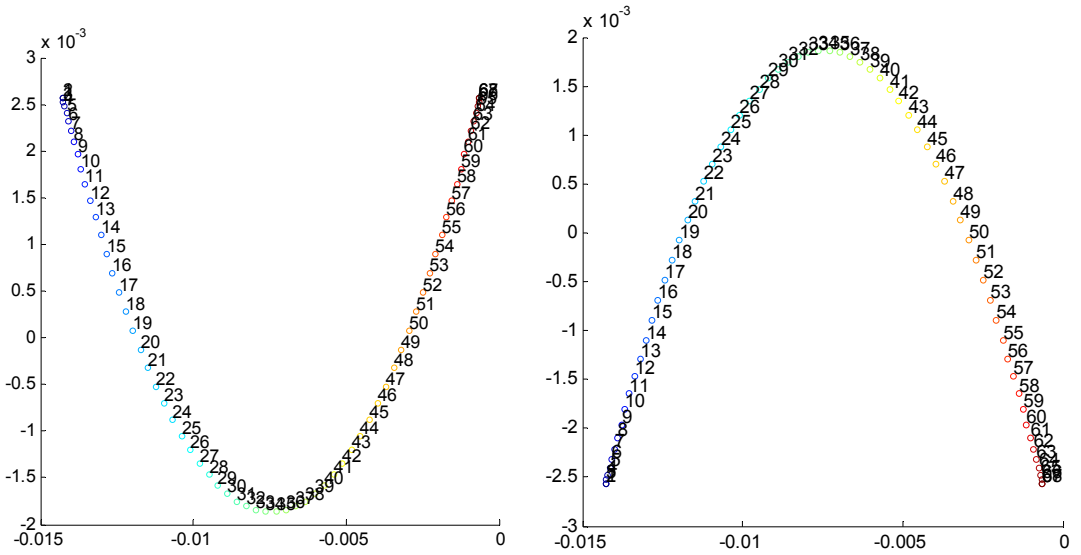


Figure B.7 – A two-dimensional representation of near dataset. The adjacency graph was constructed using *physical adjacency* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

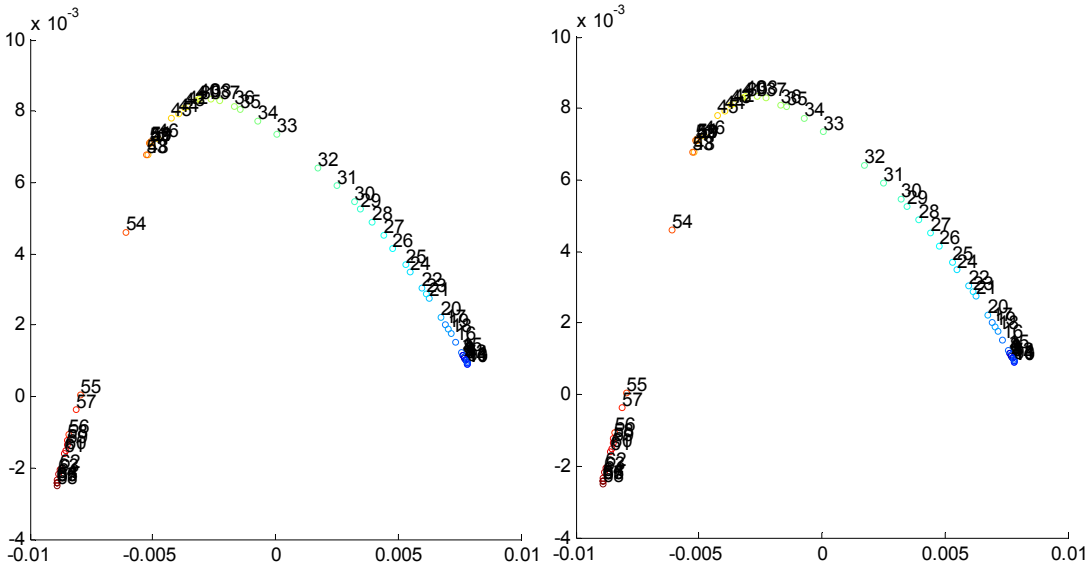


Figure B.8 – A two-dimensional representation of near dataset. The adjacency graph was constructed using a combination KNN(K=5) and physical adjacency in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.1.4 'far' dataset

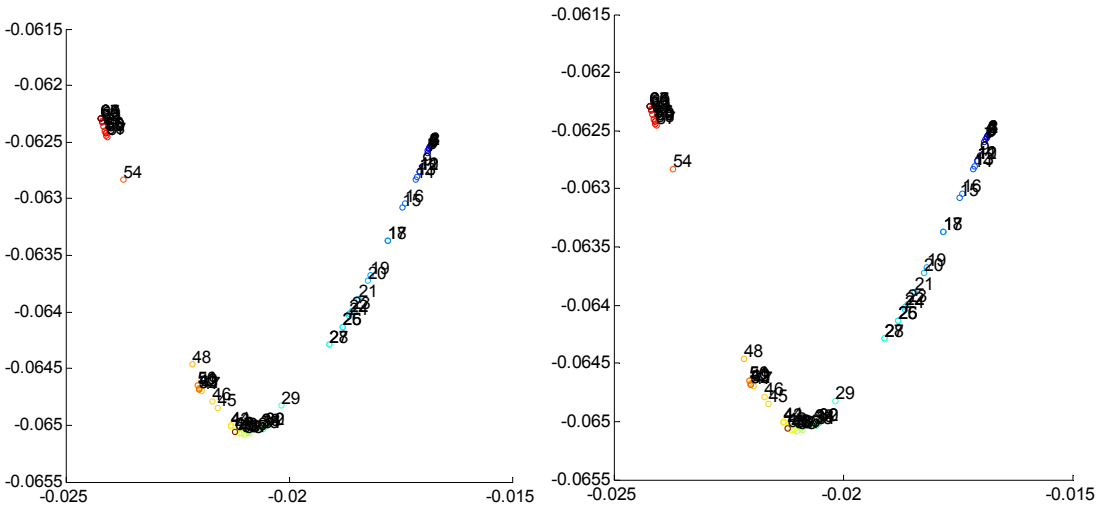


Figure B.9 – A two-dimensional representation of far dataset. The adjacency graph was constructed using *KNN (K=5)* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

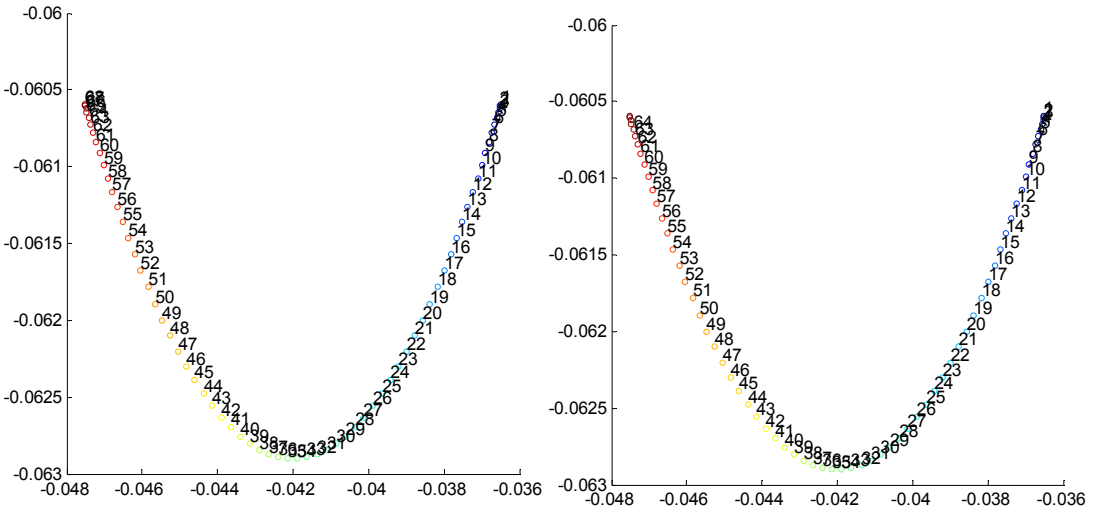


Figure B.10 – A two-dimensional representation of far dataset. The adjacency graph was constructed using *physical adjacency* in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

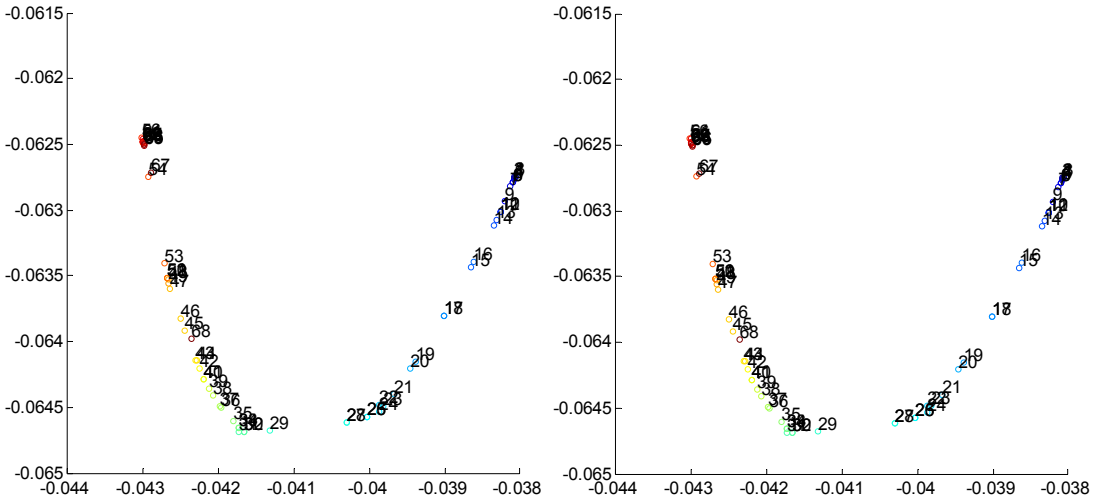


Figure B.11 – A two-dimensional representation of far dataset. The adjacency graph was constructed using a combination KNN(K=5) and physical adjacency in both plots. Weighting of edges was done by *heat kernel (t=5)* in the left plot and by simple-minded approach in the right plot. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.2 Behaviour of LPP on difference in pixels across the edge

B.2.1 Plots of LPP

B.2.1.1 LPP on edge difference values independently

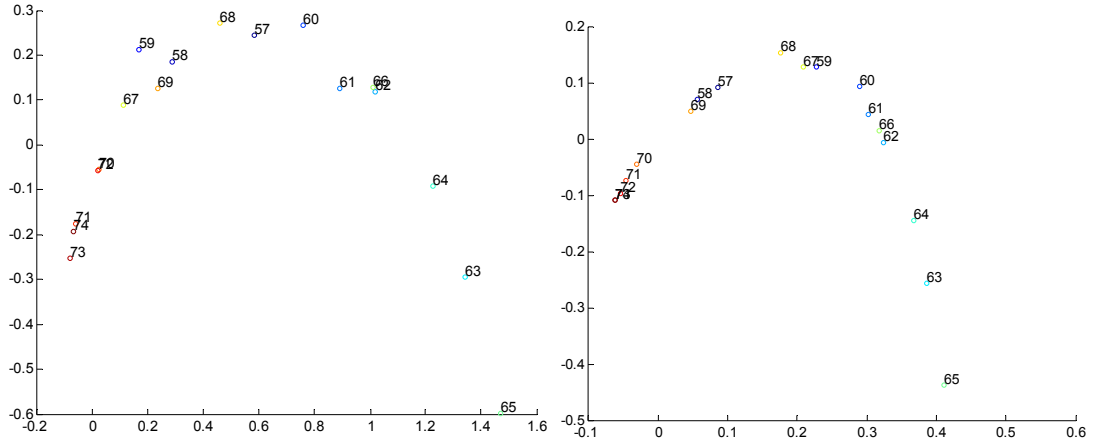


Figure B.12 A two-dimensional representation of edge difference values V_1 (left plot) and V_3 (right plot) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

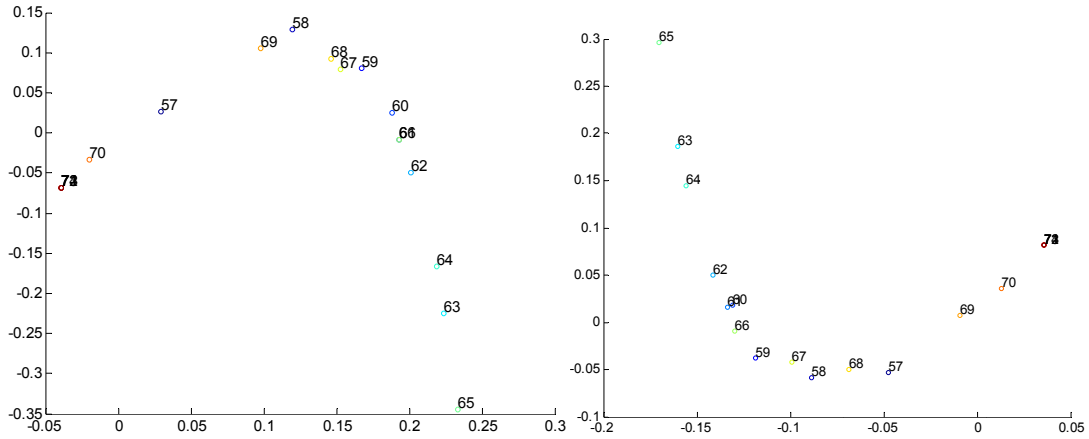


Figure B.13 A two-dimensional representation of edge difference values V_4 (left plot) and V_6 (right plot) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

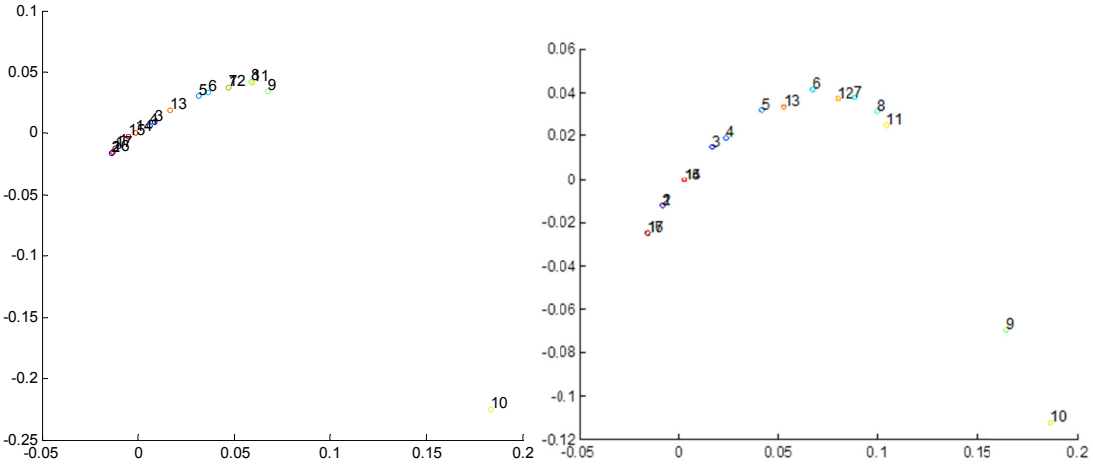


Figure B.14 A two-dimensional representation of edge difference values V_2 (left plot) and V_6 (right plot) for Boxes data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

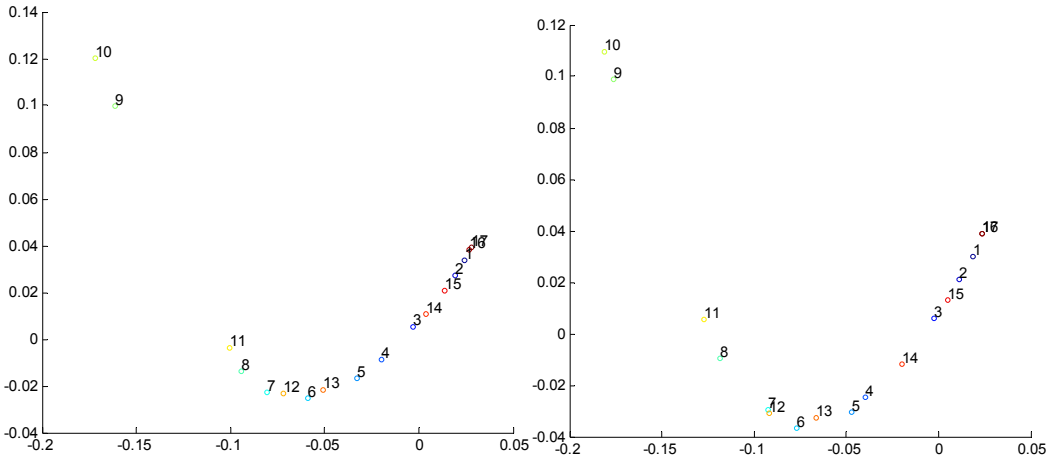


Figure B.15 A two-dimensional representation of edge difference values V_8 (left plot) and V_{10} (right plot) for Boxes data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.2.1.2 LPP on edge difference values in aggregate

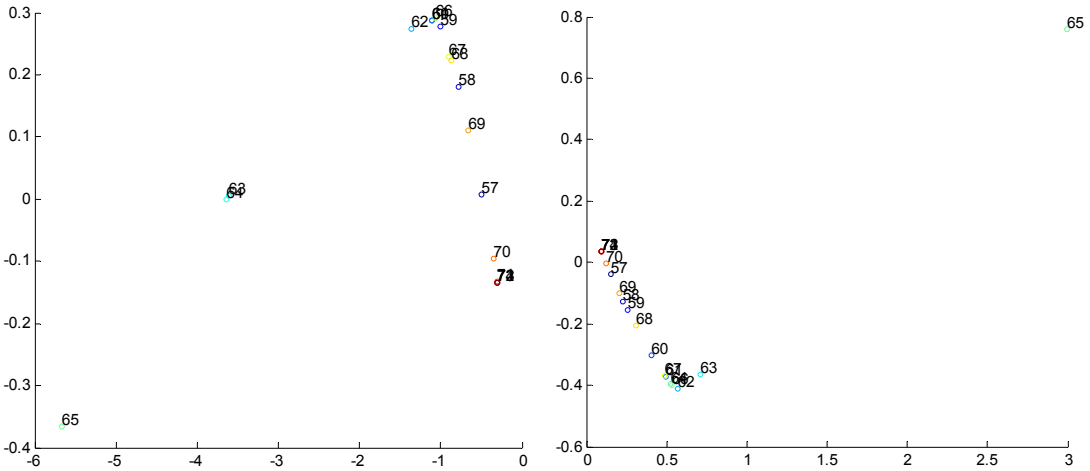


Figure B.16 A two-dimensional representation of edge difference values V(Edge2) (left plot) andV(Edge3) (right plot) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

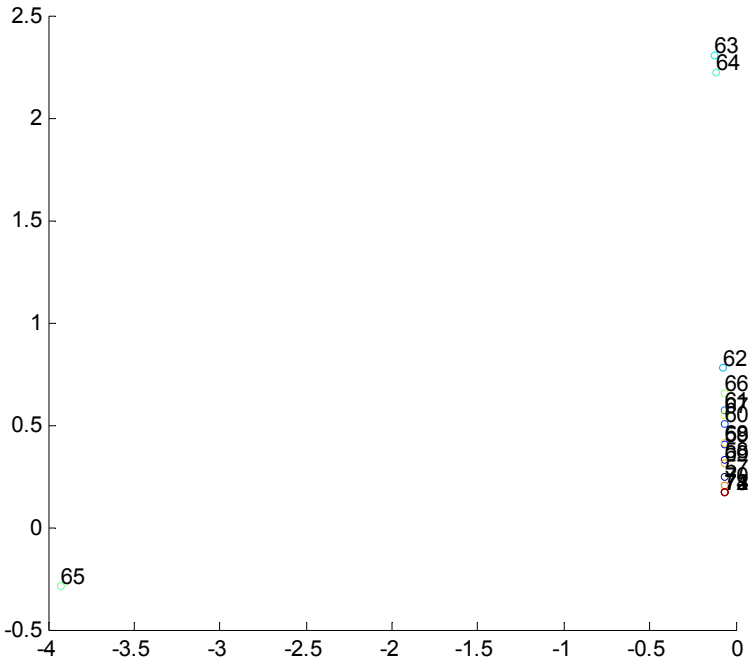


Figure B.17 A two-dimensional representation of edge difference values V($comp$) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

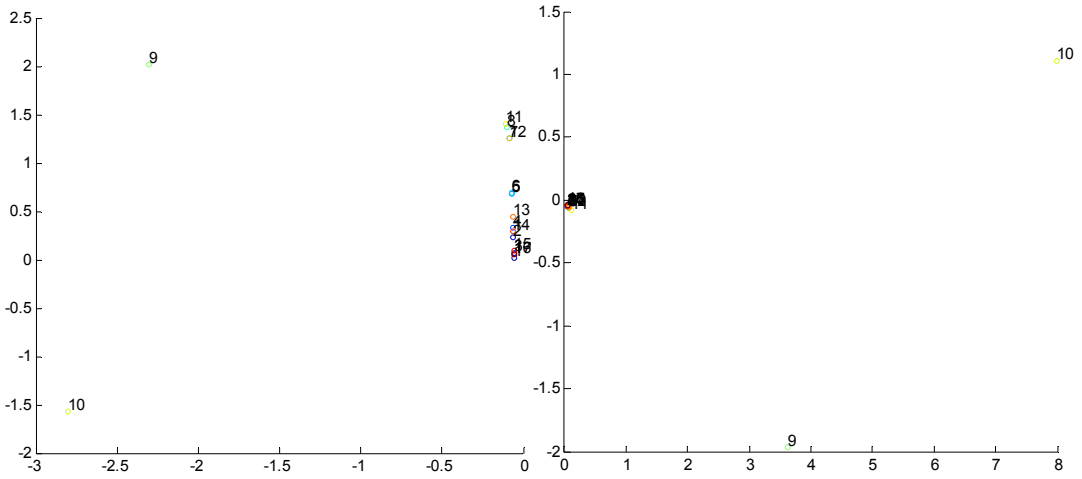


Figure B.18 A two-dimensional representation of edge difference values V(Edge1) (left plot) and V(Edge2) (right plot) for Boxes data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

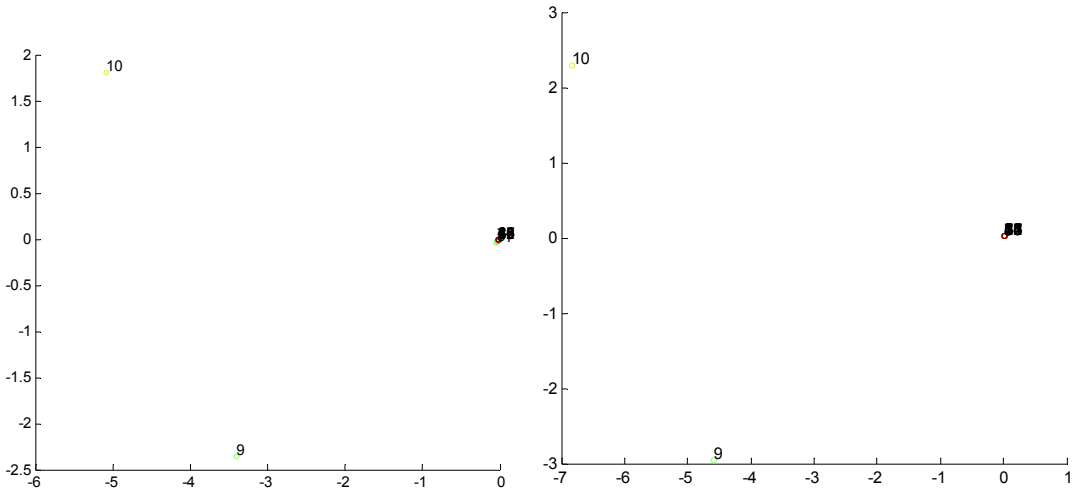


Figure B.19 A two-dimensional representation of edge difference values V(Edge3) (left plot) and V($comp$) (right plot) for sd1 data. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.2.2 Trajectory Analysis in the first dimension of LPP

B.2.2.1 LPP on edge difference values independently

sd1	V_1				V_2			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	0.580483	58-57	-0.2961		0.346525	58-57	-0.1826	
58	0.284416	59-58	-0.1202		0.163891	59-58	0.0452	✓
59	0.164247	60-59	0.5954	✓	0.209124	60-59	0.2962	
60	0.759693	61-60	0.1285		0.505313	61-60	0.1210	
61	0.888194	62-61	0.1296		0.626343	62-61	0.0927	
62	1.017768	63-62	0.3238		0.719073	63-62	0.3061	
63	1.341557	64-63	-0.1148	✓	1.025162	64-63	-0.0997	✓
64	1.226798	65-64	0.2412	✓	0.925467	65-64	0.2344	✓
65	**1.468046**	**66-65**	**-0.4589**	✓	**1.159823**	**66-65**	**-0.4573**	✓
66	1.009178	67-66	-0.9004		0.702566	67-66	-0.5017	
67	0.108767	68-67	0.3482	✓	0.200888	68-67	-0.0552	
68	0.456962	69-68	-0.2217	✓	0.145697	69-68	0.0115	✓
69	0.235226	70-69	-0.2148		0.157154	70-69	-0.1148	✓
70	0.020386	71-70	-0.0808		0.042304	71-70	-0.0113	
71	-0.060387	72-71	0.0801	✓	0.030979	72-71	-0.0641	
72	0.019749	73-72	-0.1027	✓	-0.033125	73-72	0.0744	✓
73	-0.082977	74-73	0.0144	✓	0.041265	74-73	-0.0744	✓
74	-0.068584				-0.033120			

sd1	V_3				V_4			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	0.085399	58-57	-0.0289		0.028315	58-57	0.0907	
58	0.056465	59-58	0.1706	✓	0.118986	59-58	0.0477	
59	0.227113	60-59	0.0624		0.166639	60-59	0.0211	
60	0.289515	61-60	0.0123		0.187768	61-60	0.0045	
61	0.301847	62-61	0.0221		0.192232	62-61	0.0082	
62	0.323964	63-62	0.0623		0.200481	63-62	0.0228	
63	0.386218	64-63	-0.0191	✓	0.223282	64-63	-0.0055	✓
64	0.367119	65-64	0.0442	✓	0.217822	65-64	0.0149	✓
65	**0.411357**	**66-65**	**-0.0930**	✓	**0.232710**	**66-65**	**-0.0405**	✓
66	0.318371	67-66	-0.1099		0.192221	67-66	-0.0406	
67	0.208488	68-67	-0.0326		0.151648	68-67	-0.0062	
68	0.175861	69-68	-0.1293		0.145496	69-68	-0.0485	
69	0.046529	70-69	-0.0781		0.096974	70-69	-0.1177	
70	-0.031533	71-70	-0.0152		-0.020715	71-70	-0.0194	
71	-0.046752	72-71	-0.0072		-0.040085	72-71	0.0000	✓
72	-0.053960	73-72	-0.0073		-0.040120	73-72	0.0000	
73	-0.061267	74-73	0.0000	✓	-0.040135	74-73	-0.0001	✓
74	-0.061300				-0.040203			

sd1	V_5				V_6			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	-0.038583	58-57	-0.0264		-0.047736	58-57	-0.0409	
58	-0.064997	59-58	-0.0437		-0.088644	59-58	-0.0301	
59	-0.108703	60-59	-0.0092		-0.118765	60-59	-0.0120	
60	-0.117880	61-60	-0.0029		-0.130740	61-60	-0.0026	
61	-0.120811	62-61	-0.0064		-0.133373	62-61	-0.0081	
62	-0.127219	63-62	-0.0156		-0.141465	63-62	-0.0189	
63	-0.142792	64-63	0.0033	✓	-0.160382	64-63	0.0043	✓
64	-0.139493	65-64	-0.0106	✓	-0.156101	65-64	-0.0140	✓
65	**-0.150084**	**66-65**	**0.0294**	✓	**-0.170058**	**66-65**	**0.0404**	✓
66	-0.120706	67-66	0.0310		-0.129656	67-66	0.0304	
67	-0.089657	68-67	0.0046		-0.099243	68-67	0.0309	
68	-0.085044	69-68	0.0850		-0.068391	69-68	0.0590	
69	-0.000090	70-69	0.0356		-0.009416	70-69	0.0223	
70	0.035547	71-70	0.0120		0.012873	71-70	0.0230	
71	0.047536	72-71	0.0001		0.035912	72-71	0.0001	
72	0.047588	73-72	0.0000		0.035969	73-72	0.0000	
73	0.047595	74-73	0.0000		0.035981	74-73	0.0000	
74	0.047641				0.036012			

sd1	V_7				V_8			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	-0.030774	58-57	-0.0872		0.026208	58-57	0.0940	
58	-0.118012	59-58	-0.0385		0.120178	59-58	0.0418	
59	-0.156523	60-59	-0.0079		0.161982	60-59	0.0085	
60	-0.164409	61-60	0.0000	✓	0.170443	61-60	0.0002	
61	-0.164399	62-61	-0.0086	✓	0.170621	62-61	0.0097	
62	-0.172960	63-62	-0.0188		0.180327	63-62	0.0217	
63	-0.191788	64-63	0.0003	✓	0.202075	64-63	0.0004	
64	-0.191439	65-64	-0.0087	✓	0.202483	65-64	0.0092	
65	**-0.200188**	**66-65**	**0.0435**	✓	**0.211707**	**66-65**	**-0.0494**	✓
66	-0.156698	67-66	0.0329		0.162270	67-66	-0.0358	
67	-0.123784	68-67	0.0578		0.126514	68-67	-0.0623	
68	-0.065988	69-68	0.0001		0.064188	69-68	-0.0001	
69	-0.065891	70-69	0.0586		0.064052	70-69	-0.0633	
70	-0.007274	71-70	0.0184		0.000748	71-70	-0.0199	
71	0.011144	72-71	0.0000		-0.019138	72-71	-0.0001	
72	0.011190	73-72	0.0000		-0.019209	73-72	0.0000	✓
73	0.011221	74-73	0.0000		-0.019237	74-73	0.0000	
74	0.011246				-0.019275			

sd1	V_9				V_{10}			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	0.013387	58-57	0.0745		-0.004539	58-57	0.0460	
58	0.087841	59-58	0.0544		0.041450	59-58	0.1290	
59	0.142253	60-59	0.0164		0.170469	60-59	0.0116	
60	0.158669	61-60	0.0072		0.182024	61-60	0.0109	
61	0.165912	62-61	0.0036		0.192971	62-61	0.0052	
62	0.169480	63-62	0.0242		0.198141	63-62	0.0332	
63	0.193663	64-63	0.0019		0.231326	64-63	0.0039	
64	0.195571	65-64	0.0076		0.235208	65-64	0.0090	
65	**0.203150**	**66-65**	**-0.0527**	✓	**0.244223**	**66-65**	**-0.0730**	✓
66	0.150484	67-66	-0.0314		0.171173	67-66	-0.0473	
67	0.119058	68-67	-0.0133		0.123849	68-67	-0.0190	
68	0.105795	69-68	-0.0488		0.104821	69-68	-0.0635	
69	0.057013	70-69	-0.0842		0.041333	70-69	-0.0771	
70	-0.027169	71-70	-0.0136		-0.035795	71-70	-0.0241	
71	-0.040787	72-71	-0.0001		-0.059895	72-71	-0.0001	
72	-0.040873	73-72	0.0000	✓	-0.060001	73-72	0.0000	✓
73	-0.040904	74-73	-0.0001	✓	-0.060021	74-73	-0.0001	✓
74	-0.040960				-0.060102			

Boxes	V_1				V_2			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.002912	2-1	-0.0062		-0.010011	2-1	-0.0037	
2	-0.009130	3-2	0.0280	✓	-0.013666	3-2	0.0221	✓
3	0.018909	4-3	-0.0121	✓	0.008409	4-3	-0.0020	✓
4	0.006829	5-4	0.0293	✓	0.006404	5-4	0.0253	✓
5	0.036131	6-5	0.0142		0.031662	6-5	0.0049	
6	0.050370	7-6	0.0008		0.036597	7-6	0.0107	
7	0.051189	8-7	0.0097		0.047266	8-7	0.0117	
8	0.060842	9-8	0.0049		0.058932	9-8	0.0085	
9	0.065701	10-9	0.0592		0.067428	10-9	0.1160	
10	**0.124933**	**11-10**	**-0.0639**	✓	**0.183429**	**11-10**	**-0.1241**	✓
11	0.060985	12-11	-0.0115		0.059331	12-11	-0.0124	
12	0.049518	13-12	-0.0286		0.046938	13-12	-0.0305	
13	0.020952	14-13	-0.0229		0.016461	14-13	-0.0177	
14	-0.001952	15-14	-0.0096		-0.001222	15-14	-0.0035	
15	-0.011562	16-15	0.0004	✓	-0.004753	16-15	-0.0086	
16	-0.011193	17-16	-0.0021	✓	-0.013307	17-16	0.0017	✓
17	-0.013308				-0.011587			

Boxes	V_3				V_4			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	0.013548	2-1	-0.0060		0.013499	2-1	0.0014	
2	0.007521	3-2	-0.0107		0.014912	3-2	-0.0201	✓
3	-0.003196	4-3	0.0020	✓	-0.005143	4-3	-0.0050	
4	-0.001242	5-4	-0.0189	✓	-0.010171	5-4	-0.0131	
5	-0.020176	6-5	-0.0193		-0.023279	6-5	-0.0177	
6	-0.039438	7-6	-0.0127		-0.040974	7-6	-0.0149	
7	-0.052104	8-7	-0.0052		-0.055918	8-7	-0.0060	
8	-0.057338	9-8	-0.0098		-0.061947	9-8	-0.0228	
9	-0.067113	10-9	-0.0353		-0.084743	10-9	-0.0293	
10	**-0.102388**	**11-10**	**0.0445**	✓	**-0.114074**	**11-10**	**0.0509**	✓
11	-0.057930	12-11	0.0120		-0.063174	12-11	0.0131	
12	-0.045920	13-12	0.0177		-0.050029	13-12	0.0187	
13	-0.028210	14-13	0.0287		-0.031376	14-13	0.0344	
14	0.000528	15-14	0.0071		0.003036	15-14	0.0019	
15	0.007601	16-15	0.0045		0.004950	16-15	0.0108	
16	0.012084	17-16	-0.0004	✓	0.015782	17-16	0.0027	
17	0.011647				0.018479			

Boxes	V_5				V_6			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.014811	2-1	0.0000		-0.008004	2-1	0.0000	
2	-0.014811	3-2	0.0213		-0.008004	3-2	0.0253	
3	0.006520	4-3	0.0056		0.017315	4-3	0.0067	
4	0.012134	5-4	0.0149		0.023982	5-4	0.0176	
5	0.027003	6-5	0.0214		0.041626	6-5	0.0256	
6	0.048435	7-6	0.0180		0.067199	7-6	0.0215	
7	0.066403	8-7	0.0090		0.088682	8-7	0.0115	
8	0.075373	9-8	0.0527		0.100169	9-8	0.0647	
9	0.128065	10-9	0.0321		0.164901	10-9	0.0220	
10	**0.160158**	**11-10**	**-0.0816**	✓	**0.186853**	**11-10**	**-0.0820**	✓
11	0.078561	12-11	-0.0196		0.104841	12-11	-0.0251	
12	0.058981	13-12	-0.0224		0.079771	13-12	-0.0268	
13	0.036594	14-13	-0.0423		0.052973	14-13	-0.0501	
14	-0.005665	15-14	0.0000	✓	0.002859	15-14	0.0000	✓
15	-0.005671	16-15	-0.0154	✓	0.002853	16-15	-0.0184	✓
16	-0.021097	17-16	0.0000	✓	-0.015503	17-16	0.0000	✓
17	-0.021097				-0.015503			

Boxes	V_7				V_8			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.029397	2-1	0.0051		0.024549	2-1	-0.0051	
2	-0.024271	3-2	0.0274		0.019496	3-2	-0.0224	
3	0.003161	4-3	0.0123		-0.002872	4-3	-0.0171	
4	0.015419	5-4	0.0127		-0.019923	5-4	-0.0130	
5	0.028145	6-5	0.0260		-0.032874	6-5	-0.0261	
6	0.054121	7-6	0.0200		-0.059013	7-6	-0.0216	
7	0.074107	8-7	0.0117		-0.080582	8-7	-0.0135	
8	0.085798	9-8	0.0684		-0.094068	9-8	-0.0672	
9	0.154206	10-9	0.0151		-0.161257	10-9	-0.0103	
10	**0.169327**	**11-10**	**-0.0780**	✓	**-0.171570**	**11-10**	**0.0709**	✓
11	0.091354	12-11	-0.0254		-0.100672	12-11	0.0286	
12	0.065940	13-12	-0.0201		-0.072099	13-12	0.0213	
13	0.045850	14-13	-0.0579		-0.050847	14-13	0.0546	
14	-0.012045	15-14	-0.0066		0.003765	15-14	0.0101	
15	-0.018632	16-15	-0.0152		0.013880	16-15	0.0128	
16	-0.033836	17-16	0.0000	✓	0.026645	17-16	0.0013	
17	-0.033835				0.027945			

Boxes	V_9				V_{10}			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.021821	2-1	0.0068		0.018367	2-1	-0.0069	
2	-0.015030	3-2	0.0135		0.011474	3-2	-0.0137	
3	-0.001530	4-3	0.0371		-0.002213	4-3	-0.0376	
4	0.035601	5-4	0.0073		-0.039816	5-4	-0.0075	
5	0.042919	6-5	0.0291		-0.047310	6-5	-0.0296	
6	0.072038	7-6	0.0148		-0.076898	7-6	-0.0156	
7	0.086876	8-7	0.0239		-0.092478	8-7	-0.0257	
8	0.110797	9-8	0.0664		-0.118226	9-8	-0.0581	
9	0.177223	10-9	0.0069		-0.176302	10-9	-0.0047	
10	**0.184111**	**11-10**	**-0.0650**	✓	**-0.180991**	**11-10**	**0.0540**	✓
11	0.119129	12-11	-0.0331		-0.127001	12-11	0.0355	
12	0.086011	13-12	-0.0238		-0.091519	13-12	0.0250	
13	0.062177	14-13	-0.0464		-0.066512	14-13	0.0468	
14	0.015805	15-14	-0.0246		-0.019758	15-14	0.0249	
15	-0.008794	16-15	-0.0182		0.005138	16-15	0.0185	
16	-0.026978	17-16	0.0000	✓	0.023612	17-16	0.0000	
17	-0.026979				0.023614			

B.2.2.2 LPP on edge difference values in aggregate

sd1	V(Edge1)				V(Edge2)			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	-0.339239	58-57	-0.2809		-0.503658	58-57	-0.2746	
58	-0.620137	59-58	-0.0749		-0.778236	59-58	-0.2332	
59	-0.695021	60-59	-0.5141		-1.011434	60-59	-0.0934	
60	-1.209101	61-60	-0.3751		-1.104794	61-60	-0.0079	
61	-1.584172	62-61	-0.3516		-1.112647	62-61	-0.2535	
62	-1.935786	63-62	-0.2790		-1.366157	63-62	-2.2400	
63	-2.214738	64-63	-0.0510		-3.606135	64-63	-0.0413	
64	-2.265763	65-64	-1.2632		-3.647388	65-64	-2.0135	
65	**-3.529002**	**66-65**	**1.4189**	✓	**-5.660856**	**66-65**	**4.5980**	✓
66	-2.110094	67-66	0.3359		-1.062812	67-66	0.1615	
67	-1.774155	68-67	0.6791		-0.901304	68-67	0.0327	
68	-1.095095	69-68	0.3086		-0.868569	69-68	0.2047	
69	-0.786485	70-69	0.4482		-0.663851	70-69	0.3084	
70	-0.338256	71-70	0.1059		-0.355452	71-70	0.0453	
71	-0.232405	72-71	0.0545		-0.310123	72-71	0.0011	
72	-0.177912	73-72	0.0009		-0.308994	73-72	0.0003	
73	-0.176989	74-73	-0.0003	✓	-0.308716	74-73	0.0006	
74	-0.177322				-0.308161			

sd1	V(Edge3)				V($comp$)			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
57	0.152555	58-57	0.0772		-0.060574	58-57	-0.0010	
58	0.229784	59-58	0.0242		-0.061615	59-58	-0.0011	
59	0.253941	60-59	0.1490		-0.062676	60-59	-0.0016	
60	0.402942	61-60	0.0909		-0.064308	61-60	-0.0011	
61	0.493868	62-61	0.0765		-0.065455	62-61	-0.0053	
62	0.570400	63-62	0.1425		-0.070790	63-62	-0.0578	
63	0.712857	64-63	-0.1869	✓	-0.128621	64-63	0.0189	✓
64	0.525931	65-64	2.4697	✓	-0.109692	65-64	-3.8159	✓
65	**2.995641**	**66-65**	**-2.4601**	✓	**-3.925630**	**66-65**	**3.8583**	✓
66	0.535580	67-66	-0.0496		-0.067318	67-66	0.0023	
67	0.485978	68-67	-0.1805		-0.064981	68-67	0.0023	
68	0.305488	69-68	-0.1011		-0.062706	69-68	0.0014	
69	0.204406	70-69	-0.0819		-0.061347	70-69	0.0012	
70	0.122551	71-70	-0.0261		-0.060123	71-70	0.0003	
71	0.096450	72-71	0.0000	✓	-0.059831	72-71	0.0000	
72	0.096415	73-72	-0.0001	✓	-0.059828	73-72	0.0000	
73	0.096316	74-73	-0.0001		-0.059827	74-73	0.0000	
74	0.096206				-0.059828			

Boxes	V(Edge1)				V(Edge2)			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.057668	2-1	-0.0025		0.050722	2-1	0.0010	
2	-0.060134	3-2	0.0022	✓	0.051701	3-2	0.0020	
3	-0.057901	4-3	-0.0043	✓	0.053712	4-3	0.0042	
4	-0.062163	5-4	-0.0087		0.057907	5-4	0.0011	
5	-0.070827	6-5	-0.0002		0.058961	6-5	0.0084	
6	-0.070992	7-6	-0.0183		0.067355	7-6	0.0042	
7	-0.089313	8-7	-0.0096		0.071523	8-7	0.0076	
8	-0.098913	9-8	-2.2045		0.079124	9-8	3.5413	
9	-2.303372	10-9	-0.4982		3.620398	10-9	4.3635	
10	**-2.801588**	**11-10**	**2.6942**	✓	**7.983860**	**11-10**	**-7.8688**	✓
11	-0.107371	12-11	0.0182		0.115069	12-11	-0.0471	
12	-0.089166	13-12	0.0246		0.068007	13-12	-0.0054	
13	-0.064556	14-13	0.0032		0.062568	14-13	-0.0101	
14	-0.061333	15-14	0.0034		0.052465	15-14	-0.0017	
15	-0.057977	16-15	0.0002		0.050723	16-15	0.0000	✓
16	-0.057786	17-16	0.0000		0.050772	17-16	0.0000	
17	-0.057784				0.050772			

Boxes	V(Edge3)				V($comp$)			
Data-point	1st Dimension (LPP)	Displacement		Change in Direction	1st Dimension (LPP)	Displacement		Change in Direction
1	-0.018436	2-1	-0.0001		0.017776	2-1	0.0000	
2	-0.018568	3-2	-0.0018		0.017754	3-2	0.0000	
3	-0.020341	4-3	-0.0010		0.017710	4-3	-0.0001	✓
4	-0.021377	5-4	-0.0013		0.017617	5-4	-0.0001	
5	-0.022642	6-5	-0.0036		0.017543	6-5	-0.0002	
6	-0.026243	7-6	-0.0043		0.017354	7-6	-0.0003	
7	-0.030564	8-7	-0.0083		0.017013	8-7	-0.0007	
8	-0.038906	9-8	-3.3598		0.016299	9-8	-4.5971	
9	-3.398731	10-9	-1.6846		-4.580832	10-9	-2.2505	
10	**-5.083337**	**11-10**	**5.0243**	✓	**-6.831303**	**11-10**	**6.8456**	✓
11	-0.059053	12-11	0.0303		0.014290	12-11	0.0029	
12	-0.028720	13-12	0.0041		0.017157	13-12	0.0003	
13	-0.024629	14-13	0.0054		0.017454	14-13	0.0002	
14	-0.019204	15-14	0.0004		0.017700	15-14	0.0001	
15	-0.018811	16-15	0.0004		0.017768	16-15	0.0000	
16	-0.018433	17-16	0.0000		0.017773	17-16	0.0000	
17	-0.018434				0.017773			

B.2.3 Euclidean Distances

B.2.3.1 LPP on edge difference values independently

Euclidean Distances sd1	V_1					V_2				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.41624	0.42495	0.94258	2.08751	2.14269	0.22787	0.23888	0.75089	0.93026	1.85293
(58 to 59) + (59 to 60)	0.71562	0.72098	1.42726	2.00025	2.06949	0.34142	0.35289	0.86648	1.00496	1.80131
(59 to 60) + (60 to 61)	0.72395	0.78790	1.21841	1.29905	1.44761	0.41722	0.44101	0.93316	1.09557	1.46319
(60 to 61) + (61 to 62)	0.25807	0.31992	0.33491	0.54046	1.01074	0.21376	0.23576	0.28188	0.39322	1.32386
(61 to 62) + (62 to 63)	0.45336	0.65332	0.76216	0.97889	1.37597	0.39882	0.49341	0.54673	0.57787	1.39154
(62 to 63) + (63 to 64)	0.43855	0.75561	0.96274	1.02992	1.11043	0.40579	0.55863	0.69407	0.73950	0.96299
(63 to 64) + (64 to 65)	0.35601	0.79270	1.31131	1.47971	1.55932	0.33405	0.58391	0.99629	1.34944	1.40566
(64 to 65) + (65 to 66)	0.70012	1.42059	2.21008	2.33264	2.38802	0.69161	1.10362	1.77543	2.26582	2.32404
(65 to 66) + (66 to 67)	1.35928	1.76144	2.22417	2.23413	2.23630	0.95893	1.18008	1.61378	1.79416	1.81995
(66 to 67) + (67 to 68)	1.24861	1.29542	1.38196	1.38625	1.39889	0.55687	0.55792	0.67581	0.70087	0.70631
(67 to 68) + (68 to 69)	0.56993	0.66025	0.68829	0.69394	1.19499	0.06665	0.12978	0.25404	0.46806	0.47072
(68 to 69) + (69 to 70)	0.43658	0.54651	0.58006	0.70469	1.51218	0.12631	0.22789	0.32495	0.74158	0.89580
(69 to 70) + (70 to 71)	0.29561	0.42673	0.45127	0.62616	0.96359	0.12617	0.17354	0.35648	0.72566	1.00397
(70 to 71) + (71 to 72)	0.16091	0.28995	0.32753	0.98706	1.14249	0.07543	0.22522	0.61634	0.79639	0.92203
(71 to 72) + (72 to 73)	0.18286	0.36382	0.54252	1.31245	1.46185	0.13849	0.38582	0.61051	0.87506	0.88160
(72 to 73) + (73 to 74)	0.11712	0.28146	0.51487	0.69198	0.86437	0.14878	0.36096	0.37969	0.88254	0.89270

Euclidean Distances sd1	V_3					V_4				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.19958	0.21553	0.51342	1.32591	1.36283	0.13832	0.20507	0.40708	0.42179	0.54158
(58 to 59) + (59 to 60)	0.23305	0.25109	0.47403	0.79969	0.83601	0.06878	0.12825	0.35331	0.37453	0.50672
(59 to 60) + (60 to 61)	0.07473	0.12181	0.19180	0.27817	0.29870	0.02559	0.09404	0.18752	0.22780	0.26264
(60 to 61) + (61 to 62)	0.03445	0.10597	0.14126	0.14340	0.18639	0.01271	0.07465	0.09107	0.17784	0.24625
(61 to 62) + (62 to 63)	0.08437	0.31245	0.40810	0.41103	0.46145	0.03105	0.21789	0.29209	0.38106	0.49391
(62 to 63) + (63 to 64)	0.08135	0.36986	0.53290	0.54702	0.57522	0.02826	0.23494	0.36767	0.41120	0.48312
(63 to 64) + (64 to 65)	0.06334	0.40795	0.81807	0.89407	1.04355	0.02035	0.23742	0.53691	0.78095	0.81490
(64 to 65) + (65 to 66)	0.13722	0.75585	1.38319	1.48574	1.69905	0.05538	0.51817	0.98484	1.30746	1.34733
(65 to 66) + (66 to 67)	0.20287	0.61851	1.01944	1.06059	1.12572	0.08106	0.43531	0.70559	0.79189	0.80663
(66 to 67) + (67 to 68)	0.14251	0.19885	0.37334	0.38150	0.38954	0.04673	0.11186	0.17371	0.19776	0.24727
(67 to 68) + (68 to 69)	0.16196	0.20629	0.29382	0.31062	0.31881	0.05467	0.06565	0.17554	0.28242	0.47076
(68 to 69) + (69 to 70)	0.20739	0.28749	0.32393	0.33873	0.42970	0.16621	0.23277	0.38342	0.46806	0.61041
(69 to 70) + (70 to 71)	0.09328	0.15595	0.20055	0.35044	0.47842	0.13706	0.22206	0.31664	0.37612	0.53415
(70 to 71) + (71 to 72)	0.02243	0.05731	0.17183	0.63914	0.69082	0.01941	0.03958	0.07595	0.13385	0.28849
(71 to 72) + (72 to 73)	0.01451	0.03700	0.13407	0.68796	0.79848	0.00005	0.00012	0.00022	0.00044	0.00077
(72 to 73) + (73 to 74)	0.00734	0.01343	0.01474	0.24706	0.34410	0.00008	0.00021	0.00035	0.00074	0.00103

Euclidean Distances sd1	V_5					V_6				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.07012	0.09160	0.25796	0.36308	0.64416	0.07103	0.07801	0.26130	0.32397	0.44443
(58 to 59) + (59 to 60)	0.05288	0.09229	0.27895	0.31461	0.51198	0.04210	0.09377	0.18029	0.25553	0.37491
(59 to 60) + (60 to 61)	0.01211	0.05623	0.09683	0.13333	0.36283	0.01461	0.06021	0.07072	0.21002	0.33232
(60 to 61) + (61 to 62)	0.00934	0.05371	0.06617	0.08883	0.21457	0.01073	0.03757	0.05047	0.15479	0.30432
(61 to 62) + (62 to 63)	0.02198	0.15633	0.21912	0.24687	0.34625	0.02701	0.17206	0.23027	0.28336	0.31929
(62 to 63) + (63 to 64)	0.01887	0.16118	0.25892	0.27242	0.41575	0.02320	0.17899	0.26734	0.28421	0.30479
(63 to 64) + (64 to 65)	0.01389	0.16823	0.38434	0.52618	0.67961	0.01824	0.19285	0.39285	0.53902	0.59589
(64 to 65) + (65 to 66)	0.03997	0.38606	0.75249	0.95028	1.02048	0.05436	0.45914	0.79849	0.97588	1.04632
(65 to 66) + (66 to 67)	0.06043	0.32099	0.55185	0.60815	0.63386	0.07082	0.35269	0.59787	0.63896	0.67728
(66 to 67) + (67 to 68)	0.03566	0.07825	0.13775	0.14588	0.17769	0.06127	0.07686	0.21716	0.23957	0.30711
(67 to 68) + (68 to 69)	0.08957	0.10898	0.12488	0.14072	0.20496	0.08983	0.11415	0.19868	0.22238	0.36984
(68 to 69) + (69 to 70)	0.12059	0.15488	0.17032	0.19109	0.34012	0.08126	0.11815	0.13663	0.17599	0.30673
(69 to 70) + (70 to 71)	0.04763	0.08369	0.14233	0.19380	0.40376	0.04533	0.08719	0.15242	0.20866	0.25316
(70 to 71) + (71 to 72)	0.01204	0.02709	0.07211	0.11084	0.20840	0.02310	0.05159	0.11605	0.14122	0.14632
(71 to 72) + (72 to 73)	0.00006	0.00014	0.00031	0.00050	0.00088	0.00007	0.00019	0.00044	0.00057	0.00063
(72 to 73) + (73 to 74)	0.00005	0.00013	0.00026	0.00042	0.00073	0.00004	0.00012	0.00024	0.00028	0.00038

Euclidean Distances sd1	V_7					V_8				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.12575	0.14248	0.19847	0.20380	0.32187	0.13577	0.15144	0.20090	0.20614	0.29518
(58 to 59) + (59 to 60)	0.04640	0.08029	0.14442	0.16870	0.29958	0.05027	0.08183	0.13725	0.16077	0.26293
(59 to 60) + (60 to 61)	0.00790	0.03884	0.04720	0.07098	0.11017	0.00864	0.03870	0.04485	0.06843	0.09878
(60 to 61) + (61 to 62)	0.00857	0.02739	0.02739	0.03979	0.18768	0.00988	0.02951	0.02959	0.04576	0.16048
(61 to 62) + (62 to 63)	0.02739	0.12918	0.15626	0.17238	0.41578	0.03145	0.13626	0.15790	0.17745	0.36876
(62 to 63) + (63 to 64)	0.01918	0.10444	0.13285	0.14050	0.23620	0.02216	0.11021	0.13291	0.14104	0.21777
(63 to 64) + (64 to 65)	0.00910	0.07354	0.11430	0.31045	0.32669	0.00963	0.06899	0.09802	0.29343	0.31053
(64 to 65) + (65 to 66)	0.05224	0.30644	0.41153	0.64517	0.66138	0.05866	0.30922	0.38725	0.61990	0.63717
(65 to 66) + (66 to 67)	0.07640	0.26950	0.38618	0.43412	0.43443	0.08519	0.27935	0.37430	0.42255	0.42337
(66 to 67) + (67 to 68)	0.09071	0.09194	0.20753	0.21585	0.21615	0.09808	0.09874	0.19895	0.20712	0.20763
(67 to 68) + (68 to 69)	0.05789	0.05807	0.12264	0.12447	0.12489	0.06246	0.06251	0.11785	0.11955	0.12001
(68 to 69) + (69 to 70)	0.05871	0.09088	0.12980	0.12996	0.14101	0.06344	0.09351	0.12801	0.12827	0.13683
(69 to 70) + (70 to 71)	0.07704	0.13066	0.21465	0.21935	0.27365	0.08319	0.13310	0.20838	0.21389	0.25665
(70 to 71) + (71 to 72)	0.01846	0.04000	0.08519	0.08979	0.13343	0.01996	0.03991	0.08083	0.08619	0.12086
(71 to 72) + (72 to 73)	0.00008	0.00018	0.00034	0.00038	0.00050	0.00010	0.00022	0.00041	0.00046	0.00048
(72 to 73) + (73 to 74)	0.00006	0.00014	0.00025	0.00029	0.00040	0.00007	0.00016	0.00027	0.00033	0.00035

Euclidean Distances sd1	V_9					V_{10}				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.12887	0.15844	0.25700	0.26436	0.31442	0.17501	0.20205	0.34923	0.36644	0.37797
(58 to 59) + (59 to 60)	0.07083	0.10663	0.18998	0.21391	0.26722	0.14057	0.17046	0.27843	0.29994	0.35716
(59 to 60) + (60 to 61)	0.02366	0.06271	0.07072	0.11229	0.16829	0.02250	0.06072	0.07269	0.13185	0.26749
(60 to 61) + (61 to 62)	0.01081	0.03174	0.04244	0.06165	0.08166	0.01612	0.03927	0.04889	0.09819	0.19373
(61 to 62) + (62 to 63)	0.02775	0.12365	0.15450	0.15646	0.16727	0.03835	0.15077	0.17955	0.18004	0.25202
(62 to 63) + (63 to 64)	0.02609	0.11944	0.15395	0.17079	0.18549	0.03707	0.15140	0.18454	0.21776	0.28978
(63 to 64) + (64 to 65)	0.00949	0.05971	0.10238	0.22464	0.29737	0.01290	0.07062	0.09947	0.25125	0.37152
(64 to 65) + (65 to 66)	0.06025	0.28165	0.38048	0.50789	0.57549	0.08206	0.32968	0.40116	0.54139	0.66641
(65 to 66) + (66 to 67)	0.08409	0.26699	0.38001	0.40979	0.44740	0.12037	0.32728	0.42370	0.45622	0.47825
(66 to 67) + (67 to 68)	0.04469	0.05368	0.12087	0.15883	0.20351	0.06635	0.07130	0.13103	0.20337	0.25573
(67 to 68) + (68 to 69)	0.06204	0.06989	0.12519	0.21440	0.22761	0.08252	0.08911	0.14343	0.25364	0.34153
(68 to 69) + (69 to 70)	0.13296	0.16949	0.24491	0.30684	0.31114	0.14062	0.17985	0.25780	0.30754	0.34684
(69 to 70) + (70 to 71)	0.09780	0.14354	0.20360	0.24670	0.28754	0.10123	0.16166	0.24196	0.25972	0.28147
(70 to 71) + (71 to 72)	0.01370	0.02400	0.04605	0.08789	0.12876	0.02421	0.04747	0.09017	0.10733	0.12550
(71 to 72) + (72 to 73)	0.00012	0.00022	0.00038	0.00071	0.00083	0.00013	0.00026	0.00048	0.00058	0.00062
(72 to 73) + (73 to 74)	0.00009	0.00017	0.00028	0.00057	0.00066	0.00010	0.00023	0.00040	0.00052	0.00054

Euclidean Distances Boxes	V_1					V_2				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.03426	0.04903	0.07475	0.15448	0.19098	0.02573	0.04049	0.13475	0.14442	0.15223
(2 to 3) + (3 to 4)	0.04012	0.05731	0.09009	0.18723	0.18945	0.02408	0.03733	0.14719	0.18628	0.19083
(3 to 4) + (4 to 5)	0.04138	0.05715	0.11089	0.17072	0.19040	0.02726	0.03893	0.11680	0.16885	0.18691
(4 to 5) + (5 to 6)	0.04354	0.05634	0.07922	0.09993	0.14276	0.03019	0.04080	0.07976	0.09913	0.14403
(5 to 6) + (6 to 7)	0.01506	0.02133	0.02278	0.02638	0.05044	0.01560	0.01691	0.03745	0.05188	0.07988
(6 to 7) + (7 to 8)	0.01047	0.01446	0.02460	0.04376	0.04639	0.02233	0.02385	0.05208	0.06446	0.07866
(7 to 8) + (8 to 9)	0.01451	0.01898	0.03079	0.06046	0.07070	0.02016	0.02404	0.06039	0.06085	0.10893
(8 to 9) + (9 to 10)	0.06409	0.42737	0.43146	0.44625	0.45438	0.12450	0.29547	0.31588	0.31819	0.35599
(9 to 10) + (10 to 11)	0.12318	0.84398	0.84818	0.85445	0.85496	0.24010	0.57780	0.59064	0.59404	0.59712
(10 to 11) + (11 to 12)	0.07541	0.43753	0.46208	0.46428	0.46540	0.13649	0.30700	0.31622	0.33607	0.34637
(11 to 12) + (12 to 13)	0.04003	0.04288	0.07189	0.08661	0.08872	0.04287	0.04867	0.07461	0.09315	0.10334
(12 to 13) + (13 to 14)	0.05147	0.06354	0.07234	0.10186	0.11027	0.04816	0.06078	0.10413	0.10476	0.11687
(13 to 14) + (14 to 15)	0.03251	0.04962	0.10520	0.12706	0.15886	0.02121	0.03016	0.06088	0.06635	0.08156
(14 to 15) + (15 to 16)	0.00998	0.01987	0.08633	0.09607	0.13265	0.01209	0.02000	0.03765	0.06155	0.06538
(15 to 16) + (16 to 17)	0.00248	0.00577	0.03556	0.06235	0.07514	0.01027	0.01855	0.03167	0.07025	0.07541

Euclidean Distances Boxes	V_3					V_4				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.01674	0.02881	0.08048	0.08417	0.10882	0.02147	0.03759	0.07859	0.12532	0.17806
(2 to 3) + (3 to 4)	0.01267	0.01876	0.10369	0.14366	0.15585	0.02508	0.04092	0.07802	0.08102	0.19105
(3 to 4) + (4 to 5)	0.02089	0.03010	0.12950	0.17176	0.18092	0.01814	0.02344	0.02929	0.05168	0.13252
(4 to 5) + (5 to 6)	0.03820	0.04816	0.13563	0.14208	0.14305	0.03080	0.03639	0.05674	0.09395	0.10477
(5 to 6) + (6 to 7)	0.03193	0.03327	0.07353	0.09028	0.09096	0.03264	0.03539	0.06770	0.09401	0.09562
(6 to 7) + (7 to 8)	0.01790	0.01888	0.04130	0.05988	0.06337	0.02097	0.02420	0.05439	0.06991	0.07111
(7 to 8) + (8 to 9)	0.01501	0.04065	0.06442	0.07040	0.07783	0.02882	0.07315	0.08766	0.09212	0.09353
(8 to 9) + (9 to 10)	0.04505	0.29032	0.30578	0.31657	0.32217	0.05213	0.19863	0.25340	0.25383	0.25433
(9 to 10) + (10 to 11)	0.07973	0.54337	0.56609	0.57702	0.57812	0.08023	0.32803	0.42427	0.42435	0.42491
(10 to 11) + (11 to 12)	0.05647	0.30013	0.34126	0.34499	0.34650	0.06405	0.21034	0.27273	0.27276	0.28938
(11 to 12) + (12 to 13)	0.02972	0.03038	0.06233	0.08075	0.08235	0.03180	0.03426	0.06378	0.06812	0.09961
(12 to 13) + (13 to 14)	0.04645	0.05573	0.05631	0.07527	0.08293	0.05306	0.06204	0.07065	0.08407	0.09961
(13 to 14) + (14 to 15)	0.03581	0.05185	0.06625	0.10041	0.11569	0.03633	0.04625	0.05052	0.08756	0.08780
(14 to 15) + (15 to 16)	0.01156	0.02092	0.07709	0.11710	0.13749	0.01275	0.02288	0.04444	0.07271	0.08752
(15 to 16) + (16 to 17)	0.00492	0.00785	0.06756	0.08593	0.11323	0.01353	0.02692	0.05888	0.06671	0.10499

Euclidean Distances Boxes	V_5					V_6				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.02133	0.03116	0.05318	0.05321	0.05522	0.02532	0.03727	0.05715	0.05716	0.07504
(2 to 3) + (3 to 4)	0.02695	0.03773	0.06159	0.06615	0.07100	0.03199	0.04503	0.06683	0.07023	0.14948
(3 to 4) + (4 to 5)	0.02048	0.02483	0.02674	0.09671	0.10375	0.02431	0.02944	0.03136	0.09476	0.16081
(4 to 5) + (5 to 6)	0.03630	0.04143	0.05350	0.15953	0.18633	0.04322	0.04899	0.06051	0.16332	0.16810
(5 to 6) + (6 to 7)	0.03940	0.04130	0.06668	0.13438	0.17087	0.04706	0.04907	0.07282	0.14297	0.14415
(6 to 7) + (7 to 8)	0.02694	0.02799	0.05208	0.12231	0.13785	0.03297	0.03490	0.05602	0.12517	0.13168
(7 to 8) + (8 to 9)	0.06166	0.10556	0.12246	0.16698	0.24746	0.07622	0.13317	0.15319	0.19603	0.20168
(8 to 9) + (9 to 10)	0.08478	0.16636	0.19855	0.23261	0.46168	0.08668	0.16817	0.19352	0.20263	0.20984
(9 to 10) + (10 to 11)	0.11369	0.23137	0.28606	0.33314	0.51577	0.10396	0.20804	0.24604	0.26149	0.27398
(10 to 11) + (11 to 12)	0.10118	0.18173	0.22666	0.25827	0.29300	0.10708	0.18785	0.22445	0.24697	0.30224
(11 to 12) + (12 to 13)	0.04197	0.04369	0.06486	0.08230	0.09091	0.05187	0.05500	0.07237	0.08774	0.21141
(12 to 13) + (13 to 14)	0.06465	0.07375	0.07876	0.07926	0.08702	0.07691	0.08732	0.09187	0.09257	0.16865
(13 to 14) + (14 to 15)	0.04226	0.05110	0.05118	0.05143	0.05294	0.05012	0.06027	0.06029	0.06077	0.06298
(14 to 15) + (15 to 16)	0.01543	0.02562	0.05492	0.05630	0.06592	0.01836	0.03075	0.05837	0.05872	0.06822
(15 to 16) + (16 to 17)	0.01543	0.02561	0.05491	0.05628	0.06590	0.01836	0.03074	0.05835	0.05870	0.06819

Euclidean Distances Boxes	V_7					V_8				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.03256	0.04717	0.07666	0.07947	0.16449	0.02742	0.03976	0.06252	0.07819	0.08880
(2 to 3) + (3 to 4)	0.03969	0.05396	0.07268	0.10287	0.18127	0.03942	0.05340	0.07228	0.10678	0.10802
(3 to 4) + (4 to 5)	0.02498	0.03031	0.03249	0.08758	0.18398	0.03000	0.03734	0.04736	0.06733	0.11533
(4 to 5) + (5 to 6)	0.03870	0.04282	0.05861	0.11215	0.16352	0.03909	0.04278	0.06173	0.07286	0.13657
(5 to 6) + (6 to 7)	0.04596	0.04803	0.07963	0.12180	0.13574	0.04771	0.04923	0.08048	0.10456	0.13438
(6 to 7) + (7 to 8)	0.03168	0.03402	0.06223	0.10861	0.13138	0.03505	0.03800	0.06226	0.07636	0.13143
(7 to 8) + (8 to 9)	0.08010	0.15232	0.17914	0.21055	0.22395	0.08067	0.14805	0.17835	0.17851	0.22122
(8 to 9) + (9 to 10)	0.08353	0.17365	0.20085	0.20472	0.20715	0.07750	0.15507	0.18354	0.18441	0.18902
(9 to 10) + (10 to 11)	0.09309	0.19828	0.23561	0.24222	0.24562	0.08121	0.16633	0.20285	0.20413	0.21211
(10 to 11) + (11 to 12)	0.10339	0.19334	0.23668	0.26183	0.26794	0.09947	0.17757	0.22170	0.22222	0.24262
(11 to 12) + (12 to 13)	0.04550	0.05026	0.07385	0.10670	0.13664	0.04982	0.05582	0.07865	0.07898	0.10646
(12 to 13) + (13 to 14)	0.07798	0.08904	0.09561	0.10846	0.13330	0.07586	0.08492	0.09283	0.10435	0.11637
(13 to 14) + (14 to 15)	0.06448	0.07782	0.07801	0.08059	0.08822	0.06473	0.07767	0.08691	0.12785	0.13869
(14 to 15) + (15 to 16)	0.02179	0.03408	0.06615	0.06841	0.07611	0.02288	0.03567	0.07468	0.12859	0.13943
(15 to 16) + (16 to 17)	0.01520	0.02518	0.05710	0.05795	0.05806	0.01406	0.02353	0.05505	0.17649	0.17650

Euclidean Distances Boxes	V_9					V_{10}				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.02029	0.03003	0.04303	0.05607	0.14179	0.02058	0.03145	0.04129	0.05786	0.15134
(2 to 3) + (3 to 4)	0.05063	0.06655	0.07607	0.08295	0.16896	0.05129	0.06864	0.07525	0.08401	0.17868
(3 to 4) + (4 to 5)	0.04445	0.05647	0.06196	0.07061	0.14502	0.04510	0.05812	0.06170	0.07226	0.15321
(4 to 5) + (5 to 6)	0.03644	0.03930	0.05889	0.08390	0.13342	0.03708	0.03982	0.05577	0.08753	0.13996
(5 to 6) + (6 to 7)	0.04396	0.04588	0.07318	0.11525	0.12268	0.04517	0.04731	0.06982	0.12101	0.12897
(6 to 7) + (7 to 8)	0.03876	0.04416	0.07352	0.11998	0.12106	0.04133	0.04990	0.07106	0.12789	0.12908
(7 to 8) + (8 to 9)	0.09035	0.16079	0.21332	0.23818	0.23926	0.08382	0.15566	0.20051	0.23355	0.23472
(8 to 9) + (9 to 10)	0.07331	0.14814	0.18874	0.19254	0.19304	0.06276	0.13438	0.17180	0.17787	0.17832
(9 to 10) + (10 to 11)	0.07187	0.14834	0.19652	0.20238	0.20292	0.05868	0.12849	0.17060	0.17958	0.18008
(10 to 11) + (11 to 12)	0.09810	0.17671	0.22799	0.25609	0.25680	0.08947	0.16777	0.20937	0.24527	0.24605
(11 to 12) + (12 to 13)	0.05695	0.06793	0.09035	0.13095	0.14873	0.06049	0.07589	0.09140	0.13969	0.15966
(12 to 13) + (13 to 14)	0.07021	0.07489	0.09695	0.12595	0.15075	0.07176	0.07622	0.09557	0.13006	0.15821
(13 to 14) + (14 to 15)	0.07097	0.08482	0.10888	0.15287	0.16109	0.07165	0.08623	0.10615	0.15914	0.16876
(14 to 15) + (15 to 16)	0.04278	0.06364	0.10813	0.14157	0.14476	0.04337	0.06681	0.10289	0.14278	0.14701
(15 to 16) + (16 to 17)	0.01819	0.02988	0.06192	0.06329	0.06593	0.01848	0.03173	0.05792	0.05963	0.06313

B.2.3.2 LPP on edge difference values in aggregate

Euclidean Distances sd1	V(Edge1)					V(Edge2)				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.35578	0.37140	0.44911	0.47420	0.52121	0.50778	0.57711	0.58034	0.85648	0.92372
(58 to 59) + (59 to 60)	0.58896	0.60011	0.61844	0.77838	0.83074	0.32656	0.34637	0.34872	0.62739	0.79273
(59 to 60) + (60 to 61)	0.88915	0.89823	0.92438	1.09829	1.11855	0.10121	0.10170	0.10393	0.16808	0.34069
(60 to 61) + (61 to 62)	0.72669	0.72768	0.80341	0.84209	1.18802	0.26136	0.26171	0.26982	0.28058	0.40978
(61 to 62) + (62 to 63)	0.63057	0.64206	0.72811	1.05228	1.68079	2.49349	2.50987	2.56646	2.60446	2.75323
(62 to 63) + (63 to 64)	0.32998	0.35334	0.39028	0.75939	1.24525	2.28123	2.29760	2.34787	2.37703	2.42038
(63 to 64) + (64 to 65)	1.31426	2.13736	2.25398	2.39959	2.66962	2.05472	2.08822	2.60827	2.61473	2.63217
(64 to 65) + (65 to 66)	2.68215	4.31886	4.52855	4.62514	4.71116	6.61151	6.69155	7.31363	7.32058	7.32171
(65 to 66) + (66 to 67)	1.75485	2.58176	2.73811	2.91158	2.98146	4.75955	4.81797	4.92172	5.06782	5.06786
(66 to 67) + (67 to 68)	1.01500	1.01799	1.13605	1.33153	1.52338	0.19424	0.20646	0.20728	0.37408	0.45663
(67 to 68) + (68 to 69)	0.98767	0.99543	1.05436	1.15905	1.45596	0.23745	0.26637	0.26705	0.38336	0.71388
(68 to 69) + (69 to 70)	0.75684	0.77982	0.84278	0.91465	1.10924	0.51312	0.60492	0.60912	0.84253	1.09611
(69 to 70) + (70 to 71)	0.55408	0.57849	0.68066	0.73951	0.77183	0.35373	0.43010	0.43544	0.65254	0.91633
(70 to 71) + (71 to 72)	0.16034	0.17275	0.24339	0.35408	0.49114	0.04646	0.05966	0.06089	0.14049	0.40383
(71 to 72) + (72 to 73)	0.05542	0.06041	0.08922	0.14566	0.28750	0.00141	0.00194	0.00198	0.00485	0.01146
(72 to 73) + (73 to 74)	0.00126	0.00160	0.00291	0.00763	0.02328	0.00083	0.00131	0.00134	0.00356	0.00854

Euclidean Distances sd1	**V(Edge3)**					**V(*comp*)**				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(57 to 58) + (58 to 59)	0.10139	0.15253	0.27713	0.28108	0.44262	0.00210	0.16177	0.47452	0.66242	0.66370
(58 to 59) + (59 to 60)	0.17316	0.24506	0.32512	0.41569	0.57769	0.00269	0.17747	0.40480	0.63935	0.64084
(59 to 60) + (60 to 61)	0.23993	0.32622	0.45956	0.55202	0.64587	0.00278	0.16284	0.30380	0.68261	0.68447
(60 to 61) + (61 to 62)	0.16746	0.20113	0.34891	0.52482	0.69454	0.00648	0.26840	0.30468	0.72017	0.72251
(61 to 62) + (62 to 63)	0.21899	0.23409	0.31485	0.90393	1.26552	0.06317	1.73649	2.30324	2.78381	3.35991
(62 to 63) + (63 to 64)	0.32938	0.33951	0.38869	1.41423	1.83472	0.07676	1.61586	2.21968	2.69351	6.63714
(63 to 64) + (64 to 65)	2.65664	2.91661	2.96583	3.57713	3.71910	3.83487	4.65173	4.79686	5.01684	8.75281
(64 to 65) + (65 to 66)	4.92977	5.44564	5.46686	5.47246	5.47371	7.67425	8.53624	8.66407	8.72248	9.08996
(65 to 66) + (66 to 67)	2.50966	2.77690	2.83510	2.90225	2.97381	3.86065	4.07623	4.10759	4.44941	4.45038
(66 to 67) + (67 to 68)	0.23009	0.30123	0.44221	0.52809	0.61106	0.00461	0.24394	0.39309	0.99887	1.00156
(67 to 68) + (68 to 69)	0.28157	0.39064	0.48777	0.60500	0.83426	0.00363	0.23964	0.54647	0.94926	0.95230
(68 to 69) + (69 to 70)	0.18294	0.27427	0.45500	0.57206	0.79523	0.00258	0.20764	0.63297	0.97374	0.97586
(69 to 70) + (70 to 71)	0.10796	0.17143	0.44139	0.48781	0.63852	0.00152	0.14030	0.50163	0.94048	0.94265
(70 to 71) + (71 to 72)	0.02614	0.04463	0.13709	0.16067	0.30578	0.00029	0.03399	0.13675	0.31913	0.32004
(71 to 72) + (72 to 73)	0.00013	0.00028	0.00102	0.00121	0.00291	0.00000	0.00097	0.00471	0.01242	0.01245
(72 to 73) + (73 to 74)	0.00021	0.00054	0.00199	0.00227	0.00541	0.00000	0.00077	0.00424	0.01123	0.01126

Euclidean Distances Boxes	**V(Edge1)**					**V(Edge2)**				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.00470	0.37372	0.38240	0.59681	0.72723	0.00299	0.00383	0.09012	0.16777	0.22222
(2 to 3) + (3 to 4)	0.00649	0.43660	0.44659	0.68323	0.81495	0.00621	0.00771	0.13411	0.18315	0.26817
(3 to 4) + (4 to 5)	0.01293	0.61975	0.63312	0.78605	0.95759	0.00525	0.00650	0.10369	0.12458	0.18415
(4 to 5) + (5 to 6)	0.00883	0.35610	0.36345	0.36374	0.53423	0.00945	0.01139	0.11413	0.15077	0.20174
(5 to 6) + (6 to 7)	0.01849	0.57884	0.58997	0.65344	0.65731	0.01256	0.01501	0.12304	0.20470	0.22956
(6 to 7) + (7 to 8)	0.02792	0.68447	0.69674	0.79477	0.94494	0.01177	0.01381	0.05728	0.19450	0.22169
(7 to 8) + (8 to 9)	2.21406	2.40747	2.78523	2.83359	2.98394	3.54888	4.02833	4.06000	4.14955	4.17608
(8 to 9) + (9 to 10)	2.70268	5.91808	6.86506	6.87863	6.88239	7.90474	9.35733	9.36734	9.37436	9.37438
(9 to 10) + (10 to 11)	3.19243	7.63887	8.27744	8.28666	8.29344	12.2323	13.2967	13.3010	13.3098	13.3149
(10 to 11) + (11 to 12)	2.71242	4.16631	4.23549	4.29029	4.50994	7.91585	8.01315	8.05439	8.30998	8.46365
(11 to 12) + (12 to 13)	0.04282	0.96365	0.98059	1.12062	1.34714	0.05250	0.06088	0.15233	0.41721	0.60245
(12 to 13) + (13 to 14)	0.02783	0.96699	0.98621	1.16761	1.26840	0.01554	0.01889	0.23072	0.25398	0.29146
(13 to 14) + (14 to 15)	0.00658	0.35742	0.36542	0.55042	0.75448	0.01185	0.01466	0.23259	0.28088	0.28350
(14 to 15) + (15 to 16)	0.00355	0.23520	0.24058	0.37697	0.71685	0.00179	0.00238	0.07582	0.14918	0.21222
(15 to 16) + (16 to 17)	0.00019	0.03051	0.03126	0.06967	0.29278	0.00005	0.00009	0.01310	0.04339	0.10456

Euclidean Distances Boxes	$\mathbf{V(Edge3)}$					$\mathbf{V}(\boldsymbol{comp})$				
	1-D	2-D	3-D	4-D	5-D	1-D	2-D	3-D	4-D	5-D
(1 to 2) + (2 to 3)	0.00190	0.00275	0.12277	0.18634	0.23140	0.00007	0.00008	0.16932	0.23014	0.27228
(2 to 3) + (3 to 4)	0.00281	0.00394	0.14536	0.19607	0.22838	0.00014	0.00016	0.24130	0.30274	0.34820
(3 to 4) + (4 to 5)	0.00230	0.00314	0.08976	0.09588	0.15766	0.00017	0.00020	0.25193	0.29956	0.37501
(4 to 5) + (5 to 6)	0.00487	0.00649	0.13372	0.13883	0.19424	0.00026	0.00031	0.28083	0.28743	0.39419
(5 to 6) + (6 to 7)	0.00792	0.01041	0.16400	0.20464	0.26857	0.00053	0.00063	0.42962	0.44595	0.59261
(6 to 7) + (7 to 8)	0.01266	0.01627	0.14656	0.29663	0.42320	0.00106	0.00126	0.53690	0.71623	0.82048
(7 to 8) + (8 to 9)	3.36817	4.10029	4.17783	4.29997	4.36684	4.59784	5.47620	5.87578	6.05345	6.08847
(8 to 9) + (9 to 10)	5.04443	8.58092	8.59966	8.60810	8.60812	6.84760	11.1895	11.3010	11.3157	11.3172
(9 to 10) + (10 to 11)	6.70889	9.84413	9.85865	9.87260	9.87876	9.09606	12.9292	13.0575	13.1043	13.1208
(10 to 11) + (11 to 12)	5.05462	5.39095	5.51048	5.75107	5.90471	6.84846	7.21853	7.98583	8.43635	8.66447
(11 to 12) + (12 to 13)	0.03442	0.04346	0.22969	0.47195	0.66005	0.00316	0.00377	0.88964	1.29758	1.61786
(12 to 13) + (13 to 14)	0.00952	0.01273	0.27896	0.30778	0.35051	0.00054	0.00065	0.56946	0.59562	0.75483
(13 to 14) + (14 to 15)	0.00582	0.00791	0.21604	0.23552	0.23810	0.00031	0.00038	0.47093	0.55573	0.71043
(14 to 15) + (15 to 16)	0.00077	0.00125	0.08469	0.14409	0.20576	0.00007	0.00009	0.21079	0.32078	0.56876
(15 to 16) + (16 to 17)	0.00038	0.00068	0.06117	0.11434	0.17562	0.00001	0.00001	0.06221	0.10934	0.25321

B.3 Generalization of LPP on difference in pixels across an edge

B.3.1 Generalization of LPP across different edges in the same focusing sequence

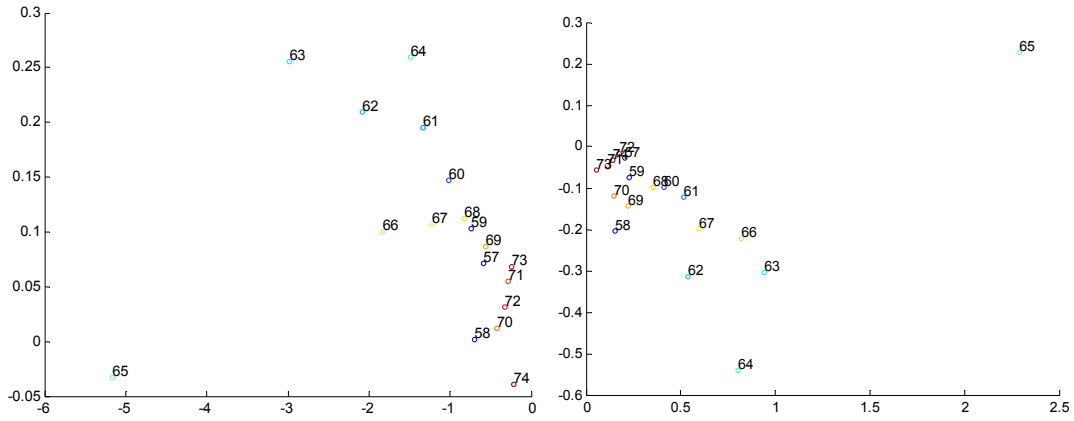


Figure B.20 - Two-dimensional representations of testing sets V(Edge3) of sd1 (left plot) and V(Edge1) of sd1 (right plot) using the respective training sets V(Edge2) of sd1 and V(Edge3) of sd1. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

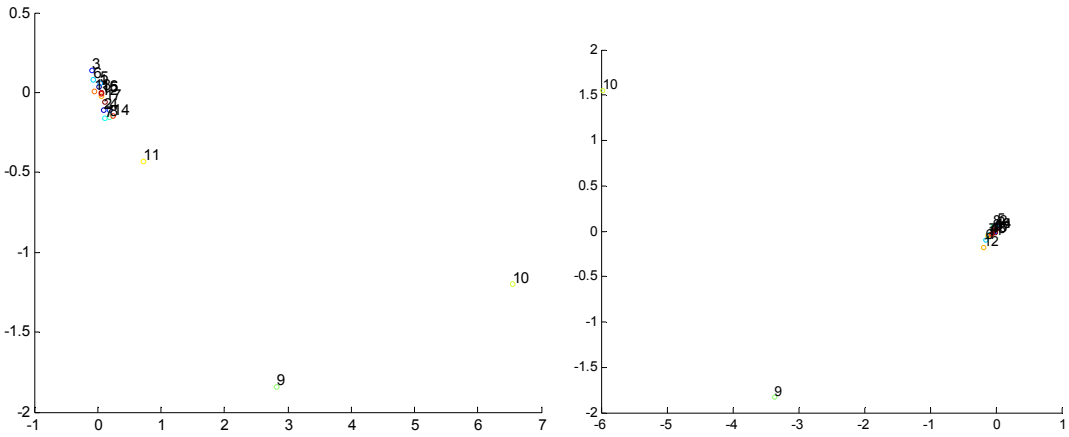


Figure B.21 - Two-dimensional representations of testing sets V(Edge1)of Boxes (left plot) and V(Edge2)of Boxes (right plot) using the respective training sets V(Edge2)of Boxes and V(Edge3) of Boxes. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

B.3.2 Generalization of LPP across different focusing sequences

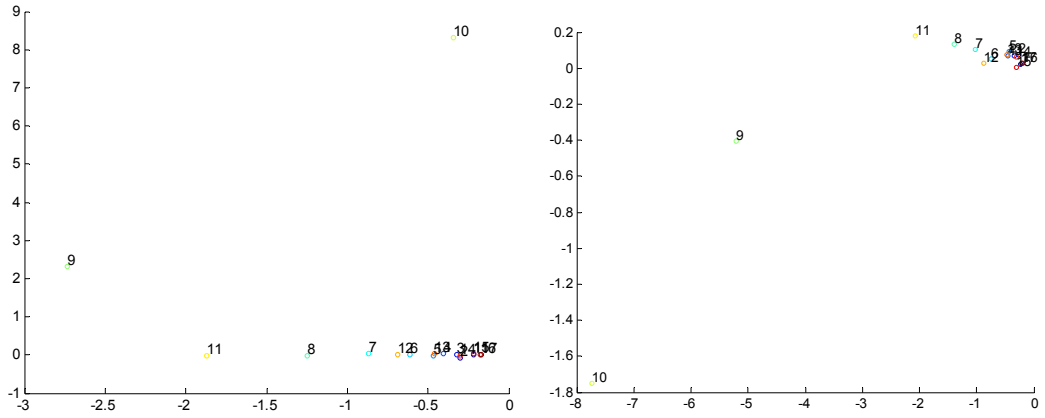


Figure B.22 - Two-dimensional representations of testing sets V(Edge2)of Boxes using the training sets V(Edge1) of sd1(left plot) and V(Edge2)of sd1(right plot). The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

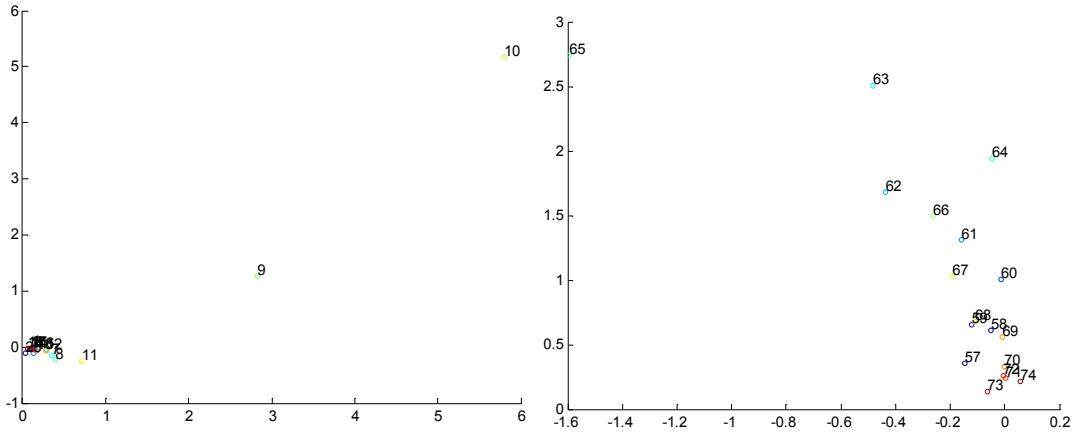


Figure B.23 - Two-dimensional representations of testing sets V(Edge2)of Boxes (left plot) and V(Edge3)of sd1 (right plot) using the respective training sets V(Edge3) of sd1 and V(Edge1)of Boxes. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

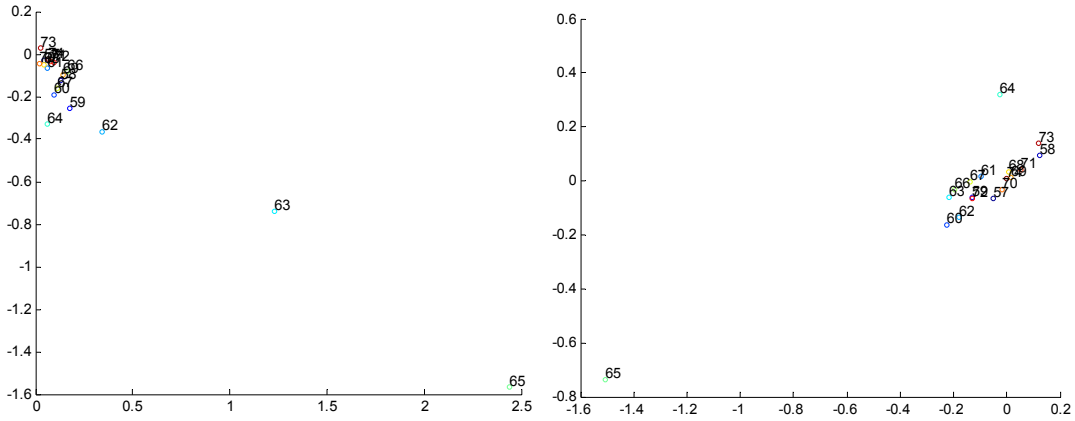


Figure B.24 - Two-dimensional representations of testing sets V(Edge2)of sd1 (left plot) and V(Edge1)of sd1 (right plot) using the respective training sets V(Edge2)of Boxes and V(Edge3)of Boxes. The x-axis represents the most significant dimension of LPP whereas the y-axis represents the second dimension.

Appendix C: Compact Disc Content

The compact disc attached is organised in two folders containing the MATLAB code and pre-processed datasets used throughout this study. The *Programs* folder contains MATLAB code whereas the *Pre-processed datasets* folder contains datasets in MATLAB format. To run programs the user should add to path the folder containing programs as in the figure below.

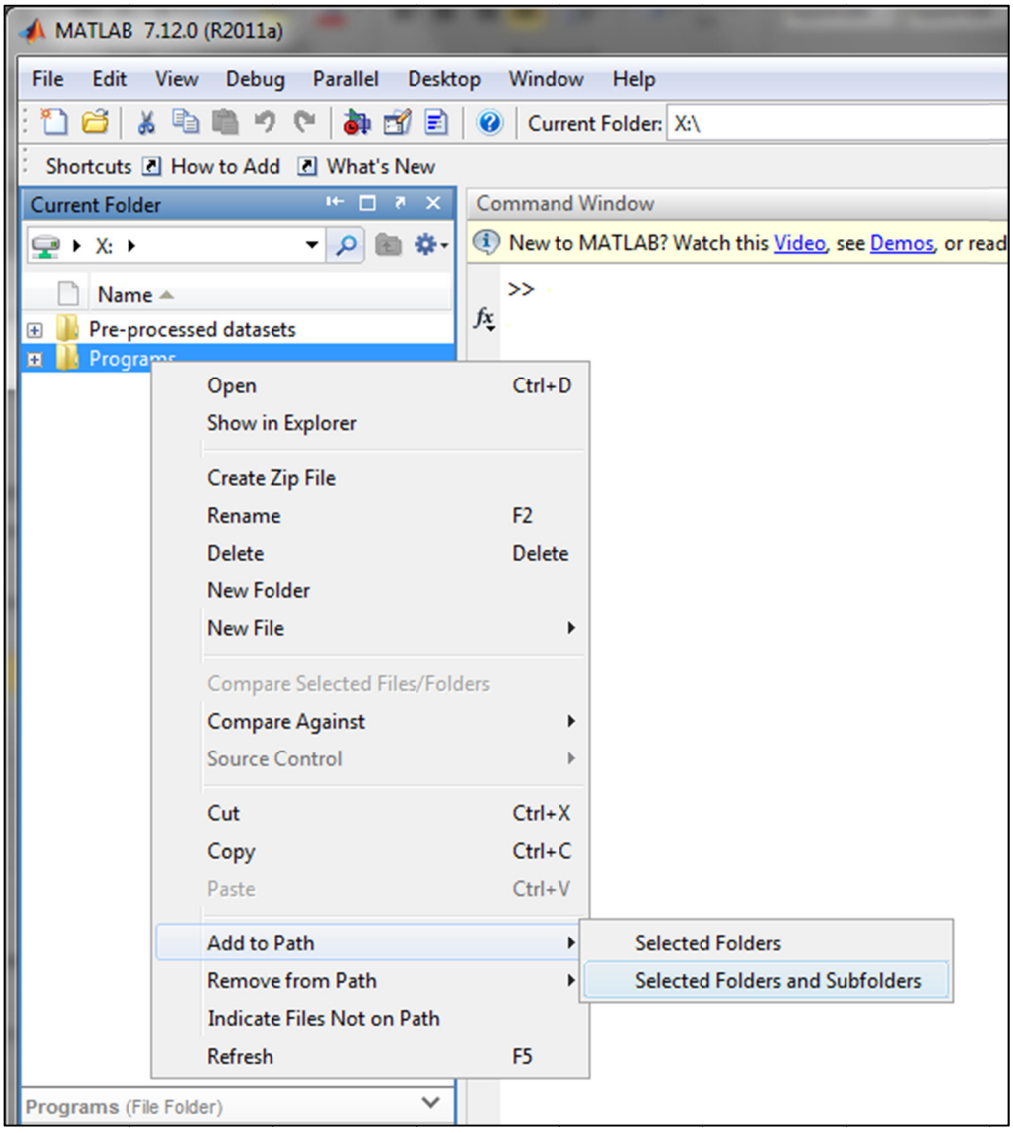


Figure B.24 – Adding to path the *Programs* folder.

The *Programs* folder is comprises five folders. The folder *Algorithms* contains the implementation of the algorithms described in section 2.2.2. These comprise *LPP, Focused_Image_Detection* and *Edge_Detection_and_Selection*.

The code in *LPP* subfolder contains an implementation of Locality Preserving Projections algorithm (He & Niyogi, 2003) - LPP.m, LGE.m, EuDist2.m, and constructW.m. These four programs were written by Cai (2011) and hence were downloaded from:

http://www.zjucadcg.cn/dengcai/Data/DimensionReduction.html

(The url has changed from the time they were downloaded – the new url is http://www.cad.zju.edu.cn/home/dengcai/Data/DimensionReduction.html)

The code myLPP.m calls LPP.m and contains the most common options for LPP according to experiments. Such options include setting LPP to use *k nearest neighbours* and *heat kernel*. The user only requires inputting the data, number of low dimensions, and *k* and *t* parameters. The program listing is also provided in Appendix D.1.

The folder *Focused_Image_Detection* contains the implementation (focImg.m) of the algorithm described in section 2.2.2.3. The program listing is also provided in Appendix D.2.

Another folder *Edge_Detection_and_Selection* comprises AutoFocus.m and getEdges.m. getEdges.m is called from AutoFocus.m. AutoFocus.m includes a whole procedure from an image dataset to the detection of the focused image as described in section 2.2.2.2. Program listing is also available in Appendix D.3

The other four folders in *Programs* are:

- *LPP_on_Images* which contains ajdLPP.m. This was used to obtain results in section 3.1.1where several approaches of LPP were concerned. phyadj.m and

phyoride.m are related with the construction of physical adjacency affinity matrix and are called from adjLPP.m. Program listing is provided in Appendix D.4

- *RMSE* contains RMSE2.m used in section 3.2.1 and edims.m used in section 3.2.2. Program listing is provided in Appendix D.5
- *Edges* folder contains pixDifference.m to extract edge difference values and hence prepare datasets in section 3.3. The other program, sepData.m takes a dataset with in aggregate difference values as input and divides such dataset into independent datasets $V_1, V_2, \ldots$ up to V_{10}. This was used to prepare data for experiment in section 3.3.1 . Program listing is provided in Appendix D.6
- *Pre-Processing* which contains normalizing to unit Norm (unitNorm.m) and scaling values to [0,1] (scaledValues.m). Program listing is provided in Appendix D.7

The datasets folder is organised in three folders.

Image_datasets contains:

- UnitNorm_Image_datasets which was used for obtaining the results in sections 3.1.1and 3.2. The datasets in this file are normalized to unit norm.
- ScaledValues_Image_datasets which was used for obtaining results in section 3.1.2. The datasets in this file are scaled to [0,1].

Edge_datasets contains:

- Boxes_Indep and sd1_Indep in which datasets are scaled to [0,1] and were used for obtaining results in section 3.3.1.

- Edges contain datasets relating to edge datasets. Such datasets were also scaled to [0,1]. These datasets were used to obtain the results in sections 3.3.2 and 3.4.

Finally HighResolution in High_Res_datasets folder contains datasets used to test the algorithm in section 2.2.2.2 and hence obtain the results in section 3.3.4.

Appendix D: MATLAB Code Listing

D.1 LPP

D.1.1 LPP.m

```
function [eigvector, eigvalue, elapse] = LPP(W, options, data)
% LPP: Locality Preserving Projections
%
%       [eigvector, eigvalue] = LPP(W, options, data)
%
%             Input:
%               data       - Data matrix. Each row vector of fea is a data
point.
%               W       - Affinity matrix. You can either call "constructW"
%                         to construct the W, or construct it by yourself.
%               options - Struct value in Matlab. The fields in options
%                         that can be set:
%
%                         Please see LGE.m for other options.
%
%             Output:
%               eigvector - Each column is an embedding function, for a new
%                           data point (row vector) x,  y = x*eigvector
%                           will be the embedding result of x.
%               eigvalue  - The sorted eigvalue of LPP eigen-problem.
%               elapse    - Time spent on different steps
%
%
%    Examples:
%
%       fea = rand(50,70);
%       options = [];
%       options.Metric = 'Euclidean';
%       options.NeighborMode = 'KNN';
%       options.k = 5;
%       options.WeightMode = 'HeatKernel';
%       options.t = 5;
%       W = constructW(fea,options);
%       options.PCARatio = 0.99
%       [eigvector, eigvalue] = LPP(W, options, fea);
%       Y = fea*eigvector;
%
%
%       fea = rand(50,70);
%       gnd = [ones(10,1);ones(15,1)*2;ones(10,1)*3;ones(15,1)*4];
%       options = [];
%       options.Metric = 'Euclidean';
%       options.NeighborMode = 'Supervised';
%       options.gnd = gnd;
%       options.bLDA = 1;
%       W = constructW(fea,options);
%       options.PCARatio = 1;
%       [eigvector, eigvalue] = LPP(W, options, fea);
%       Y = fea*eigvector;
%
```

```
%
% Note: After applying some simple algebra, the smallest eigenvalue problem:
%               data^T*L*data = \lemda data^T*D*data
%      is equivalent to the largest eigenvalue problem:
%               data^T*W*data = \beta data^T*D*data
%       where L=D-W;  \lemda= 1 - \beta.
% Thus, the smallest eigenvalue problem can be transformed to a largest
% eigenvalue problem. Such tricks are adopted in this code for the
% consideration of calculation precision of Matlab.
%
%
% See also constructW, LGE
%
%Reference:
%   Xiaofei He, and Partha Niyogi, "Locality Preserving Projections"
%   Advances in Neural Information Processing Systems 16 (NIPS 2003),
%   Vancouver, Canada, 2003.
%
%   Xiaofei He, Shuicheng Yan, Yuxiao Hu, Partha Niyogi, and Hong-Jiang
%   Zhang, "Face Recognition Using Laplacianfaces", IEEE PAMI, Vol. 27, No.
%   3, Mar. 2005.
%
%   Deng Cai, Xiaofei He and Jiawei Han, "Document Clustering Using
%   Locality Preserving Indexing" IEEE TKDE, Dec. 2005.
%
%   Deng Cai, Xiaofei He and Jiawei Han, "Using Graph Model for Face
Analysis",
%   Technical Report, UIUCDCS-R-2005-2636, UIUC, Sept. 2005
%
%   Xiaofei He, "Locality Preserving Projections"
%   PhD's thesis, Computer Science Department, The University of Chicago,
%   2005.
%
%   version 2.1 --June/2007
%   version 2.0 --May/2007
%   version 1.1 --Feb/2006
%   version 1.0 --April/2004
%
%   Written by Deng Cai (dengcai2 AT cs.uiuc.edu)
%


bGlobal = 0;
if ~exist('data','var')
    bGlobal = 1;
    global data;
end

if (~exist('options','var'))
   options = [];
end

[nSmp,nFea] = size(data);
if size(W,1) ~= nSmp
    error('W and data mismatch!');
end
```

```
%==========================
% If data is too large, the following centering codes can be commented
% options.keepMean = 1;
%==========================
if isfield(options,'keepMean') & options.keepMean
    ;
else
    if issparse(data)
        data = full(data);
    end
    sampleMean = mean(data);
    data = (data - repmat(sampleMean,nSmp,1));
end
%==========================


D = full(sum(W,2));

 if ~isfield(options,'Regu') | ~options.Regu
    DToPowerHalf = D.^.5;
    D_mhalf = DToPowerHalf.^-1;

    if nSmp < 5000
        tmpD_mhalf = repmat(D_mhalf,1,nSmp);
        W = (tmpD_mhalf.*W).*tmpD_mhalf';
        clear tmpD_mhalf;
    else
        [i_idx,j_idx,v_idx] = find(W);
        v1_idx = zeros(size(v_idx));
        for i=1:length(v_idx)
            v1_idx(i) = v_idx(i)*D_mhalf(i_idx(i))*D_mhalf(j_idx(i));
        end
        W = sparse(i_idx,j_idx,v1_idx);
        clear i_idx j_idx v_idx v1_idx
    end
    W = max(W,W');

    data = repmat(DToPowerHalf,1,nFea).*data;
    [eigvector, eigvalue, elapse] = LGE(W, [], options, data);
else
    options.ReguAlpha = options.ReguAlpha*sum(D)/length(D);

    D = sparse(1:nSmp,1:nSmp,D,nSmp,nSmp);

    if bGlobal & isfield(options,'keepMean') & options.keepMean
        [eigvector, eigvalue, elapse] = LGE(W, D, options);
    else
        [eigvector, eigvalue, elapse] = LGE(W, D, options, data);
    end
end


eigIdx = find(eigvalue < 1e-3);
eigvalue (eigIdx) = [];
eigvector(:,eigIdx) = [];
```

D.1.2 LGE.m

```
function [eigvector, eigvalue, elapse] = LGE(W, D, options, data)
% LGE: Linear Graph Embedding
%
%       [eigvector, eigvalue] = LGE(W, D, options, data)
%
%             Input:
%               data       - data matrix. Each row vector of data is a
%                            sample vector.
%               W       - Affinity graph matrix.
%               D       - Constraint graph matrix.
%                            LGE solves the optimization problem of
%                            a* = argmax (a'data'WXa)/(a'data'DXa)
%                            Default: D = I
%
%               options - Struct value in Matlab. The fields in options
%                            that can be set:
%
%                     ReducedDim   -  The dimensionality of the reduced
%                                     subspace. If 0, all the dimensions
%                                     will be kept. Default is 30.
%
%                           Regu  -  1: regularized solution,
%                                       a* = argmax
(a'X'WXa)/(a'X'DXa+ReguAlpha*I)
%                                    0: solve the sinularity problem by SVD
(PCA)
%                                    Default: 0
%
%                       ReguAlpha -  The regularization parameter. Valid
%                                    when Regu==1. Default value is 0.1.
%
%                           ReguType  -  'Ridge': Tikhonov regularization
%                                        'Custom': User provided
%                                                  regularization matrix
%                                         Default: 'Ridge'
%                       regularizerR  -   (nFea x nFea) regularization
%                                         matrix which should be provided
%                                         if ReguType is 'Custom'. nFea is
%                                         the feature number of data
%                                         matrix
%
%                           PCARatio     -  The percentage of principal
%                                           component kept in the PCA
%                                           step. The percentage is
%                                           calculated based on the
%                                           eigenvalue. Default is 1
%                                           (100%, all the non-zero
%                                           eigenvalues will be kept.
%                                           If PCARatio > 1, the PCA step
%                                           will keep exactly PCARatio
principle
%                                           components (does not exceed the
```

```
%                                           exact number of non-zero
components).
%
%
%             Output:
%               eigvector - Each column is an embedding function, for a new
%                           sample vector (row vector) x,  y = x*eigvector
%                           will be the embedding result of x.
%               eigvalue  - The sorted eigvalue of the eigen-problem.
%               elapse    - Time spent on different steps
%
%
%
%    Examples:
%
% See also LPP, NPE, IsoProjection, LSDA.
%
%
%Reference:
%
%   Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han, and Thomas Huang,
%   "Learning a Spatially Smooth Subspace for Face Recognition", CVPR'2007
%
%   version 2.1 --June/2007
%   version 2.0 --May/2007
%   version 1.0 --Sep/2006
%
%   Written by Deng Cai (dengcai2 AT cs.uiuc.edu)
%

if ~exist('data','var')
    global data;
end


if (~exist('options','var'))
   options = [];
end

if isfield(options,'ReducedDim')
    Dim = options.ReducedDim;
else
    Dim = 30;
end

if ~isfield(options,'Regu') | ~options.Regu
    bPCA = 1;
    if ~isfield(options,'PCARatio')
        options.PCARatio = 1;
    end
else
    bPCA = 0;
    if ~isfield(options,'ReguType')
        options.ReguType = 'Ridge';
    end
    if ~isfield(options,'ReguAlpha')
        options.ReguAlpha = 0.1;
    end
end
```

```
bD = 1;
if ~exist('D','var') | isempty(D)
    bD = 0;
end


[nSmp,nFea] = size(data);
if size(W,1) ~= nSmp
    error('W and data mismatch!');
end
if bD & (size(D,1) ~= nSmp)
    error('D and data mismatch!');
end

tmp_T = cputime;

bChol = 0;
if bPCA & (nSmp > nFea) & (options.PCARatio >= 1)
    if bD
        DPrime = data'*D*data;
    else
        DPrime = data'*data;
    end
    if issparse(DPrime)
        DPrime = full(DPrime);
    end
    DPrime = max(DPrime,DPrime');
    [R,p] = chol(DPrime);

    if p == 0
        bPCA = 0;
        bChol = 1;
    end
end

%======================================
% SVD
%======================================

if bPCA
    if nSmp > nFea
        ddata = data'*data;
        if issparse(ddata)
            ddata = full(ddata);
        end
        ddata = max(ddata,ddata');

        [eigvector_PCA, eigvalue_PCA] = eig(ddata);
        eigvalue_PCA = diag(eigvalue_PCA);
        clear ddata;

        maxEigValue = max(abs(eigvalue_PCA));
        eigIdx = find(eigvalue_PCA/maxEigValue < 1e-12);
        eigvalue_PCA(eigIdx) = [];
        eigvector_PCA(:,eigIdx) = [];

        [junk, index] = sort(-eigvalue_PCA);
        eigvalue_PCA = eigvalue_PCA(index);
```

```
    eigvector_PCA = eigvector_PCA(:, index);

    %======================================
    if options.PCARatio > 1
        idx = options.PCARatio;
        if idx < length(eigvalue_PCA)
            eigvalue_PCA = eigvalue_PCA(1:idx);
            eigvector_PCA = eigvector_PCA(:,1:idx);
        end
    elseif options.PCARatio < 1
        sumEig = sum(eigvalue_PCA);
        sumEig = sumEig*options.PCARatio;
        sumNow = 0;
        for idx = 1:length(eigvalue_PCA)
            sumNow = sumNow + eigvalue_PCA(idx);
            if sumNow >= sumEig
                break;
            end
        end
        eigvalue_PCA = eigvalue_PCA(1:idx);
        eigvector_PCA = eigvector_PCA(:,1:idx);
    end
    %======================================

    if bD
        data = data*eigvector_PCA;
    else
        eigvalue_PCA = eigvalue_PCA.^-.5;
        data = (data*eigvector_PCA).*repmat(eigvalue_PCA',nSmp,1);
    end
else
    ddata = data*data';
    if issparse(ddata)
        ddata = full(ddata);
    end
    ddata = max(ddata,ddata');

    [eigvector, eigvalue_PCA] = eig(ddata);
    eigvalue_PCA = diag(eigvalue_PCA);
    clear ddata;

    maxEigValue = max(eigvalue_PCA);
    eigIdx = find(eigvalue_PCA/maxEigValue < 1e-12);
    eigvalue_PCA(eigIdx) = [];
    eigvector(:,eigIdx) = [];

    [junk, index] = sort(-eigvalue_PCA);
    eigvalue_PCA = eigvalue_PCA(index);
    eigvector = eigvector(:, index);

    %======================================
    if options.PCARatio > 1
        idx = options.PCARatio;
        if idx < length(eigvalue_PCA)
            eigvalue_PCA = eigvalue_PCA(1:idx);
            eigvector = eigvector(:,1:idx);
        end
    elseif options.PCARatio < 1
        sumEig = sum(eigvalue_PCA);
```

```
            sumEig = sumEig*options.PCARatio;
            sumNow = 0;
            for idx = 1:length(eigvalue_PCA)
                sumNow = sumNow + eigvalue_PCA(idx);
                if sumNow >= sumEig
                    break;
                end
            end
            eigvalue_PCA = eigvalue_PCA(1:idx);
            eigvector = eigvector(:,1:idx);
        end
        %=========================================

        eigvalue_PCA = eigvalue_PCA.^.5;
        eigvalue_PCAMinus = eigvalue_PCA.^-1;

        eigvector_PCA =
(data'*eigvector).*repmat(eigvalue_PCAMinus',nFea,1);

        if bD
            data = eigvector.*repmat(eigvalue_PCA',nSmp,1);
        else
            data = eigvector;
        end

        eigvalue_PCA = eigvalue_PCAMinus;


        clear eigvector;
    end

    if bD
        DPrime = data'*D*data;
        DPrime = max(DPrime,DPrime');
    end
else
    if ~bChol
        if bD
            DPrime = data'*D*data;
        else
            DPrime = data'*data;
        end

        switch lower(options.ReguType)
            case {lower('Ridge')}
                for i=1:size(DPrime,1)
                    DPrime(i,i) = DPrime(i,i) + options.ReguAlpha;
                end
            case {lower('Tensor')}
                DPrime = DPrime + options.ReguAlpha*options.regularizerR;
            case {lower('Custom')}
                DPrime = DPrime + options.ReguAlpha*options.regularizerR;
            otherwise
                error('ReguType does not exist!');
        end

        DPrime = max(DPrime,DPrime');
    end
end
```

```
WPrime = data'*W*data;
WPrime = max(WPrime,WPrime');


elapse.timePCA = cputime - tmp_T;

tmp_T = cputime;


%=====================================
% Generalized Eigen
%=====================================

dimMatrix = size(WPrime,2);

if Dim > dimMatrix
    Dim = dimMatrix;
end


if isfield(options,'bEigs')
    if options.bEigs
        bEigs = 1;
    else
        bEigs = 0;
    end
else
    if (dimMatrix > 1000 & Dim < dimMatrix/10) | (dimMatrix > 500 & Dim <
dimMatrix/20) | (dimMatrix > 250 & Dim < dimMatrix/30)
        bEigs = 1;
    else
        bEigs = 0;
    end
end


if bEigs
    %disp('use eigs to speed up!');
    option = struct('disp',0);
    if bPCA & ~bD
        [eigvector, eigvalue] = eigs(WPrime,Dim,'la',option);
    else
        if bChol
            option.cholB = 1;
            [eigvector, eigvalue] = eigs(WPrime,R,Dim,'la',option);
        else
            [eigvector, eigvalue] = eigs(WPrime,DPrime,Dim,'la',option);
        end
    end
    eigvalue = diag(eigvalue);
else
    if bPCA & ~bD
        [eigvector, eigvalue] = eig(WPrime);
    else
        [eigvector, eigvalue] = eig(WPrime,DPrime);
    end
    eigvalue = diag(eigvalue);
```

```
    [junk, index] = sort(-eigvalue);
    eigvalue = eigvalue(index);
    eigvector = eigvector(:,index);

    if Dim < size(eigvector,2)
        eigvector = eigvector(:, 1:Dim);
        eigvalue = eigvalue(1:Dim);
    end
end



if bPCA
    if bD
        eigvector = eigvector_PCA*eigvector;
    else
        eigvector =
eigvector_PCA*(repmat(eigvalue_PCA,1,length(eigvalue)).*eigvector);
    end
end

for i = 1:size(eigvector,2)
    eigvector(:,i) = eigvector(:,i)./norm(eigvector(:,i));
end


elapse.timeMethod = cputime - tmp_T;
elapse.timeAll = elapse.timePCA + elapse.timeMethod;
```

D.1.3 EuDist2.m

```
function D = EuDist2(fea_a,fea_b,bSqrt)
% Euclidean Distance matrix
%   D = EuDist(fea_a,fea_b)
%   fea_a:    nSample_a * nFeature
%   fea_b:    nSample_b * nFeature
%   D:      nSample_a * nSample_a
%       or  nSample_a * nSample_b


if ~exist('bSqrt','var')
    bSqrt = 1;
end


if (~exist('fea_b','var')) | isempty(fea_b)
    [nSmp, nFea] = size(fea_a);

    aa = sum(fea_a.*fea_a,2);
    ab = fea_a*fea_a';

    aa = full(aa);
    ab = full(ab);

    if bSqrt
```

```
        D = sqrt(repmat(aa, 1, nSmp) + repmat(aa', nSmp, 1) - 2*ab);
        D = real(D);
    else
        D = repmat(aa, 1, nSmp) + repmat(aa', nSmp, 1) - 2*ab;
    end

    D = max(D,D');
    D = D - diag(diag(D));
    D = abs(D);
else
    [nSmp_a, nFea] = size(fea_a);
    [nSmp_b, nFea] = size(fea_b);

    aa = sum(fea_a.*fea_a,2);
    bb = sum(fea_b.*fea_b,2);
    ab = fea_a*fea_b';

    aa = full(aa);
    bb = full(bb);
    ab = full(ab);

    if bSqrt
        D = sqrt(repmat(aa, 1, nSmp_b) + repmat(bb', nSmp_a, 1) - 2*ab);
        D = real(D);
    else
        D = repmat(aa, 1, nSmp_b) + repmat(bb', nSmp_a, 1) - 2*ab;
    end

    D = abs(D);
end
```

D.1.4 constructW.m

```
function [W, elapse] = constructW(fea,options)

%   Usage:
%   W = constructW(fea,options)
%
%   fea: Rows of vectors of data points. Each row is x_i
%   options: Struct value in Matlab. The fields in options that can be set:
%           Metric -  Choices are:
%               'Euclidean' - Will use the Euclidean distance of two data
%                             points to evaluate the "closeness" between
%                             them. [Default One]
%               'Cosine'    - Will use the cosine value of two vectors
%                             to evaluate the "closeness" between them.
%                             A popular similarity measure used in
%                             Information Retrieval.
%
%           NeighborMode -  Indicates how to construct the graph. Choices
%                           are: [Default 'KNN']
%                'KNN'            -  k = 0
%                                       Complete graph
%                                    k > 0
%                                      Put an edge between two nodes if and
%                                      only if they are among the k nearst
```

```
%                                       neighbors of each other. You are
%                                       required to provide the parameter k
in
%                                       the options. Default k=5.
%               'Supervised'      -  k = 0
%                                       Put an edge between two nodes if and
%                                       only if they belong to same class.
%                                    k > 0
%                                       Put an edge between two nodes if
%                                       they belong to same class and they
%                                       are among the k nearst neighbors of
%                                       each other.
%                                    Default: k=0
%                                   You are required to provide the label
%                                   information gnd in the options.
%
%           WeightMode   -  Indicates how to assign weights for each edge
%                           in the graph. Choices are:
%               'Binary'       - 0-1 weighting. Every edge receiveds weight
%                                of 1. [Default One]
%               'HeatKernel'   - If nodes i and j are connected, put weight
%                                W_ij = exp(-norm(x_i - x_j)/2t^2). This
%                                weight mode can only be used under
%                                'Euclidean' metric and you are required to
%                                provide the parameter t.
%               'Cosine'       - If nodes i and j are connected, put weight
%                                cosine(x_i,x_j). Can only be used under
%                                'Cosine' metric.
%
%            k          -   The parameter needed under 'KNN' NeighborMode.
%                           Default will be 5.
%            gnd        -   The parameter needed under 'Supervised'
%                           NeighborMode.  Colunm vector of the label
%                           information for each data point.
%            bLDA       -   0 or 1. Only effective under 'Supervised'
%                           NeighborMode. If 1, the graph will be constructed
%                           to make LPP exactly same as LDA. Default will be
%                           0.
%            t          -   The parameter needed under 'HeatKernel'
%                           WeightMode. Default will be 1
%         bNormalized   -   0 or 1. Only effective under 'Cosine' metric.
%                           Indicates whether the fea are already be
%                           normalized to 1. Default will be 0
%       bSelfConnected  -   0 or 1. Indicates whether W(i,i) == 1. Default 1
%                           if 'Supervised' NeighborMode & bLDA == 1,
%                           bSelfConnected will always be 1. Default 1.
%
%
%    Examples:
%
%       fea = rand(50,15);
%       options = [];
%       options.Metric = 'Euclidean';
%       options.NeighborMode = 'KNN';
%       options.k = 5;
%       options.WeightMode = 'HeatKernel';
%       options.t = 1;
%       W = constructW(fea,options);
%
```

```
%
%       fea = rand(50,15);
%       gnd = [ones(10,1);ones(15,1)*2;ones(10,1)*3;ones(15,1)*4];
%       options = [];
%       options.Metric = 'Euclidean';
%       options.NeighborMode = 'Supervised';
%       options.gnd = gnd;
%       options.WeightMode = 'HeatKernel';
%       options.t = 1;
%       W = constructW(fea,options);
%
%
%       fea = rand(50,15);
%       gnd = [ones(10,1);ones(15,1)*2;ones(10,1)*3;ones(15,1)*4];
%       options = [];
%       options.Metric = 'Euclidean';
%       options.NeighborMode = 'Supervised';
%       options.gnd = gnd;
%       options.bLDA = 1;
%       W = constructW(fea,options);
%
%
%    For more details about the different ways to construct the W, please
%    refer:
%       Deng Cai, Xiaofei He and Jiawei Han, "Document Clustering Using
%       Locality Preserving Indexing" IEEE TKDE, Dec. 2005.
%
%
%    Written by Deng Cai (dengcai2 AT cs.uiuc.edu), April/2004, Feb/2006,
%                                                 May/2007
%

if (~exist('options','var'))
   options = [];
else
   if ~isstruct(options)
       error('parameter error!');
   end
end

%=================================================
if ~isfield(options,'Metric')
    options.Metric = 'Cosine';
end

switch lower(options.Metric)
    case {lower('Euclidean')}
    case {lower('Cosine')}
        if ~isfield(options,'bNormalized')
            options.bNormalized = 0;
        end
    otherwise
        error('Metric does not exist!');
end

%=================================================
if ~isfield(options,'NeighborMode')
    options.NeighborMode = 'KNN';
end
```

```
switch lower(options.NeighborMode)
    case {lower('KNN')}  %For simplicity, we include the data point itself
in the kNN
        if ~isfield(options,'k')
            options.k = 5;
        end
    case {lower('Supervised')}
        if ~isfield(options,'bLDA')
            options.bLDA = 0;
        end
        if options.bLDA
            options.bSelfConnected = 1;
        end
        if ~isfield(options,'k')
            options.k = 0;
        end
        if ~isfield(options,'gnd')
            error('Label(gnd) should be provided under ''Supervised''
NeighborMode!');
        end
        if ~isempty(fea) && length(options.gnd) ~= size(fea,1)
            error('gnd doesn''t match with fea!');
        end
    otherwise
        error('NeighborMode does not exist!');
end

%=================================================

if ~isfield(options,'WeightMode')
    options.WeightMode = 'Binary';
end

bBinary = 0;
switch lower(options.WeightMode)
    case {lower('Binary')}
        bBinary = 1;
    case {lower('HeatKernel')}
        if ~strcmpi(options.Metric,'Euclidean')
            warning('''HeatKernel'' WeightMode should be used under
''Euclidean'' Metric!');
            options.Metric = 'Euclidean';
        end
        if ~isfield(options,'t')
            options.t = 1;
        end
    case {lower('Cosine')}
        if ~strcmpi(options.Metric,'Cosine')
            warning('''Cosine'' WeightMode should be used under ''Cosine''
Metric!');
            options.Metric = 'Cosine';
        end
        if ~isfield(options,'bNormalized')
            options.bNormalized = 0;
        end
    otherwise
        error('WeightMode does not exist!');
end
```

```
%=================================================

if ~isfield(options,'bSelfConnected')
    options.bSelfConnected = 1;
end

%=================================================
tmp_T = cputime;

if isfield(options,'gnd')
    nSmp = length(options.gnd);
else
    nSmp = size(fea,1);
end
maxM = 62500000; %500M
BlockSize = floor(maxM/(nSmp*3));


if strcmpi(options.NeighborMode,'Supervised')
    Label = unique(options.gnd);
    nLabel = length(Label);
    if options.bLDA
        G = zeros(nSmp,nSmp);
        for idx=1:nLabel
            classIdx = options.gnd==Label(idx);
            G(classIdx,classIdx) = 1/sum(classIdx);
        end
        W = sparse(G);
        elapse = cputime - tmp_T;
        return;
    end

    switch lower(options.WeightMode)
        case {lower('Binary')}
            if options.k > 0
                G = zeros(nSmp*(options.k+1),3);
                idNow = 0;
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
                    D = EuDist2(fea(classIdx,:),[],0);
                    [dump idx] = sort(D,2); % sort each row
                    clear D dump;
                    idx = idx(:,1:options.k+1);

                    nSmpClass = length(classIdx)*(options.k+1);
                    G(idNow+1:nSmpClass+idNow,1) =
repmat(classIdx,[options.k+1,1]);
                    G(idNow+1:nSmpClass+idNow,2) = classIdx(idx(:));
                    G(idNow+1:nSmpClass+idNow,3) = 1;
                    idNow = idNow+nSmpClass;
                    clear idx
                end
                G = sparse(G(:,1),G(:,2),G(:,3),nSmp,nSmp);
                G = max(G,G');
            else
                G = zeros(nSmp,nSmp);
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
```

```
                    G(classIdx,classIdx) = 1;
                end
            end

            if ~options.bSelfConnected
                for i=1:size(G,1)
                    G(i,i) = 0;
                end
            end

            W = sparse(G);
        case {lower('HeatKernel')}
            if options.k > 0
                G = zeros(nSmp*(options.k+1),3);
                idNow = 0;
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
                    D = EuDist2(fea(classIdx,:),[],0);
                    [dump idx] = sort(D,2); % sort each row
                    clear D;
                    idx = idx(:,1:options.k+1);
                    dump = dump(:,1:options.k+1);
                    dump = exp(-dump/(2*options.t^2));

                    nSmpClass = length(classIdx)*(options.k+1);
                    G(idNow+1:nSmpClass+idNow,1) =
repmat(classIdx,[options.k+1,1]);
                    G(idNow+1:nSmpClass+idNow,2) = classIdx(idx(:));
                    G(idNow+1:nSmpClass+idNow,3) = dump(:);
                    idNow = idNow+nSmpClass;
                    clear dump idx
                end
                G = sparse(G(:,1),G(:,2),G(:,3),nSmp,nSmp);
            else
                G = zeros(nSmp,nSmp);
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
                    D = EuDist2(fea(classIdx,:),[],0);
                    D = exp(-D/(2*options.t^2));
                    G(classIdx,classIdx) = D;
                end
            end

            if ~options.bSelfConnected
                for i=1:size(G,1)
                    G(i,i) = 0;
                end
            end

            W = sparse(max(G,G'));
        case {lower('Cosine')}
            if ~options.bNormalized
                [nSmp, nFea] = size(fea);
                if issparse(fea)
                    fea2 = fea';
                    feaNorm = sum(fea2.^2,1).^.5;
                    for i = 1:nSmp
                        fea2(:,i) = fea2(:,i) ./ max(1e-10,feaNorm(i));
                    end
```

```
                    fea = fea2';
                    clear fea2;
                else
                    feaNorm = sum(fea.^2,2).^.5;
                    for i = 1:nSmp
                        fea(i,:) = fea(i,:) ./ max(1e-12,feaNorm(i));
                    end
                end

            end

            if options.k > 0
                G = zeros(nSmp*(options.k+1),3);
                idNow = 0;
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
                    D = fea(classIdx,:)*fea(classIdx,:)';
                    [dump idx] = sort(-D,2); % sort each row
                    clear D;
                    idx = idx(:,1:options.k+1);
                    dump = -dump(:,1:options.k+1);

                    nSmpClass = length(classIdx)*(options.k+1);
                    G(idNow+1:nSmpClass+idNow,1) =
repmat(classIdx,[options.k+1,1]);
                    G(idNow+1:nSmpClass+idNow,2) = classIdx(idx(:));
                    G(idNow+1:nSmpClass+idNow,3) = dump(:);
                    idNow = idNow+nSmpClass;
                    clear dump idx
                end
                G = sparse(G(:,1),G(:,2),G(:,3),nSmp,nSmp);
            else
                G = zeros(nSmp,nSmp);
                for i=1:nLabel
                    classIdx = find(options.gnd==Label(i));
                    G(classIdx,classIdx) = fea(classIdx,:)*fea(classIdx,:)';
                end
            end

            if ~options.bSelfConnected
                for i=1:size(G,1)
                    G(i,i) = 0;
                end
            end

            W = sparse(max(G,G'));
        otherwise
            error('WeightMode does not exist!');
    end
    elapse = cputime - tmp_T;
    return;
end


if strcmpi(options.NeighborMode,'KNN') && (options.k > 0)
    if strcmpi(options.Metric,'Euclidean')
        G = zeros(nSmp*(options.k+1),3);
        for i = 1:ceil(nSmp/BlockSize)
            if i == ceil(nSmp/BlockSize)
```

```
                smpIdx = (i-1)*BlockSize+1:nSmp;
                dist = EuDist2(fea(smpIdx,:),fea,0);
                dist = full(dist);
                [dump idx] = sort(dist,2); % sort each row
                idx = idx(:,1:options.k+1);
                dump = dump(:,1:options.k+1);
                if ~bBinary
                    dump = exp(-dump/(2*options.t^2));
                end

                G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),1) =
repmat(smpIdx',[options.k+1,1]);
                G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),2) =
idx(:);
                if ~bBinary
                    G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),3)
= dump(:);
                else
                    G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),3)
= 1;
                end
            else
                smpIdx = (i-1)*BlockSize+1:i*BlockSize;
                dist = EuDist2(fea(smpIdx,:),fea,0);
                dist = full(dist);
                [dump idx] = sort(dist,2); % sort each row
                idx = idx(:,1:options.k+1);
                dump = dump(:,1:options.k+1);
                if ~bBinary
                    dump = exp(-dump/(2*options.t^2));
                end

                G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),1) =
repmat(smpIdx',[options.k+1,1]);
                G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),2) = idx(:);
                if ~bBinary
                    G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),3) = dump(:);
                else
                    G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),3) = 1;
                end
            end
        end

        W = sparse(G(:,1),G(:,2),G(:,3),nSmp,nSmp);
    else
        if ~options.bNormalized
            [nSmp, nFea] = size(fea);
            if issparse(fea)
                fea2 = fea';
                clear fea;
                for i = 1:nSmp
                    fea2(:,i) = fea2(:,i) ./ max(1e-
10,sum(fea2(:,i).^2,1).^.5);
                end
                fea = fea2';
```

```
                clear fea2;
            else
                feaNorm = sum(fea.^2,2).^.5;
                for i = 1:nSmp
                    fea(i,:) = fea(i,:) ./ max(1e-12,feaNorm(i));
                end
            end
        end

        G = zeros(nSmp*(options.k+1),3);
        for i = 1:ceil(nSmp/BlockSize)
            if i == ceil(nSmp/BlockSize)
                smpIdx = (i-1)*BlockSize+1:nSmp;
                dist = fea(smpIdx,:)*fea';
                dist = full(dist);
                [dump idx] = sort(-dist,2); % sort each row
                idx = idx(:,1:options.k+1);
                dump = -dump(:,1:options.k+1);

                G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),1) =
repmat(smpIdx',[options.k+1,1]);
                G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),2) =
idx(:);
                G((i-1)*BlockSize*(options.k+1)+1:nSmp*(options.k+1),3) =
dump(:);
            else
                smpIdx = (i-1)*BlockSize+1:i*BlockSize;
                dist = fea(smpIdx,:)*fea';
                dist = full(dist);
                [dump idx] = sort(-dist,2); % sort each row
                idx = idx(:,1:options.k+1);
                dump = -dump(:,1:options.k+1);

                G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),1) =
repmat(smpIdx',[options.k+1,1]);
                G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),2) = idx(:);
                G((i-
1)*BlockSize*(options.k+1)+1:i*BlockSize*(options.k+1),3) = dump(:);
            end
        end

        W = sparse(G(:,1),G(:,2),G(:,3),nSmp,nSmp);
    end

    if strcmpi(options.WeightMode,'Binary')
        W(find(W)) = 1;
    end

    if isfield(options,'bSemiSupervised') && options.bSemiSupervised
        tmpgnd = options.gnd(options.semiSplit);

        Label = unique(tmpgnd);
        nLabel = length(Label);
        G = zeros(sum(options.semiSplit),sum(options.semiSplit));
        for idx=1:nLabel
            classIdx = tmpgnd==Label(idx);
            G(classIdx,classIdx) = 1;
```

```
        end
        Wsup = sparse(G);
        if ~isfield(options,'SameCategoryWeight')
            options.SameCategoryWeight = 1;
        end
        W(options.semiSplit,options.semiSplit) =
(Wsup>0)*options.SameCategoryWeight;
    end

    if ~options.bSelfConnected
        for i=1:size(W,1)
            W(i,i) = 0;
        end
    end

    W = max(W,W');

    elapse = cputime - tmp_T;
    return;
end


% strcmpi(options.NeighborMode,'KNN') & (options.k == 0)
% Complete Graph

if strcmpi(options.Metric,'Euclidean')
    W = EuDist2(fea,[],0);
    W = exp(-W/(2*options.t^2));
else
    if ~options.bNormalized
%         feaNorm = sum(fea.^2,2).^.5;
%         fea = fea ./ repmat(max(1e-10,feaNorm),1,size(fea,2));
        [nSmp, nFea] = size(fea);
        if issparse(fea)
            fea2 = fea';
            feaNorm = sum(fea2.^2,1).^.5;
            for i = 1:nSmp
                fea2(:,i) = fea2(:,i) ./ max(1e-10,feaNorm(i));
            end
            fea = fea2';
            clear fea2;
        else
            feaNorm = sum(fea.^2,2).^.5;
            for i = 1:nSmp
                fea(i,:) = fea(i,:) ./ max(1e-12,feaNorm(i));
            end
        end
    end

    W = full(fea*fea');
end

if ~options.bSelfConnected
    for i=1:size(W,1)
        W(i,i) = 0;
    end
end

W = max(W,W');
```

```
elapse = cputime - tmp_T;
```

D.1.5 myLPP.m

```
function [Y,eigvalue,eigenvector] = myLPP(data,d,K,T)
%   myLPP.m     This function sets options for applying LPP. It
%               then uses Deng Cai's LPP.m to compute LPP
%   Author: Richard Muscat
%
%       Input:
%           data  - Data matrix. Each row vector of data is a data point.
%
%           d     - The dimensionality of the reduced
%                   subspace. If 0, all the dimensions
%                   will be kept. Default is 30.
%
%           K     - The parameter for 'KNN' NeighborMode.
%           T     - The parameter needed under 'HeatKernel'
%                   WeightMode. Default will be 1
%
%        Output:
%           eigvector - Each column is an embedding function, for a new
%                       data point (row vector) x,  y = x*eigvector
%                       will be the embedding result of x.
%           eigvalue  - The sorted eigvalue of LPP eigen-problem.
%           Y         - The low-dimensional output of LPP

    options = [];
    options.Metric = 'Euclidean';
    options.NeighborMode = 'KNN';
    options.k = K;
    options.WeightMode = 'HeatKernel';
    options.t = T;
    options.ReducedDim = d;
    W = constructW(data,options);
    [eigenvector,eigvalue] = LPP(W,options,data);
    Y = data*eigenvector;
End
```

D.2 Implementation of Algorithm for detection of focused image

D.2.1 focImg.m

```
function [ focusedImage,cid ] = focImg(Y)
% focImg.m - Automates detection of focused image
%   Author: Richard Muscat
%   Input:
%       Y - Output of LPP (Should have a minimum of 3 dimensions)
%           Each row vector of pixdiff is a data point.
%
%   Ouput:
```

```
%       focusedImage -  The index of the focused image in Y
%       cid          -  A matrix showing a set of data-points
%                       in which a change in direction was
%                       detected and their respective
%                       3-dimensional Manhattan distance

[r,c] = size(Y);
prevdir1 = 2;
prevdir2 = 2;
cidr = 1;
    for i=2:r
        if (Y(i,1)-Y(i-1,1))<0
            direction1 = -1;
        else
            direction1 = 1;
        end

        if (Y(i,2)-Y(i-1,2))<0
            direction2 = -1;
        else
            direction2 = 1;
        end

        if((direction1 == -prevdir1)||(direction2 == -prevdir2))
            cid(cidr,1) = i-1;
            cid(cidr,2) = abs(Y(i,1)-Y(i-1,1))+abs(Y(i-1,1)-Y(i-
2,1))+abs(Y(i,2)-Y(i-1,2))+abs(Y(i-1,2)-Y(i-2,2))+abs(Y(i,3)-Y(i-
1,3))+abs(Y(i-1,3)-Y(i-2,3));
            cidr=cidr+1;
        end
        prevdir1 = direction1;
        prevdir2 = direction2;
    end
    [mag,index] = max(cid(:,2)) ;
    focusedImage = cid(index,1);
end
```

D.3 Automating edge detection and selection

D.3.1 getEdges.m

```
function [ info ] = getEdges( bw )
% getEdges.m Extracts the location and length of vertical edges
%   Author: Richard Muscat
%   Input:
%       bw - binary image
%
%   Output:
%       info - matrix containing location and length
%              for each vertical edge.
%

[r,c] = size(bw);
rec = 1;
    for i = 1:c;
        length = 0;
```

```
        reco = 0;
        for j = 1:r;
            if(bw(j,i)==1)
                if(length ==0)
                    reco = 1;
                    info(rec,1) = j;
                    info(rec,2) = i;
                    length = length+1;
                else
                    length = length+1;
                end
            else
                if(reco==1)
                    info(rec,3) = length;
                    reco = 0;
                    length =0;
                    rec = rec+1;
                end
            end
        end
        if(reco==1)
            info(rec,3) = length;
            reco = 0;
            length =0;
            rec = rec+1;
        end

end
```

D.3.2 AutoFocus.m

```
function [foc,ed,cid,YY] = AutoFocus( data,m,n,B )
% AutoFocus.m
%   Author: Richard Muscat
%   Input:
%       data  - Data matrix. Each row vector of data is a data
point.
%       m,n   - size of images
%       B     - number of pixels across edge
%   Output:
%       foc     -   The index of the focused image in Y
%       cid     -   A matrix showing a set of data-points
%                   in which a change in direction was
%                   detected and their respective
%                   3-dimensional Manhattan distance
%       ed      -   The location and length of the selected edge
%       YY      -   Output of LPP. Each row vector of YY
%                   is a low dimensional data point
%
%
%
    mdata = mean(data);                 % compute mean image
    mimg = reshape(mdata,m,n);          % shape pixels to size
```

```
    [BW,t] =  edge(mimg,'sobel');
    BW = edge(mimg,'sobel',(2*t));      % apply sobel
    eds = getEdges(BW);                 % get vertical edges
    [mag,index]=max(eds(:,3));          % select longest edge
    ed = eds(index,:);
    XX = pixDifference(data,m,n,ed(1,1),ed(1,2),ed(1,3),B);
    nXX = scaleValues(XX);
    [YY,eigval,eigvec] = myLPP(nXX,5,5,5);
    [foc,cid] = focImg(YY);
end
```

D.4 LPP on images

D.4.1 adjLPP.m

```
function
[Ahk,Ehk,Yhk,Ahp,Ehp,Yhp,Ahc,Ehc,Yhc,Abk,Ebk,Ybk,Abp,Ebp,Ybp,Abc,Ebc
,Ybc] = adjLPP(data,d,K,T)

% adjLPP.m - This code was written to study several approaches for
LPP on
%             images. These include:
%
%                    Adjacency graph - KNN/physical adjacency/the
%                    combination of the two
%
%                    Weight Matrix - Heat Kernel/Simple-Minded
%
%
%       Input:
%           data  - Data matrix. Each row vector of data is a data
point.
%
%           d     - The dimensionality of the reduced
%                   subspace. If 0, all the dimensions
%                   will be kept. Default is 30.
%
%           K     - The parameter for 'KNN' NeighborMode.
%           T     - The parameter needed under 'HeatKernel'
%                   WeightMode. Default will be 1
%
%       Output: There are the following 3 outputs
%                for each of the 6 approaches
%
%           A**  - transformation matrix A
%           E**  - Eigenvalues
%           Y**  - LPP Low-dimensional output


[m,n] = size(data);
```

```
%Heat Kernel and KNN
options = [];
options.Metric = 'Euclidean';
options.NeighborMode = 'KNN';
options.k = K;
options.WeightMode = 'HeatKernel';
options.t = T;
W1 = constructW(data,options);
options.ReducedDim = d;
[Ahk,Ehk] = LPP(W1,options,data);
Yhk = data*Ahk;

%Heat Kernel and Physical Adjacency
options = [];
options.Metric = 'Euclidean';
options.NeighborMode = 'Supervised';
options.k = 0;
options.WeightMode = 'HeatKernel';
options.t = 5;
options.gnd = ones(m,1)
F = full(constructW(data,options));
phy = phyadj(zeros(size(F))).*F;   %Impose physical adjacency having
                                   %Heat-Kernel values for connected
nodes
W2=sparse(phy);
options.ReducedDim = d;
[Ahp,Ehp] = LPP(W2,options,data);
Yhp = data*Ahp;

%Heat Kernel and KNN + Physical Adjacency
C = phyoride(full(W1),phy);         %combine KNN and physical
adjacency
W3= sparse(C);
options.ReducedDim = d;
[Ahc,Ehc] = LPP(W3,options,data);
Yhc = data*Ahc;

%Simple-minded and KNN
options = [];
options.Metric = 'Euclidean';
options.NeighborMode = 'KNN';
options.k = K;
options.WeightMode = 'Binary';      %Simple-minded approach
options.t = T;
W4 = constructW(data,options);
options.ReducedDim = d;
[Abk,Ebk] = LPP(W4,options,data);
Ybk = data*Abk;

%Simple-minded and Physical Adjacency
```

```
W5 = sparse(phyadj(zeros(size(F)))); %Create physical adjacency
matrix
options.ReducedDim = d;
[Abp,Ebp] = LPP(W5,options,data);
Ybp = data*Abp;

%Simple-minded and KNN + Physical Adjacency
W6 = sparse(phyadj(W4));            %combine KNN and physical
adjacency
options.ReducedDim = d;
[Abc,Ebc] = LPP(W6,options,data);
Ybc = data*Abc;

end
```

D.4.2 phyadj.m

```
function[W] = phyadj(S)
% Inserts physical adjacency
l = size(S);
for i=1:l
    S(i,i)= 1;
    if(i<l)
       S(i,i+1) = 1;
      S(i+1,i) = 1;
   end
end

W=S;
end
```

D.4.3 phyoride.m

```
function[W] = phyoride(S,O)
%Replaces physical adjacency values in matrix S with those of matrix
O
l = size(S);
    for i=1:l
        S(i,i)= O(i,i);
        if(i<l)
            S(i,i+1) = O(i,i+1);
            S(i+1,i) = O(i+1,i);
        end
    end

W=S;
end
```

D.5 RMSE

D.5.1 RMSE2.m

```
function [ R ] = RMSE2( Y,A,X )
%   RMSE2.m -Calculates RMSE for LPP
%   Auhor: Richard Muscat
%
%   Inputs:
%       Y   -   LPP Low-dimensional output
%       A   -   transformation matrix A
%       X   -   Data matrix. Each row vector of data is a data point.
%   Output:
%       R - RMSE

    W = pinv(A);    %Moore-Penrose pseudoinverse of LPP transformation
matrix
    XX = Y*W;       %Reconstruct Data
    D = XX-X;       %Calculate Error
    DD = D.*D;      %Square Error
    DDT = transpose(DD);
    MSE = mean(DDT);
    RMSE = sqrt(MSE);
    R = transpose(RMSE);
end
```

D.5.2 eDims.m

```
function [ R ] = eDims( X,K )
%      Calculate RMSE for a range of dimensions (1 to 30)
%      Author: Richard Muscat
%       Input:
%           X  - Data matrix. Each row vector of data is a data point.
%           T     - The parameter needed under 'HeatKernel'
%                   WeightMode. Default will be 1
%       Output:
%           R - 3D Matrix -> Datapoint x Error x Number of dimensions
    for i=1:30
        [Y,e,A] = myLPP(X,i,K,5);
        R(:,:,i)=RMSE2(Y,A,X);
    end
end
```

D.6 Edges

D.6.1 pixDifference.m

```
function [ pixdiff ] = pixDifference( data,m,n,re,ce,M,B)
```

```
%   pixDifference.m computes difference in pixels across a vertical
edge
%   Author: Richard Muscat
%       Inputs:
%           data   - Data matrix. Each row vector of data is a data
point.
%           m,n    - size of images
%           re     - row location of the edge with respect to top
%           ce     - column location of the edge with respect to
left side
%           M      - number of pixels along the edge
%           B      - number of pixels across edge
%
%       Output:
%           pixdiff - Pixel Difference data matrix.
%                     Each row vector of pixdiff is a data point.


[r,c]=size(data);
    for i = 1:r;
        img = reshape(data(i,:),m,n);   %get image from data matrix
        pA = ce-1;
        pB = ce;
        for ii = 1 : B
            for jj = re:(re+M-1)
                      pixdiff(i,(jj-re+1)+(M *(ii-1)))=
                      abs(img(jj,pA)-img(jj,pB));
            end
            pA=pA-1;
            pB=pB+1;
        end
    end
end
```

D.6.2 sepData.m

```
function [ d1,d2,d3,d4,d5,d6,d7,d8,d9,d10 ] = sepData(pd,r)
%   sepData.m  - generates 10 separate dataset and mean dataset
%   Author: Richard Muscat
%       Inputs:
%          pixdiff - Pixel Difference data matrix.
%                    Each row vector of pixdiff is a data point.
%           r       - length of the edge in pixels
%       Output:
%          10 datasets with independent positions across edge and mean
dataset
%
d1=pd(:,1:r);
d2=pd(:,r+1:2*r);
d3=pd(:,(2*r)+1:3*r);
d4=pd(:,(3*r)+1:4*r);
d5=pd(:,(4*r)+1:5*r);
d6=pd(:,(5*r)+1:6*r);
d7=pd(:,(6*r)+1:7*r);
```

```
d8=pd(:,(7*r)+1:8*r);
d9=pd(:,(8*r)+1:9*r);
d10=pd(:,(9*r)+1:10*r);
end
```

D.7 Pre-Processing

D.7.1 unitNorm.m

```
function [data] = unitNorm(data)
%unitNorm.m - Normalize to unit norm
%   Author: Richard Muscat
[nSmp,nData] = size(data);
    for i = 1:nSmp
        data(i,:) = data(i,:) ./ max(1e-12,norm(data(i,:)));
    end
end
```

D.7.2 scaledValues.m

```
function [maxv] = scaleValues(inp)
%Scale values to 0,1 by dividing all values by the maximum value
%   Author: Richard Muscat
 maxv = inp/(max(max(inp)));

end
```

Appendix E: Project Description Form

Name: Richard Muscat Registration No: 040333821

Title of Project: **Investigating the Auto-Focusing problem using Locality Preserving Projections (LPP): Can LPP be used to determine the best focused image from a sequence of incrementally focused images?**

1. Statement of Objectives

a. What do you intend to achieve?

Investigate whether Locality Preserving Projections (LPP) can assist in the problem of Auto-Focusing.

b. Why have you chosen the proposed project?

The LPP algorithm reduces the data dimensionality while preserving the local data similarity. Since a focusing sequence is a sequence of similar images, LPP may preserve some important information related to the focusing problem.

c. Itemised results/documents you intend to deliver in the achievement of your objectives:

Documented experiments and results on:

- Behaviour of LPP on focusing image sequences.
- Effect of image content on LPP-transformed data
- Behaviour of LPP on difference in pixels across edges
- Any further experiments evolving throughout the investigation.

The report will also include recommendations and conclusions based on the outcome of this investigation

2. The Methods to be used

a. How you intend to achieve the objectives listed above:

- Review literature on auto-focusing, and LPP
- Find an implementation in MATLAB (of LPP algorithm) and apply it on some real data.
- Shoot focusing image sequences
- Perform investigative experiments such as:
    - apply LPP on datasets and experiment with various different data, approach and parameters.
    - Reconstruct image data from reduced data and compute RMSE. Plot RMSE vs. samples for a range of reduced dimensions.
    - Choose an edge for some datasets. Apply LPP on pixel difference across the edge.
- Do any other necessary experimentation according to the outcome of the experiments performed.

b. Why you are intending to do it this way:

- Review literature: to see how the investigated technique differs from approached reported in the literature
- Implementation of LPP: to understand the concept of LPP and be able to work with the algorithm.
- Shoot: to have controlled data sets
- Investigative experiments to analyse:
    - the behaviour of LPP on focusing images sequences.
    - the effect of image content on LPP-transformed data
    - the behaviour of LPP on difference in pixels across edges.

- Further experiments: The outcome of experiments is unknown until such experiment is performed. Hence such outcome determines the next experiment in the course of investigation towards the main objective.

c. Your strategy for getting started

The first step is to review literature on auto focusing and LPP. Understand LPP and be able to apply it as to initiate experimentation. Then the behaviour of this algorithm on focusing image sequences will be studied and subsequently a logical investigation will develop. Any further relevant literature related to the auto focusing problem would also be considered while carrying out such investigation.

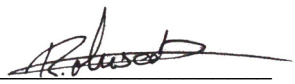
3. The Work Plan

a. A schedule showing key milestones in the project.

- Preliminary research on Auto-Focusing (end of July)
- Review academic research on locality preserving projections algorithm to get to know its concept (August)
- Find an implementation in MATLAB of locality preserving projections algorithm and apply it to some real data thus getting used to working with the algorithm. (beginning of September)
- Acquire data by shooting focusing image sequences (end of September)
- Experiment with LPP on image focusing sequences (October)
- Study the effect of image content on LPP-transformed data (beginning of November).
- Experiment with LPP on pixel difference (end of November/beginning of December)
- Preliminary Project Report (end of December)
- Further investigative experiments according to the outcomes of experiments performed. (January-February)

b. A production schedule for the report (i.e. what date you will start writing and what date it will be finished.).

	Start Writing	Finish Writing
Introduction	1 December	31 December
Literature review	2 January	20 January
Methods	21 January	15 March
Results and Discussion	16 March	15 April
Conclusion	16 April	28 April
Evaluation	29 April	29 April
Print, bind and dispatch	30 April	5 May

Signed  Name:Richard Muscat Date:14 Jan 2012

Bibliography

Cai, D., 2011. *Matlab codes for dimensionality reduction*. [Online] Available at: http://www.zjucadcg.cn/dengcai/Data/DimensionReduction.html [Accessed 3 September 2011].

Ens, J. & Lawrence, P., 1993. An investigation of methods for determining depth from focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(2), pp.97-108.

He, X., 2005. *Locality Preserving Projections*. PhD's Thesis. Computer Science Department, The University of Chicago.

He, X. & Niyogi, P., 2003. Locality Preserving Projections. In *Advances in Neural Information Processing Systems 16*., 2003. MIT Press.

Pentland, A.P., 1987. A New Sense for Depth of Field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4), pp.523-31.

Subbarao, M., 1991. *Spatial-Domain Convolution/Deconvolution Transform*. Technical Report. Computer Vision Laboratory, Dept. of Electrical Engineering, SUNY at Stony Brook.

Subbarao, M., Choi, T.S. & Nikzad, A., 1993. Focusing Techniques. *Journal of Optical Engineering*, pp.2824-36.

Subbarao, M. & Surya, G., 1994. Depth from Defocus: A Spatial Domain Approach. *International Journal of Computer Vision,* 13(3), pp.271-94.

Subbarao, M. & Tyan, J.-K., 1998. Selecting the optimal focus measure for autofocusing and depth-from-focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), pp.864 - 870.

Subbarao, M. & Wei, T.-C., 1992. Depth from Defocus and Rapid Autofocusing: A Practical Approach. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*., 1992.

Xian, T. & Subbarao, M., October 2005. Performance Evaluation of Different Depth From Defocus (DFD) Techniques. In *Proc. of SPIE: Two- and Three- dimensional Methods for Inspection and Metrology III*. Boston, October 2005.

Xiong, Y. & Shafer, S.A., 1993. Depth from Focusing and Defocusing. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. New York, USA, 1993.

Evaluation

This project has given me an opportunity to develop and improve various technical and organisational skills. To accomplish such study, it was necessary to break the study into key parts and to set up a work plan. This was quite challenging at the beginning since the smaller parts were still unclear. However defining a plan at a higher level and updating such plan systematically helped me in being more structured and in improving my time management skills. Since this project was mainly an investigation, it was an entire evolutionary process.

From a technical view point I have learned new concepts mainly relating to dimensionality reduction. The nature of the experiments helped to improve my analytical skills, think critically and develop new ideas. The first attempts had failed to address the problem of auto-focusing as there wasn't a consistent outcome. However through developing ideas and with the help of my knowledgeable supervisor, a successful method to address the problem was soon discovered. From then onwards the investigation evolved progressively and I believe that it still has the potential to evolve further.