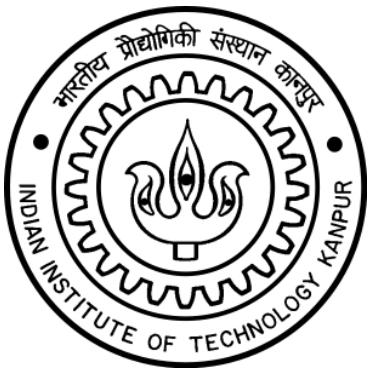
A manifold based clustering algorithm and application to object discovery in RGBD data

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Rahul Erai



to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July 6, 2012

CERTIFICATE

It is certified that the work contained in this thesis entitled "**A manifold based clustering algorithm and application to object discovery in RGBD data**", by **Rahul Erai (Roll No. 10111029)**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Amitabha Mukerjee)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

2 July, 2012

Abstract

Traditional clustering algorithms like *k means clustering* assumes that the data points that are to be to be clustered are distributed as spherical blobs. Hence, they fail when the data to be clustered comes from multiple underlying manifolds. The situation becomes more complex, when the constituting manifolds intersects each other. In this thesis we introduce a new manifold based clustering algorithm, that would cluster these points into their corresponding manifold. The algorithm is inspired from the ideas of gestalt perception and tries to model how we, as humans, perceive manifolds. We also introduce a new distance metric called *Correlation based Earth Mover's Distance*(CEMD), a modified version of the traditional Earth Mover Distance, which is very efficient in data clustering, especially in manifold separation.

As an interesting application of the proposed algorithm, we employ it in discovering object classes from a 3D *pointcloud* dataset containing household objects. With the help of CEMD and some shape and color features extracted from the pointclouds, the manifold clustering algorithm was able to discover object classes present in the dataset. The results of this unsupervised clustering were comparable with the results of applying a supervised *K nearest neighbor* classification on the same dataset.

To my parents...

Acknowledgements

First and foremost, I would like to thank to my thesis supervisor Dr. Amitabha Mukerjee for all his valuable guidance and advice through out the course my thesis. I sincerely appreciate the academic freedom that he gave me in my research. He has been a constant source of inspiration not only in my research, but also for me, as a person.

I would like to thank three amazing persons, Neethi, Ajith,and Varun, who more or less have been my family here for the last two years. With out their support and encouragements, this thesis would not have been possible. I also would like to thank two of my good friends, Ashendra and Chittibabu, for those late night discussions and their valuable suggestions which greatly benefited this thesis. I would also like to express my gratitude to Radu B Rusu at *Willow Garage* for the amazing Point Cloud Library and for personally helping me with some of the technical problems that I faced. I am also grateful to the department of CSE, IIT Kanpur for the the facilities, freedom and the great research environment that I enjoyed. I also would like to thank all my batch mates, for making the last two years memorable.

Last, but certainly not in anyway the least, I sincerely thank my parents and my little brother for their love, support, and care, with out which none of this would have been even remotely possible ever.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

This thesis makes two main contributions. First, it is one of the first works to unsupervised approaches for object classification on image-cum- depth (RGB-D) data made available by integrated rangefinder cameras such as the Kinect (Mic10). But perhaps more general is the second claim, where we develop a novel approach to discovering structures in high-dimensional data based using dynamic neighborhood expansion on the underlying manifold. Since the unsupervised object classification task may be considered an application of the manifold-based clustering approach, we first introduce the manifold-based approach in this presentation.

Unsupervised clustering techniques have always played an important role in statistical analysis of large amount of data. Even though there are a lot of sophisticated algorithms available to do the job, most of them, like the popular *k means clustering*, assumes that points data would be spread as a spherical blob in the space in which they are portrayed. This is an acceptable assumption if one does not have any prior information about the way in which data is distributed in the space. But often in practical situations, data points tends follow some specific structure than the generic spherical distribution causing algorithms like k means clustering to fail spectacularly. Even though the explicit dimensionality of the data space could be huge, these geometric structures tends to have a comparatively smaller intrinsic dimension. These low dimensional topological constructs are often called as manifolds. Hence, in this thesis, we propose a powerful manifold based spatial clustering algorithm that would be able to separate and cluster points from a multi-manifold embedded space.Our studies presented in this thesis can be conceptuality divided into two phases. In the first phase,

we discuss the new spatial clustering algorithm we introduced. In the second phase, we look into an interesting application of our algorithm, ie., unsupervised discovery of object categories from a huge 3D image dataset of household objects.

Figure 1.1: Microsoft Kinect

Object recognition using RGB-D data is still a relatively less explored area. But this field is becoming more and more popular, since cheaper technologies like *Microsoft Kinect* to capture 3D data are being introduced. Kinect came as a part of XBOX, Microsoft's gaming console, and uses infrared structured light to capture the depth data. The image captured using Kinect is not exactly full 3D, but it is more like a 2D image with an extra depth information associated with every pixel, along with the color information. Nevertheless, this is a giant leap from the traditional imaging technologies, starting a whole new era in computer vision and its applications.

1.1 Manifold based clustering

A manifold is a topological space that locally resembles a euclidean space at small scales. That is, they are topological structures such that each point on them has a neighborhood that is homeomorphic to a open set of the Euclidean space. In machine learning, one encounters data in high dimensions, e.g. RGB-D data may have an m x n image, with 3 RGB colors and one depth per pixel. This implies that it has $m \cdot n$ dimensions, i.e. each image is a point in a R^{mn} dimensional space, where the value of the point is a 4-vector of RGBD. Let's say we are considering the task of distinguishing an apple from a lemon and a banana. Now, all possible images of an apple (or all these objects altogether) constitute a vanishingly small fraction of all the possible images in the R^{mn} space. Thus, the all images in the class apple would lie on some subspace of the image space, which can be said with probability approaching 1, to be

dominantly embedded in some lower dimensional subspace. In manifold-based dimensionality reduction algorithms, it is assumed that these lower-dimensional subspaces are smooth manifolds - i.e. images that are similar have in-between images that continuously deform one to the other, and also that everywhere in the subspace, the local neighborhoods resemble an euclidean disk of the some dimension, say d. The task then is to estimate d and also to construct an embedding of the mn-dimensional image space for the classes apple, and discriminate these from the classes lemon or banana. In a supervised task each image would be associated with labels such as "apple" or "banana". In an unsupervised classification task, we may assume that each of these objects lie on separable manifolds, and we wish to identify all manifolds in the mixed-up image sets and characterize the objects in this manner.

As an example, a line or a circle has a intrinsic dimensionality of 1, irrespective of the dimension of the space in which they are drown. Study of manifolds are particularly interesting in machine learning and data mining, since sometimes data sampled from in a very high dimension could form some smooth manifolds of much lower dimension, and if one could detect it, then those data points can be mapped down to the lower dimension, making the learning/mining much more effective and efficient. However, in this thesis we are *not* concerned about mapping these high dimensional manifolds into their respective lower dimensional embedding. Rather we propose an algorithm that would cluster the points sampled from a multi-manifold embedded high dimensional space to the corresponding manifold they belong to. This manifold separation algorithm can be effectively used in classification/clustering problems, provided there is some topological structure to the data. Manifold based machine learning approaches are particularly interesting in a computer vision standpoint, since, very often image datasets would contain images that are very much alike, with some slight changes in parameters like viewpoint, light intensity or orientation. Such images would in fact be lying on some manifold, that automatically characterizes the way the parameters changes from image to image, enabling the model free interpretation of image data. Similarly, if one is analyzing activities in videos, then consecutive frames would be very much alike, and frames containing some specific activity would form a smooth manifold in the image space. Thus by separating these manifolds, we in turn would be separating the activities involved in the video.

Figure 1.2: Examples for intersecting manifolds (Dub11)

But in most practical cases, these constituent manifolds would be intermingled in complex ways. Hence identifying and and separating these embedded manifolds are very difficult. The separation of intersecting manifolds is not that straightforward, since one need to define why a point would belong to a particular manifold rather than any other. For this we have to see how humans perceive and interprets manifolds.

There are many psychological theories trying to explain human perception, *gestalt perception* being the prominent among them. There has been studies which investigated the relationship between manifolds and gestalt perception. For example, (CCM+07) uses manifold learning to explain human perception of architecture of buildings. Refer the part(a) of the figure 1.3. We as humans tends to interpret them as two separate circles, even though there are other possible interpretations(part(b), part(c)).This can be explained by gestalt laws of grouping. More on gestalt perception and how it ties up with our algorithm, can be seen in chapter 2 in detail.

1.1.1 Traditional dimensionality reduction techniques

Classical dimensionality reduction techniques on a dataset generally rely on either Principle Component Analysis(PCA) (Jol02), or Independent Component Analysis(ICA) (HKO01). These techniques tries to find a basis for the observed data, and represent them as a linear combination of mutually independent basis vectors. This is very efficient if the underlying data has a linear structure. But, as in most of the cases, data could lie on a non-linear complex structure with possibly an very low intrinsic dimensionality. But

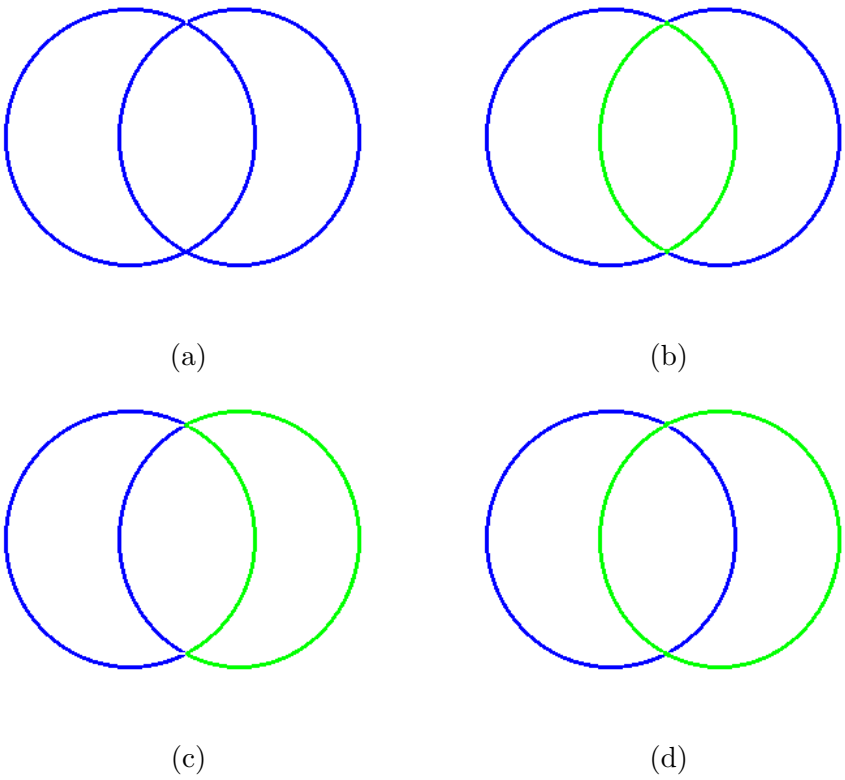


Figure 1.3: Even though the constituent manifolds in figure (a) can be interpreted as (b), (c) and (d), we as humans tends to prefer (d) as the preferred interpretation

despite of the low dimensionality of the data, linear dimensional techniques like PCA would not be able to find its intrinsic dimension (TDSL00).

This led to the introduction of a series of new methods that tried to estimate the low dimensional non-linear manifolds, by analyzing the local neighborhood of a point and preserving the local properties in the low dimensional embeddings. One of them was the Isomap algorithm (TDSL00), which, in fact, was the one to introduce the term NLDR(Non Linear Dimensionality Reduction) (LV07). Most NLDR algos assume that the data has been densely sampled from a smooth d-dimensional manifold. Thus, small neighborhoods on the manifold are taken to constitute linear d-dimensional spaces,n and these are often combined on a graph, the nodes of which represent each data point, and edges are instantiated only between near neighbors.

In the Isomap algorithm , the distance between any two points is set to their eu-

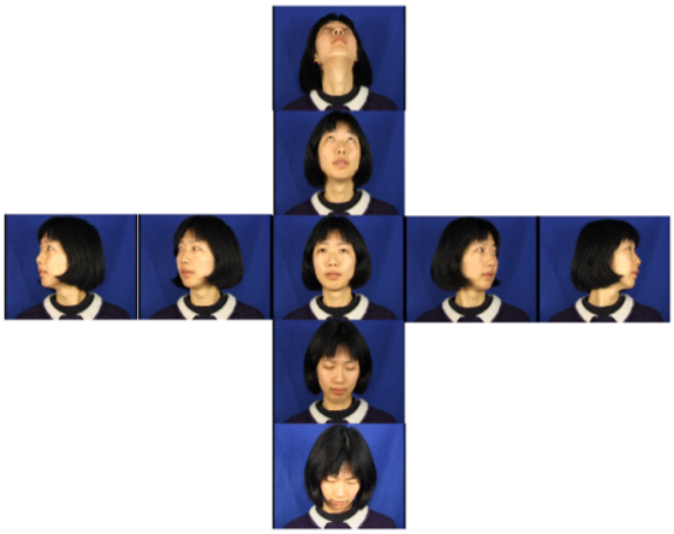


Figure 1.4: Real-life example of intersecting manifold. Two separate actions (head up-down movement and head left-right movement) sharing some similar frames.Source: (Dub11)

clidean distance if they are within this small neighborhood. Otherwise, assuming that they lie on a connected component of the graph, a a shortest path connecting them is found on the neighborhood graph. This shortest path can be thought of as an approximation of the geodesic path along the manifold. By adding the lengths of the edges along the path, the isomap estimates the geodesic distance (distance measured along the surface of the manifold) from one point to the other. Thus, Isomap imagines that the manifold is constructed by stitching together the various local neighborhoods. Once these distances have been computed, Isomap uses the classical linear dimensionality reduction technique of MDS to map the input data into a lower-dimensional space while minimizing the distortion in the geodesic distances. Locally Linear Embedding(LLE) (RS00) another example for a NLDR algorithm, and it tries to preserve the neighborhood structure by representing a point as a weighted combination of its neighbors in lower dimension.

But all these methods safely assume that the data is sampled from separate manifolds and hence they can not handle intersecting manifolds. But we are more concerned about a mixed-manifold embedded space with possible mutual and self intersecting manifolds. We would like to identify and separate these manifolds in the high dimen-

sional space itself, and thus clustering the points based on the manifold it belongs to. There are two types of mixed manifolds. A single manifold can be mixed, if its dimensionality changes from one part of the manifold to other. Our algorithm is not designed to handle thsese sort of manifolds. They are meant for the second type of mixed manifolds, which is a collection of multiple manifolds, each with a fixed intrinsic dimension, possibly intersecting others and themselves.

1.1.2 Analyzing intersecting manifolds

Despite of being one of the powerful ideas in data mining and machine learning, manifold based clustering has not been exhaustively explored yet. But there have been some interesting studies in manifold separation in the past. Some of them were in *neighborhood selection*, where points belonging to manifolds are identified in the high dimensional space itself, where other studies even tried map these identified points into their lower dimensional projections.

In (BM05), Bengio et al., discuss the problem of *multi-manifolds*, where the data comes from a large set of underlying low dimensional manifolds. They suggested a non-local manifold learning algorithm which attempted to discover shared structure in the tangent planes at different positions thus identifying the constituting manifolds. But this algorithm assumed that even though there could be multiple manifolds in the high dimensional space, none of the intersects with each other and this is the major drawback of this method.

A major contribution to neighborhood selection was by Souvenir et al (SP05). Their algorithm was general enough to handle multiple intersecting manifolds. They used Expectation Maximization and wighted MDS to cluster the manifold points. The algorithm accepted three parameters.

- Number of manifolds and their dimensionality
- Threshold variance: Determines when to stop iteration. If the residual variance of error in lower dimension embedding falls bellow this level, algorithm assumes all the points has been correctly clustered and terminates.
- Maximum number of iteration: Since this is an iterative algorithm, one needs to define maximum number of iterations allowed to reach a convergence. Algorithm terminates once this limit is reached, often with out satisfactory clusters.

The algorithm runs in two steps. In E step, each data point is assigned to the best fit manifold. In M step, model parameter are updated based on the embedding error occurred while mapping the points to the lower dimensional manifold that was assigned to them in E step. Embedding error is calculated by a wighted variant of the classical MDS algorithm, which they called WMDS. When ran iteratively, these EM steps are supposed to converge, giving the optimal manifold clusters. The major merits of this algorithm are,

- Handles intersecting manifolds
- Algorithm is iterative, implying that the global structure of the manifold is also taken into account while doing the clustering. This is something that our algorithm can not do.

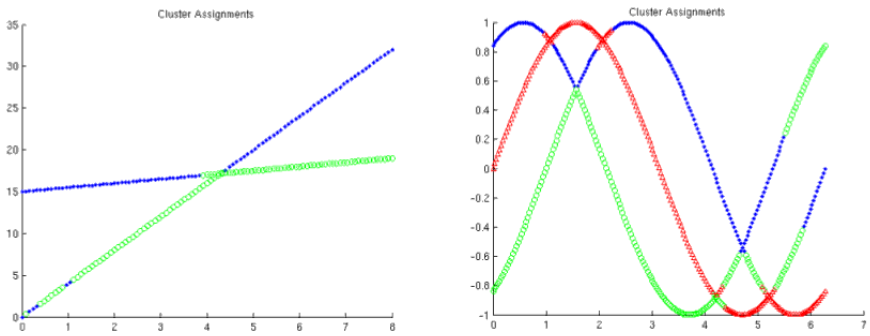


Figure 1.5: Results of Souvenir et al's manifold clustering algorithm as reported as Nandan et al. (Dub11)

Despite of these attractive merits, the algorithm suffers due to following facts.

- Most of the time the algorithm doesn't converge. Even if they converge, it has been noticed that the results are not accurate enough.1.5.
- The other side of the algorithm being iterative is that the computational time required is too heavy that algorithm can not be employed in any practical situation with large number of data points.

Recently Dubey et al. proposed a *manifold pursuit algorithm* that claimed to effectively separate intersecting manifolds. The inputs parameters required for this algorithm are,

- Number of manifolds.
- Dimensionality of each manifolds.
- A safe starting point on each manifold.
- th_{spread}, the upper limit on the allowed error in selecting a point as the part of the neighborhood of a point.
- th_{curv}, Maximum value of curvature allowed when local neighborhoods of two points are merged together while growing a manifold.
- k_{max}, the maximum neighborhood size of any point.

As the first step, they defined a *tangent space* for every point in the data set. It is a local d dimensional orthonormal coordinate system, such that some k neighbors of the point are optimally represented in the defined coordinate system with minimal error.

Then, for every point in the manifold, a local neighborhood is found out adaptively. This is done by calculating k_max neighbors of every point, and then deleting all those points which has an embedding error more than th_{spread}. Embedding error shows how well a member of the neighborhood of a point p_i can be represented in the tangent space defined at p_i. Embedding error of a point p_j in the neighborhood of the point p_i can be calculated as,

$$\epsilon^*_{ij} = ||z_{ij} - \sum_{l=1}^{l=k} (z_{ij}.\hat{\mathbf{v}}_{il})\hat{\mathbf{v}}_{il}|| \tag{1.1}$$

Once the tangent space and local neighborhood are defined for every point in the manifold, a graph is grown from each given starting point. Let p_i be a point that already has been added to the current manifold. Let point p_j be a member of the neighborhood of the point p_i, whose membership in the current manifold has to be evaluated. This is carried out based on two parameters namely subspace-subspace distance(SSD) and spread. SSD basically defines a distance between the tangent space at the two points p_i and p_j. It is defined as follows.
Let S_1 and S_2 be the two tangent space defined at the points p_1 and p_2 respectively. Let

$\{\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_d\}$ be an orthonormal basis of $S1$ and $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_d\}$ be the orthonormal basis for $S2$. Let $d_H(\mathbf{u}_i, S2)$ denote the so-called L_2 -hausdorff distance which is defined between a vector and a subspace ($S2$) as

$$d_H(\mathbf{u}_i, S2) = min_{\mathbf{v} \in S2} ||\mathbf{u}_i - \mathbf{v}|| \quad \text{where, } \mathbf{u}_i \text{ is any vector.} \tag{1.2}$$

Then SSD between S_i and S_j is defined as

$$dist(S1, S2) = \sqrt{\sum_{i=1}^{d} d_H^2(\hat{\mathbf{u}}_i, S2)} \tag{1.3}$$

Intuitively, the distance of two subspaces should reflect the difference between their directions.

Spread, on the other hand, depicts how well the new point p_j can be represented in the tangent space S_i at the point p_i. It is defined as,

$$spread(p_i, p_j)_{Si} = \frac{||(p_i - p_j) - (p_i - p_j).S_i||}{||(p_i - p_j).S_i||} \tag{1.4}$$

Using these two parameters, every point in the neighborhood of the point p_i is ranked, and satisfactory points are added into the current manifold.

Some of the drawbacks of these algorithm are,

- Too many parameters to be specified. Hence wont work if one doesn't have much prior knowledge about the data.

- SSD and spread are not strong enough to handle intersections of manifolds. The reason is that, assume the two points p_i and p_j are sampled near to the intersection of two manifolds such that p_j belongs to a different manifold than that of p_i. If they are adequately close to each other, the neighborhood of both p_i and p_j would have a lot of common points, making SSD and spread between these two points small. Closer these points are, lesser would be the value of these parameters. Thus, as the manifold growing reaches at p_i, spread and SSD of p_j would have very similar values to the values of other nearby points that belong to the same manifold as p_i belongs to. This would confuse the algorithm at manifold junctions and thus a good result can not be always guaranteed.

In 2011, Mike Gashler et al. proposed a adaptive neighborhood finding algorithm named *saffron* (GM11). Generally, to find a proper neighborhood of a point, the structure of the manifold around the point has to be known a priori. But the shape of the manifold can not be estimated until the whole manifold is grown for which the neighborhoods of every points has to be calculated first. *Saffron* algorithm claims to have addressed this. It iteratively approximate the tangent space around a point, and comes up with a set of neighbors that matches with the approximated tangent space. Initially, to find a neighborhood of size k around a point p_i, the algorithm starts with a set candidate neighborhood points whose cardinality is bigger than k. In the beginning, an equal weight is given to all the candidate neighborhood points, and using them, the tangent space for the point p_i is calculated. Now, the weights of each candidate neighbors are reevaluated again based on how well the points fit to the calculated tangent space at p_i. "Membershipness" of a point p_j to the tangent space S_i at point p_i is quantized using two angles called monohedral angle and dihedral angle. Monohedral angle is computed by projecting point p_j onto S_i, and then computing the angle formed by the three points p_j , p_i, and $p_i + S_i(p_j, p_i)$. Equation 1.5 computes the cosine of monohedral angle.

$$m(p_i, S_i, p_j) = \frac{||S_i(p_j - p_i)||}{||(p_j - p_i)||} \tag{1.5}$$

The dihedral (two-surfaces) angle is dened between two tangent spaces, S_i and S_j. Informally, it is the angle formed between the normals of the surfaces at p_i and p_j. The equation 1.6 defines how well the point p_j belongs to the space S_i.

$$\alpha(p_i, S_i, p_j, S - j) = m(p_i, S_i, p_j)^2 * m(p_j, S_j, p_i)^2 * d(S_i, S_j) \tag{1.6}$$

Weights of the candidate neighborhood points are reassigned accordingly, and the process is continued until the convergence is attained. Again, this algorithm can not be effective near the intersection of two manifolds candidate neighbors of a point near the junction of two manifold would contain points from both the manifold, resulting the algorithm to converge providing undesirable tangent space of the point.

1.2 3D perception and Pointclouds

Object recognition is a well studied topic in computer vision, with much progress made in recent years. As of now, there are some impressive object recognition systems and

algorithms (Goo11) that are (almost) comparable with human capabilities in recognition. With the introduction of Microsoft's Kinect, a low cost camera that can capture depth data along with the RGB image, the field of RGB-D images got a significant boost.

Figure 1.6: A colorless pointcloud

Pointclouds are 3D counterparts of images. They are a collection of 3D points with the three coordinate values and possibly with RGB color information for each of the points. The pointcloud captured from Kinect is not exactly in full 3D, but more like a 2.5D image (Mar74), with X,Y and Z world coordinates associated with each pixel.

1.2.1 Kinect RGB-D object dataset

In this work, we shall be using Washington University's RGB-D object dataset (LBRF11a), an excellent dataset of pointclouds of household objects, captured using Kinect in a systematic way. The dataset contains pointclouds, RGB images, and depth images of 51 categories of household objects. The images are of 640x480 resolution, scanned at the rate of 30Hz. For each category, there are multiple instances available, hence the dataset can be used for instance detection as well. For each instance of an object, there are 3 image sequences shot with Kinect at three different heights, approximately 30, 45 and 60 degree above the horizon, which captures multiple views of the object. Each of these three video shots has around 250 different views that spans all the 360 degrees around the object. This multiple views of each of these object would form a strong manifold in a rightly chosen higher dimensional space.

Figure 1.7: A few of the objects in RGBD dataset

Our objective is to use our manifold based spatial clustering algorithm to discover the various objects in the dataset, with out any supervision. We expect the multiple views of every object would form a manifold and thus by separating the manifolds, we should be able to discover object classes. More on pointclouds and the results of applying our algorithm on RGB-D dataset can be seen in chapter 4.

1.3 Organization of the thesis

The thesis is organized into five chapters. The second chapter, the core of this thesis, describes the newly proposed manifold based clustering algorithm in detail. A brief note on gestalt perception and how it is connected with our algorithm, is also given. In the third chapter, we introduce a new distance measure called Correlation Based Earth Mover Distance, which is very effective for comparing two data points drown from a distribution. Fourth chapter presents the results of using our algorithm in object class discovery. The chapter also contains brief description about three feature vectors that we used in the process. The final chapter is mainly the conclusion and some possible future works.

2

Manifold based spatial clustering

2.1 Problem statement

Let $X = \{x_1, x_2, .., x_n\}$ be the dataset where x_i is sampled from $\Re^D$, a D dimensional space. Each x_i is sampled from an underlying manifold of dimension d, where $d << D$. But neither the number of manifolds(eg: number of objects) nor their dimensionality is known before hand. One merely assumes that the algorithm initialization point does not lie on any manifold intersection regions. The aim is to associate every $x_i \in X$ with some label $l_j \in \{l_1, l_2, .., l_m\}$ such that $\forall x_i \in X$, one and only one label l_j would be associated. Note that m, the number of manifolds involved, need not have to be known before hand.

2.2 The proposed manifold based clustering algorithm

Unlike the algorithms like Isomap(TDSL00) and LLE(RS00), we are not trying to find the lower dimension projection of the high dimensional data. We are only interested in clustering the points from the higher dimensional space based on the corresponding manifold they are drawn from. Thus we would be separating the manifold in the higher dimensional space rather than mapping them onto its intrinsic lower dimension subspace. So our algorithm is more of a manifold based *clustering* algorithm than a manifold identification algorithm.

Manifolds also can be impure, meaning, a single manifold would have different dimensionality at different parts. Our algorithm is not designed to handle these cases, hence it just ignores this situation.

If the manifolds are separable in the high dimensional space, a simple graph growing algorithm would cluster the manifolds. Graph growing algorithms starts from a random point in the data, and grows a graph, by connecting the point to its k nearest neighbors. Since the manifolds are not intersecting each other, all the manifolds would be separated, provided an apt value for k is chosen.

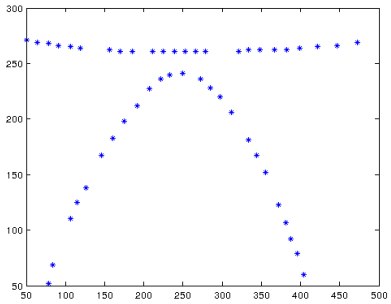
But having a static k may not always give one good results. A small k could result in the breakage of a manifold into multiple smaller sub-manifold patches, while a larger k would bring too much freedom in graph growing, resulting the merging of multiple manifolds into a single connected component. Refer figure 2.1 and section 2.2.1 for more details. Hence, *a manifold growing algorithm should have a k, that changes according to the local neighborhood of the manifold.*

Also, as a third case, manifolds could be intermingled with each other, making their separation extremely difficult. So when the graph growing reaches at a point where multiple manifolds cross over each other, the manifold growing algorithm should be smart enough to decide the direction in which it has to proceed. This can be made possible only by analyzing the structure of the manifold that it has grown till then. Thus, as the second property, *the ideal manifold growing algorithm should analyze the shape/structure of the neighborhood of a point to make sure that the graph is grown in the right direction from the point.*

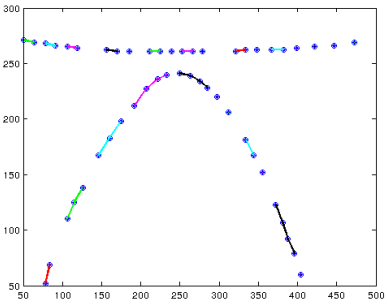
Since our proposed manifold growing algorithm takes care of both the cases discussed above, it is powerful enough to separate even the complicatedly intermingled manifolds. Based on this, the two major features of our algorithm are *dynamic branching* and *continuity score*

2.2.1 Dynamic branching factor

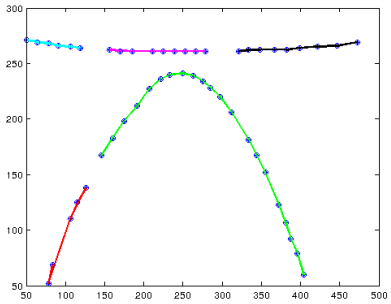
Problem with the static k is handled in our algorithm by having an lower limit and an upper limit on the branching factor namely k_{min} and k_{max} respectively. Hence, after the graph growing, each node would have at least k_{min} edges starting from it. At each node x_i, the algorithm would start with a branching factor k_{min}. If all the k_{min} closest neighbors are unvisited, then the graph simply grows to include all these k_{min} neighbors. On the other hand, if some of the k_{min} neighbors of x_i are already visited, then the k is iteratively incremented until k_{min} unvisited neighbors are found



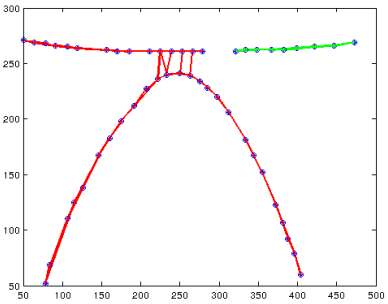
Two synthetic 2-dimensional manifolds



Naive graph growing with k=2
Number of clusters: 27



Naive graph growing with k=3
Number of clusters: 5



Naive graph growing with k=4
Number of clusters: 2

Figure 2.1: Results of running graph growing algorithm with static k to separate two synthetic manifolds. Lower branching factor left the discovered manifold in patches, while increasing the branching factor resulted in merging of the manifolds together.

or till $k > k_{max}$. k_{min} and k_{max} are the parameters of the algorithm, and should be set keeping the expected minimum inter-manifold distances in mind.

As seen in figure 2.2, dynamic branching factor helps the algorithm to satisfy the gestalt low of closure.

2.2.2 Continuity score

Continuity score between two points are determined by two factors. Firstly, the points should not be to far from each other, and secondly, they both should lie on the local d dimensional hyperplane. These two factors are captured by the two parameters that we are going to introduce, namely *scaled projected distance* and *embedding error*.

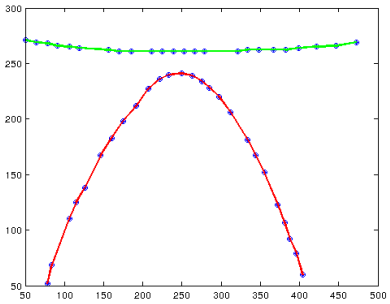


Figure 2.2: Result of running tha graph growing algorithm with dynamic k on the same manifolds as shown in 2.2, with $k_{min} = 1$ and $k_{max} = 9$. The manifolds are separated perfectly as one would expect.

Now, if the intrinsic dimensionality of manifold is d, then each point on it has a d-dimensional tangent space defined by the d basis vectors, also called tangent bundles. In manifold learning from the data points, we don't have a continuous manifold and hence we can not define a continuum which can give us the tangent vectors. To counter this problem, we define the tangent space by finding the linear d- dimensional eigen space for the local neighborhood. We define a linear subspace(S) of dimension d by the principal components of the mean centered data points. These principle components are calculated just as in the manifold pursuit algorithm(Dub11), but with a small difference. In order to find the tangent space at a point x_i, instead of finding a best fit vector space taking k nearest neighbors of x_i, we take k *visited* neighbors of x_i that are the part of currently growing manifold. Then a principal component analysis(Jol02) is done on these k visited neighbors to identify d principal components, whose eigen values are non-zeros. The reason why we insist that the tangent space has to be calculated from the visited neighbors is that, it make sure that point chosen to calculate tangent space comes from the currently growing manifold. We represent the approximated tangent space together with the eigen value corresponding to each basis vector at a point x_i by,

$$S_i = \{(e_1, \lambda_1), (e_2, \lambda_2), .., (e_d, \lambda_d)\} \tag{2.1}$$

where,

e_k: Unit eigen vector along the k^{th} principle component.

λ_k: Eigen value corresponding to k^{th} eigen vector.

Note that one can not pre-compute the tangent space at every point before the manifold growing algorithm is initiated, since the *unvisited neighbors* of a point can be determined only while the manifold is being grown.

Based on these, lets define what *continuity score* is. Assume that the manifold has been grown till some point x_i. While trying to grow the graph from x_i in the manifold, the proposed algorithm considers the local neighborhood around x_i. Let x_j be a point in the vicinity of the point x_i, to which the manifold could potentially grow in the next step. Before adding the new point x_j into the manifold, our algorithm first conforms that the point added is in consensus with the structure of the manifold grown so far. The decision is made based on a weighted combination of scaled distances of x_j from the local neighborhood of the point x_j, and the deviation from the local tangent space. This is termed as *continuity score*, a parameter that suggests how well the new point x_j matches with the local topology of the manifold around point x_i.

Let $S_i = \{(e_1, \lambda_1), (e_2, \lambda_2), .., (e_d, \lambda_d)\}$ be the eigen decomposition at the point x_i. The *continuity score* of x_j is calculated as per algorithm 1. It is basically the weighted sum of two terms, namely *scaled projected distance* and *embedding error*. *Scaled projected distance* of a point x_j shows the distance of the point from the origin of the tangent space S_i, *along* the surface of the grown manifold, calculated as in equation 2.2.

$$scaled\_projected\_distance(S_i, x_j) = \sum_{k=1}^{d} \frac{\Lambda}{\sqrt{\lambda_k}} \cdot < e_k, (x_j - \mu_i) > \tag{2.2}$$

where,
μ_i is the origin of the tangent space at x_i.
$\Lambda = \sum_{k=1}^{d} \sqrt{\lambda_k}$

Now, $\sqrt{\lambda_k}$, the square root of the eigen value corresponding to an eigen vector e_k, is in fact the standard deviation of the points along the k^{th} principal component(Jol02). Thus $\frac{\sqrt{\lambda_k}}{\Lambda}$ is the corresponding normalized standard deviation. Dividing components of a point x_j along each principal component by the normalized standard deviation along the principal component, would relax the distance *along* the surface of the local patch of the manifold, and would amplify any deviation from the manifold surface. That is, one unit of distance along the surface of the manifold would contribute much lesser to the *scaled projected distance* than one unit of distance perpendicular to the surface of the manifold.

Embedding error (Dub11) at x_j, on the other hand, shows how well the tangent space S_i at the point x_i captures the point x_j. It is the error of representing the point x_j as a linear combination of the basis vectors of S_i and is calculated as in equation 2.3.

$$embedding\_error(S_i, x_j) = |x_j - \sum_{k=1}^{d}(< e_k, (x_j - \mu_i) > \cdot e_k)| \tag{2.3}$$

where μ_i is the origin of the tangent space S_i.

The final continuity score of the point x_j is calculated as the weighted sum of two components, as bellow,

$$\begin{aligned} continuity\_score(S_i, x_j) = \alpha \cdot scaled\_projected\_distance(S_i, x_j) + \\ (1 - \alpha) \cdot embedding\_error(S_i, x_j) \end{aligned}$$

where, α determines the weight given for the two factors, *scaled projected distance* and *embedding error*. Lower the α, the higher would be the weight given for *embedding error*, enabling the algorithm to prefer the low curvature direction while growing the manifold. Since *continuity score* quantizes the smoothness of the manifold growth, one can say that it directly implements gestalt principle of continuity. Figure 2.3 shows how effective the continuity scores really are in growing the manifold.

To decide α, one need to test the relative importance of *scaled projected distance* and *embedding error*(Figure 2.4). Lower the α, higher is the weight given to the embedding error, and vise versa. Consider part (a) of the figure. Since $\alpha = 0$, scaled projected distance is completely ignored, making the embedding error the only criterion for choosing points. When all the k_{max} points are coming from the same manifold, their embedding error would be almost zero, making the selection of k_{min} points from k_{max} random. This makes the manifold growing stop abruptly, dividing the structure in to fragments. In part (c), α has raised to 0.3. As one can see, still circles can not be separated. However, at $alpha = 0.5$ the algorithm is found to separate the two circles. As one keep on increasing α, less weight is given to the embedding error, reducing the penalty of taking turns along the manifold. Thus, as expected, in part (d) of the figure, extracted manifolds are not circles, but ones with sharp corners. Thus, scaled projected distance basically captures the distance from a point from a surface *patch*, as where

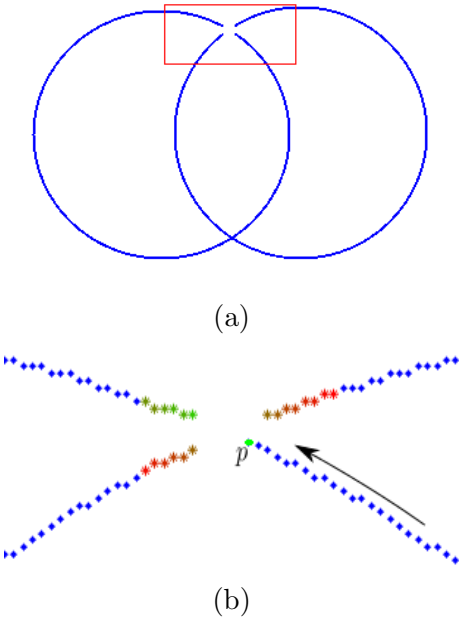


Figure 2.3:
part(a): Two intersecting circles. Note the intersecting part marked in red. There is a gap introduced between the circles.
part(b): *Continuity score* at the junction. Manifold is grown till the point p, in the direction of the arrow shown. Candidate neighbors of p are color coded based on their local *continuity*. Greener the points, better they belongs to the grown manifold. Redder the point, worse they fit in to the current manifold.

embedding error indicates how sharply the surface curvature changes along the surface of the manifold.

k_{min} and k_{max} on the other hand, are set according to the expected variance in the data and gaps in the manifold. k_{min} determined the degree of growth of the manifold. Higher k_{min} implies the graph is grown into more points from every point. A higher k_{min} is preferred when the dimensionality of the expected manifold is high. k_{max} on the other hand, determines the maximum number of points that the algorithm can consider to choose k_{min} neighbors of a point. This essentially indicates the gaps in the manifold. If there are bigger gaps in the manifold, a larger k_{max} would be needed to bridge them. Increasing the k_{max} also increases the chance of merging one manifold into another. Thus k_{max} has to be set carefully.

Algorithm 1 Calculate *continuity_score*($x_j, visited\_neighbors$, α)
x_j: Point for which the score has to be calculated
visited_neighbors: The set of visited neighbors of the point x_i, on to which x_j has to be projected.
α: Weighing factor

1: t = cardinality(*visited_neighbors*)
2: Calculate the mean of *visited_neighbors*, $\mu_i = \frac{1}{t}\sum_{z=1}^{t} x_z$
3: Center *visited_neighbors* and x_j; $\forall x_p \in visited\_neighbors, x_p = x_p - \mu_i$ and $x_j = x_j - \mu_i$
4: Calculate C, the covariance matrix of *visited_neighbors*;
5: Calculate eigen vectors e_k and eigen values λ_k) of C, where $1 \leq k \leq d$
6: $\Lambda = \sum_{k=1}^{d} \sqrt{\lambda_k}$
7: $scaled\_projected\_distance = \sum_{k=1}^{d} \frac{\Lambda}{\sqrt{\lambda_k}} \cdot < e_k, (x_j - \mu_i) >$
8: $embedding\_error = |x_j - \sum_{k=1}^{d} (< e_k, (x_j - \mu_i) > \cdot e_k)|$
9: $continuity\_score(S_i, x_j) = \alpha \cdot scaled\_projected\_distance(S_i, x_j) + (1 - \alpha) \cdot embedding\_error(S_i, x_j)$
10: **return** *continuity_score*

2.2.3 Informal description of the algorithm

The proposed algorithm make use of both the properties explained above, the dynamic branching factor and continuity score parameter. The algorithm is initiated from a random point in the set of data points. At anytime, manifold would have a "growing frontier" consisting of points/nodes from the boundary of the manifold grown so far. We call this the *open nodes*. Let $manifold\_id$ be a counter to keep track the manifolds being discovered, and initialize it to 1 in the beginning. A node to be expanded, say x_i, is chosen from the set of open nodes. Then find k_{max} neighbors of x_i, including the already visited ones. Now, using the algorithm 1, calculate the *continuity score* for each of the calculated neighbors of x_i. Then, among from the k_{max} neighbors, identify k_{min} unvisited nodes with the least *continuity score*. Connect all the k_{min} newly discovered nodes to x_i, and label them as $manifold\_id$. Finally push all the newly discovered nodes to the *open nodes* queue, and repeat the procedure. If *open nodes* becomes empty, that implies that current manifold has been traversed fully. Increment the $manifold\_id$ and repeat the whole procedure from a randomly chosen unvisited node until all the nodes are visited. The complete steps are shown in algorithm 2.

Algorithm 2 Cluster manifolds(X, k_{min}, k_{max}, k_{patch}, α)

$X = \{x_1, x_2, .., x_n\}$: Set of points to be clustered

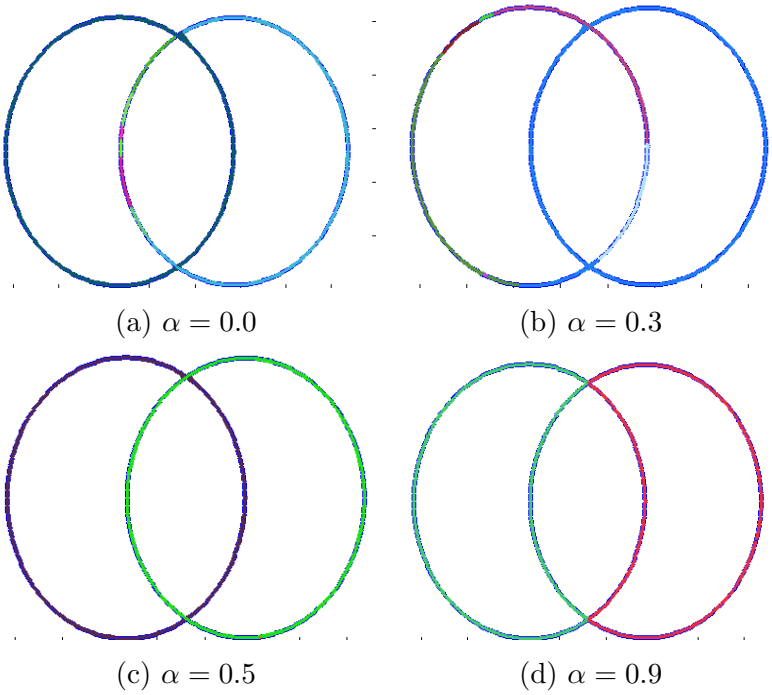
k_{min}:Lower bound of k

k_{max}:Upper bound of k

k_{patch}: Neighborhood size on which PCA has to be performed

α: weighing factor

Initialize array $visited[n] = 0$
Initialize array $label[n] = 0$
$open\_nodes = \{r\}$ where r the index of some randomly chosen point x_r from X
Initialize $manifold\_id = 1$
while ($\exists i$ *such that* $visited[i] \neq 1$) **do**
    if ($open\_nodes$ is empty) **then**
        $manifold\_id = manifold\_id + 1$
        $open\_nodes = r$, where r is the index of a random unvisited point x_r from X
        continue
    end if
    $current = open\_nodes.dequeue()$
    $visited[current] = 1$
    $visited_neighbors$= k_{patch} visited, nearest neighbors of $x_current$.
    $candidate\_neighbors$ = closest k_{max} neighbors of $x_{current}$
    $candidate\_neighbors = \{x_i \in candidate\_neighbors\ s.t\ visited[i] = 0\}$
    $\forall x_i \in candidate\_neighbors$, calculate $continuity\_score(x_i, visited_neighbors)$ by algorithm 1
    Sort $candidate\_neighbors$ according to $continuity\_scores$
    $new\_nodes = \{\}$
    for $i = 1$ to $min(k_{min}, |candidate\_neighbors|)$ **do**
        $new\_nodes = candidate\_neighbors[i]$
    end for
    $\forall x_i \in new\_nodes$, update $label[i] = manifold\_id$
    $open\_nodes.enqueue(new\_nodes)$
end while
return *label*



(a) $\alpha = 0.0$ (b) $\alpha = 0.3$

(c) $\alpha = 0.5$ (d) $\alpha = 0.9$

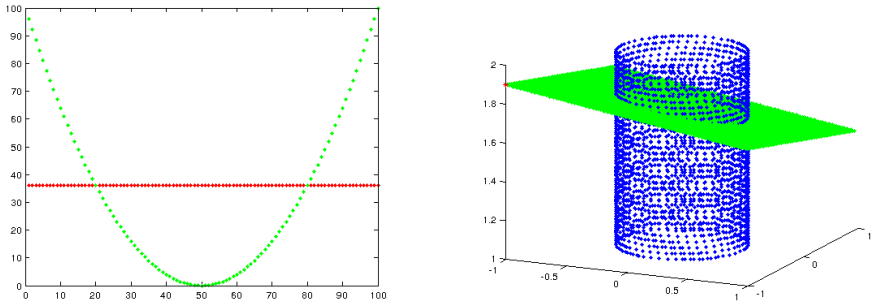
Figure 2.4: Effect of weighing factor α on continuity score

The complexity of the algorithm mainly depends on two factors. Complexity of the distance calculation, and complexity of the PCA algorithm. Let the time complexity of the distance calculation be f_d. PCA analysis of n points from a D dimensional space would take $O(D^3 + D^2 n)$ time (GVL96). Then the totl complexity of the *Cluster manifold* algorithm(2) would be, $O(n^2 \cdot f_d \cdot log(k_{max} + O(nk_{max}(D^3 + D^2 k_{patch}))$. Here, $O(n^2 \cdot f_d \cdot log(k_{max}))$ is the time taken to find k_{max} nearest neighbors of all the point using some distance metric, $O(nk_{max}(D^3 + D^2 k_{patch}))$ is the complexity of finding *continuity score* for k_{max} neighbors of every point.

As explained in the earlier chapter, our manifold algorithm models gestalt perception.

2.3 Gestalt perception

Gestalt theory of perception suggests that humans sees objects as a "whole" than as a collection of individual building parts. In other words, "the whole is greater than



Line and a curve Cylinder and a plane

Figure 2.5: Results of running our algorithm on synthetically created intersecting manifolds. Algorithm was able to separate the manifolds successfully.

the sum of the parts"(Pet77). That is, a brick wall is more than a collection of bricks and a rectangle is much more than a set of four lines. The "whole" object would be having some additional emergent properties that none of its constituent has. Gestalt psychologists reject the claim that we recognize object by identifying individual parts or features; instead, we see and recognize each object or a unit, as a whole(Gal09).

Consider the example given in figure 2.6. Part (a) shows a structure which could possibly be a combination of multiple manifolds. A manifold separation algorithm could treat the whole structure as a single manifold, or it could assume it consists of multiple manifolds, and might break it down to multiple sub-manifolds. Some of the possible breakdowns are shown in part (b), (c), and (d) of the figure 2.6. Perhaps we as humans, tend to prefer the one showed in part (d) which shows two complete circles. This can be explained by *gestalt principles of grouping*. Some of its main aspects are defined bellow as defined in (SMM07), and we would be trying to stick to these guidelines while designing our manifold growing algorithm.

2.3.1 Principle of Proximity

Principle of proximity states that when we perceive a group of objects, we tend to see the objects that are close to each other as forming a group. (Figure 2.7)

This property can be imposed on the manifold growing algorithm, by making sure that manifolds grow from a point to its neighbors first. This would be taken care in

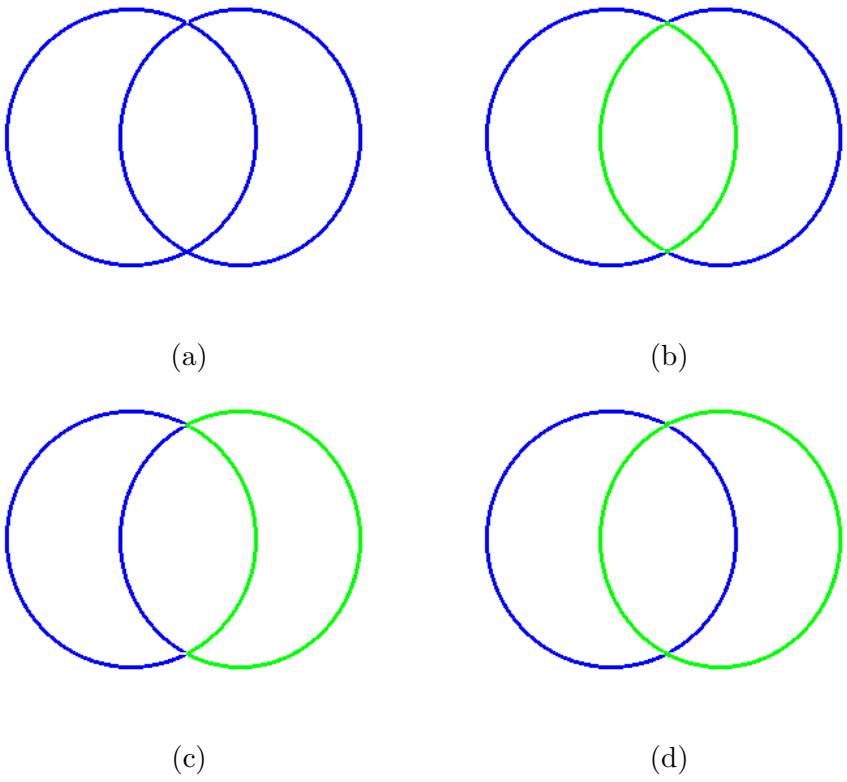


Figure 2.6: Different ways of interpreting the constituent manifolds in a multi-manifold embedded structure

most of the manifold growing algorithms((Dub11), (TDSL00), (GM11)), since in one way or the other, they would be considering *k-nearest neighbors* of a point before the manifold is grown from a point.

2.3.2 Principle of Closure

Low of closure states that humans tend to perceptually close up, or complete, object that are not, infact, complete. Our brain fills up the missing gaps and sees the object as a whole entity, thus generalizing it. Refer figure 2.9. Principle of closure is implemented in our algorithm by the *dynamic branching factor*, which is discussed in section 2.2.1.

Figure 2.9 shows a rectangle by the side of a circle. However, there are discontinuity in the shapes. But despite of the gaps, we perceive them as two separate shapes, rather than a set of discontinuous lines. To mimic human perception, a manifold growing

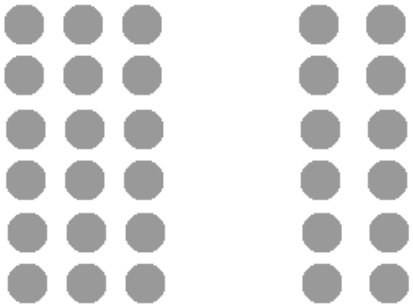


Figure 2.7: Gestalt principle of Proximity: We tends to perceive the collection of points as two clusters

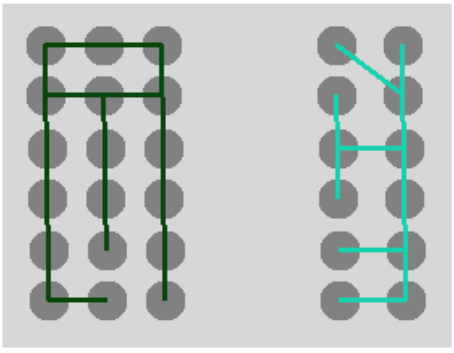


Figure 2.8: Result of applying our algorithm on the structure in figure 2.7

algorithm should have this property, since in most of the practical cases, manifolds in consideration would have uneven gaps in it due to irregular sampling or noise.

2.3.3 Principle of Continuity

Principle of suggests states that humans tends to perceive smoothly flowing or continuous forms rather than disrupted or discontinuous ones. This also explains why we tends to perceive the figure 2.6 as two separate circles as opposed to shapes shown in part (b) or part (c), since when viewed as two circles, it offers least amount interruption in the structure of the newly formed shapes. This idea is implemented in our algorithm

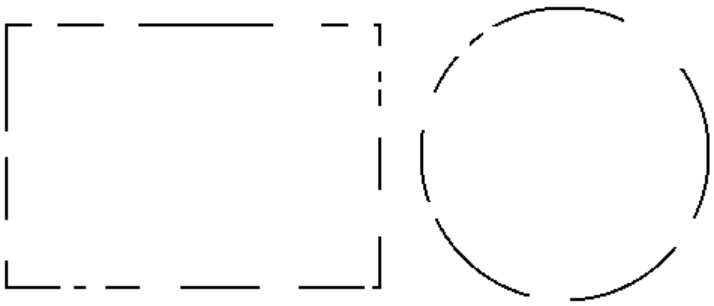


Figure 2.9: Gestalt principle of Closure

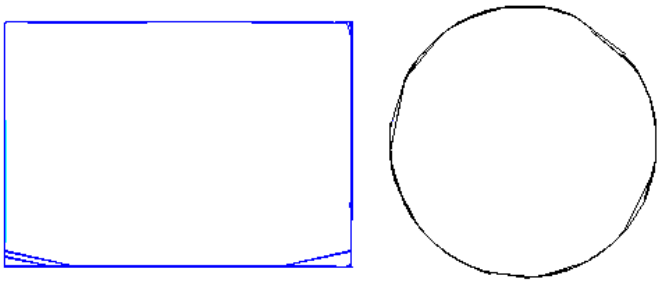


Figure 2.10: Result of applying our algorithm on the structure in Figure 2.9

with the concept of *continuity score*, that would be discussed in section 2.2.2.

There are three more principles in Gestalt laws of perception, but they are not relevant to the problem in hand. They are mentioned bellow.

Principle of similarity: Suggests that we tend to group objects that are visually similar (Figure 2.12).

Principle of symmetry: We tend to cluster objects that are mirror images of one another (Figure 2.13).

Principle of Figure-Ground: While perceiving a visual field, some object seems prominent(*figures*), while other aspects of the field reside into background(*ground*).

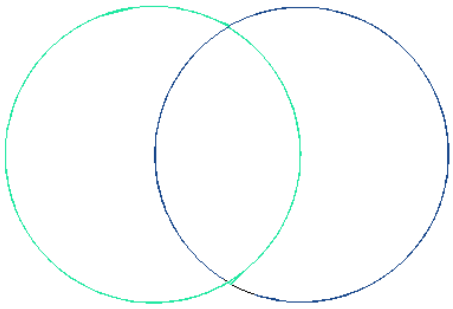


Figure 2.11: Result of applying our algorithm on the structure in Figure 2.6

As seen in figure 2.5, our algorithm works well in clustering artificially created manifolds. To test its performance on real world data, it was run on the Humaneva dataset, and the results are given bellow.

2.4 Results on Humaneva dataset

Humaneva(SB06) is a dataset from the Brown university, which mainly depicts human actions and poses. It contains 7 calibrated video sequences (4 gray scale and 3 color) that are synchronized with 3D body poses obtained from a motion capture system. The database contains 4 subjects performing a 6 common actions (e.g. walking, jogging, gesturing, etc.). Some of those videos contains multiple actions. We considered such a video sequence, where the subject walks around an empty room, followed by jogging, then hopping in the center of the room, and finally balancing himself on a single leg. In theory, all these actions should form a separate manifold in the image space. To test the theory, we ran our algorithm on the video sequence. Some of the interesting clusters identified are given in figures 2.14,2.15 and 2.16. Algorithm generally performed well on this dataset, even though it was not able to distinguish between running and walking. The possible reason could be the test images were too scaled down to separate these two similar actions.

The results presented here were on RGB data, with Euclidean as the distance measure. The experiments in chapter four are on RGB-D data, with the newly introduced

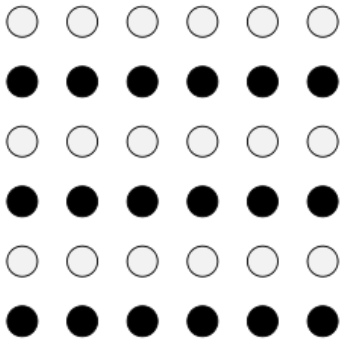


Figure 2.12: Gestalt principle of similarity

Figure 2.13: Gestalt principle of symmetry

CEMD distance as the distance measure.

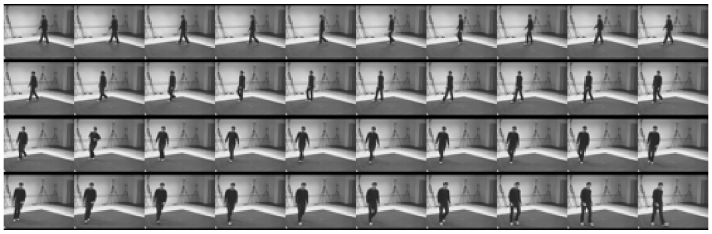


Figure 2.14: This cluster contains frames corresponding to the subject walking around the room. Note that no frames from the second round, where the subject was running, is present here

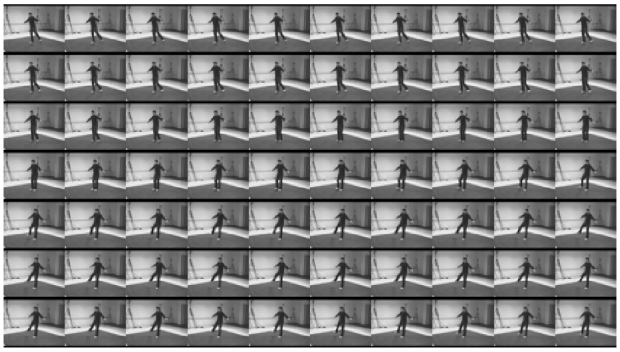


Figure 2.15: This cluster contains frames corresponding to the subject balancing himself on one leg

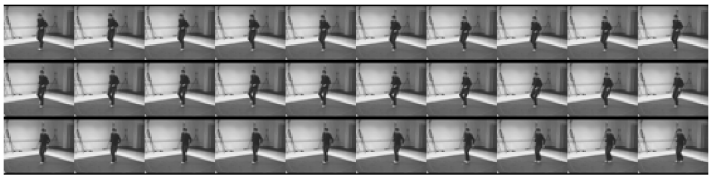


Figure 2.16: This cluster contains frames where the subject was hopping in the center of the room

3

Correlation based Earth Mover's Distance (CEMD)

Earth mover's distance(EMD) is defined between two probability distribution over a region R. Informally, if the distributions are seen as two piles of dirt over a region R, then EMD can be viewed as the minimum work required to transform one pile to another. "Work" done is calculated as the units of dirt moved times the distance they moved across the distribution. This is perfect for comparing two histograms. But before we proceed, one have to define the *ground distance*, ie the work needed to move a unit amount of dirt from one bin into another. If the bins are ordered systematically, the ground distance between i^{th} and j^{th} bin may defined as $(i - j)$. This make sure that if at all the contents of a bin are transfered, they are transfered to the nearest bin as possible. Often this is an acceptable assumption. But very well need not have to be true. Hence, we uses a modified version of EMD, where ground distance is defined in terms of correlation between the bins.

Suppose we have two histograms/signatures H_1 and H_2 both from a d dimensional space. We defined the ground distance as a measure that is inversely prepositional to the correlation between the two bins i and j. This results the two bins to have a small ground distance between them, if they are highly correlated. This make sure that the higher the correlation between two bins, easier it is to move points between the bins. This is exactly what we needed. The ground distance between the bins, g_{ij} is defined

as,

$$\begin{aligned} g_{ij} &= 1 - corr(S^i, S^j) \\ &= 1 - \frac{Cov(S^i, S^j)}{\sigma_i \sigma_j} \end{aligned} \tag{3.1}$$

with,

$$\begin{aligned} \sigma_i &= \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(H_k^i - \widehat{S^i})^2} \\ Cov(S^i, S^j) &= \frac{1}{n-1}\sum_{k=1}^{n}(H_k^i - \widehat{S^i})(H_k^j - \widehat{S^j}) \end{aligned}$$

where,
σ_i is the standard deviation of the i^{th} bin of the signature,
$Cov(S^i, S^j)$ is the covariance between i^{th} and j^{th} bin of the signature,
H_k^i is the value of i^{th} bin of the k^{th} signature,
$\widehat{S^i}$ is the mean value of the i^{th} bin,
n is the number of total number of signatures/data points.

With this definition of the ground distance, the new earth mover's distance between H_1 and H_2 calculated by estimating f_{ij}, the flow between i^{th} and j^{th} bin such that the following expression is minimized,

$$CEMD(H_1, H_2) = \frac{\sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij} \cdot g_{ij}}{\sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij}} \tag{3.2}$$

subjected to following constrains.

$$f_{ij} \geq 0 \text{ ; such that } 1 \leq i, j \leq d \tag{3.3}$$

$$\sum_{j=1}^{d} f_{ij} \leq H_1^i \text{ ; such that } 1 \leq i \leq d \tag{3.4}$$

$$\sum_{i=1}^{d} f_{ij} \leq H_2^j \text{ ; such that } 1 \leq j \leq d \tag{3.5}$$

$$\sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij} = min(\sum_{i=1}^{d} H_1^i, \sum_{j=1}^{d} H_1^j) \tag{3.6}$$

where,
H_1 and H_2 are the two signatures,

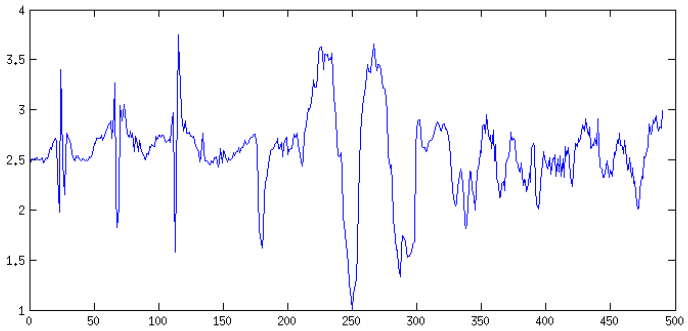


Figure 3.1: Ground distance between 250^{th} bin and the remaining bins. As expected, distance to the neighboring bins are small. Additionally, distance to some far away bins are identified to be small, presumably because those bins are highly correlated to the 250^{th} bin.

d is the number of bins, ie., dimension, of the signatures,

f_{ij} is the flow calculated between i^{th} bin and the j^{th} bin,

H_1^i and H_2^j are the values of i^{th} and j^{th} bins of H_1 and H_2 respectively.

This optimization can be seen as a instance of *transportation problem*, and can be solved using algorithm((KV06) section 9.5). More discussions on the algorithm and its completely can be seen in section 3.3.

3.1 Properties of CEMD

It can be seen the CEMD does not always qualifies as a *metric*. But still, its worth noting that CEMD possess following properties.

3.1.1 Non-Negativity

Lemma 1. *For any two signatures H_1 and H_2, $CEMD(H_1, H_2) \geq 0$*

Proof. For CEMD to be negative, either numerator or the denominator of Eqn.3.2 has to be negative. The denominator $\sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij}$ can not be negative since flow can never be negative due to the constrain 3.3. Neither the numerator can be negative, since correlation is always non-negative. Hence CEMD can never be negative. □

3.1.2 Identity of indiscernibles

Lemma 2. *For any two signatures H_1 and H_2, $CEMD(H_1, H_2) = 0$ if $H_1 = H_2$, but the converse is not necessarily true.*

Proof. We know that ground distance between a bin and itself is 0, since correlation between a bin and itself would be always 1, making $g_{ii} = 0$. Also, since $H_1 = H_2$, we know that $H_1^i = H_2^i$ for all $1 \leq i \leq d$. Now if we could define $f_{ij} = H_1^i$ for all $i = j$ and $f_{ij} = 0$ for all $i \neq j$, it can be seen that $CEMD(H_1, H_2) = 0$ with all the constrains satisfied. Previously, we already have shown that CEMD can not be negative. Hence $CEMD(H_1, H_2) = 0$ is the optimal solution, proving the claim.
Also, its noteworthy that the converse of this claim need not have to be always true. The nature of the ground distance g between the bins are to be blamed for this. As a counterexample, consider a case where all the bins are completely correlated to each other, making all $g_{ij} = 0$. From the eqn. 3.2, it can be directly seen that this makes CEMD 0, irrespective of the values of H_1 and H_2. □

3.1.3 Symmetry

Lemma 3. *For any two signatures H_1 and H_2, $CEMD(H_1, H_2) = CEMD(H_2, H_1)$.*

Proof. Since correlation is symmetric, $g_{ij} = g_{ji}$.
Now, $CEMD(H_1, H_2) \not> CEMD(H_2, H_1)$, since reversing the flows f_{ij} together with the fact that $g_{ij} = g_{ji}$ would make $CEMD(H_1, H_2) = CEMD(H_2, H_1)$.
Similarly, one can also show that $CEMD(H_2, H_1) \not> CEMD(H_1, H_2)$.
Form the above two statements, $CEMD(H_1, H_2) = CEMD(H_2, H_1)$. Hence the proof. □

3.1.4 Triangular inequality

Lemma 4. *For any three signatures H_1, H_2, and H_3, $CEMD(H_1, H_3) \leq CEMD(H_1, H_2) + CEMD(H_2, H_3)$.*

Proof. This is trivial since CEMD is calculated as an optimization problem of minimizing the expression 3.2. As defined earlier, CEMD between two histograms is defined as the minimum work required to rearrange points between the bins of one histogram to make it similar to the second histogram. Thus $CEMD(H_1, H_2) + CEMD(H_2, H_3)$ is the work needed to be done to rearrange H_1 to H_2, followed by a second rearrangement to change H_2 to H_3. Now, if this total work was indeed less than $CEMD(H_1, H_3)$,

that implies that $CEMD(H_1, H_3)$ is not the optimal solution that we expecting it to be, which violates our premise. Hence the proof. □

3.2 "Wormholes" and their implications

We have already seen that $CEMD(H_1, H_2) = 0$ does not necessarily implies that $H_1 = H_2$. This implies that the same point can exist at two different parts of the CEMD space, creating a "worm hole" in space. Thus, when you are growing a manifold through the space and ends up in one end of this worm hole, you would emerge through the other end of worm hole at a different corner of the space. This makes the CEMD space non-continuous thus making the space non-differentiable.

But analyzing CEMD in more detail shows that "worm holes" can be easily avoided by carefully choosing the signatures. Assume that we have two non-empty signatures H_1 and H_2 such that $H_1 \neq H_2$ and they from a worm hole. Ie, $CEMD(H_1, H_2) = 0$.

$$
\begin{aligned}
\Rightarrow \quad & \sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij} \cdot g_{ij} = 0 && \text{— from Eqn. 3.2} \\
\Rightarrow \quad & f_{ij} \cdot g_{ij} = 0, \forall i, j. && \text{— since } g_{ij}, f_{ij} \geq 0. \\
\Rightarrow \quad & \text{either } f_{ij} = 0 \text{ or } g_{ij} = 0, \forall i, j && \\
\Rightarrow \quad & \exists g_{ij} = 0 \text{ for some } i \neq j && \text{— since } H_1 \neq H_2 \\
\Rightarrow \quad & corr(S^i, S^j) = 1 && \text{— from the equation 3.1}
\end{aligned}
$$

This shows that $CEMD(H_1, S2) = 0$ only if there exists to bins S^i,S^j s.t they are completely correlated to each other. This implies that in such cases, any signature S, we can find a dual $\hat{S}$ with $CEMD(S, \hat{S}) = 0$, just by exchanging the contents of i^{th} bin of S with j^{th} bin, thus forming a pair of worm holes. That is, every point in the space would have a dual point. Thus, the j^{th}(or the i^{th}, if you prefer) dimension acts as a mirror, replicating the entire space and thus not carrying any extra information in it. Thus one can safely drop the j^th dimension completely, before the correlations are calculated, hence removing the possibility of fully correlated bins and worm holes.

Thus, if we discard all the the dimensions which are completely correlated to some other bins, we can be assured that even identity of indiscernibles holds for CEMD, making it a *metric*.

Sidenote: CEMD is effective only in the situations where all the data points between which you want to find distance, is given beforehand. This is because CEMD is based on correlation between the different dimensions of data points, and this correlation can potentially change when more and more new data points are added. This is not a major concern, because CEMD is intended for clustering purposes, where all the data points are available beforehand.

3.3 Computational complexities of CEMD

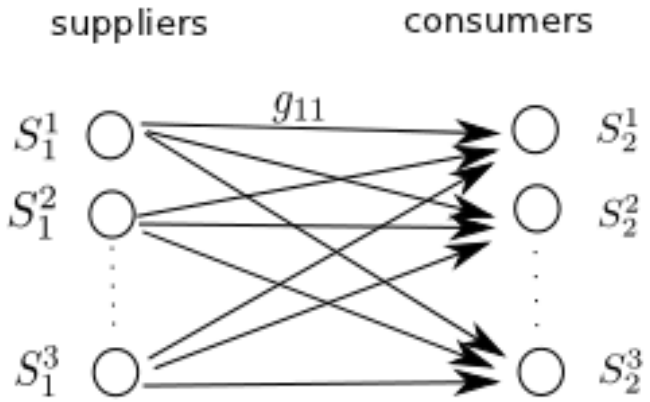


Figure 3.2: CEMD as transportation problem

Like any other earth mover's distance, CEMD can also be seen as a instance of transportation problem. This is a bipartite network flow problem which can be formalized as the Eqn. 3.2 and the constrains 3.3, 3.4, 3.5 and 3.6. If the aim is to find $CEMD(H_1, H_2)$, then bins of H_1 can be seen as the source nodes and its supply, while bins of H_2 can be taken as destination nodes and its demands. The cost of moving a unit of supply from one bin to another is showed by the edges between the supplier nodes and the consumer nodes. This is an *uncapacitated minimum cost flow problem*, which can be solved by Orlins algorithm((KV06) section 9.5) in $O(d^3 logd)$ time, where d is the number of bins.

3.3.1 Optimizing CEMD for manifold growing

As explained before, CEMD distance is a powerful measure to compare signatures to each other. But it come with the cost of increased time complexity of $O(d^3 logd)$ as compared to $O(d)$ complexity of calculating euclidean and cityblock distance.Thus if one wants to find k nearest neighbors of a point among n points in CEMD distance, then the running time of the algorithm would be $O(nd^3 log(d) log(k))$. This made it impractical for large datasets. For example,it was evident from simulations that it could take years for a program to calculate CEMD distance between the signatures of all the objects in the RGB-D dataset. Hence the complexity of calculating CEMD had to be brought down considerably for it to be tested on any big dataset.

We reduce this time completely by approximating CEMD in two phases.

As the first step, we introduce a greedy approximation algorithm by which the time complexity can be brought down to $O(d^2 logd)$. The assumptions are that maximum amount of points would be moved between the pair of bins with minimum ground distance and that the ground distance follows triangular inequality. We are fully aware that the correlation does not follow triangular inequality. But we presume that most of the time the correlation between the bins do follow triangular inequality. When they don't, say $g_{ik} \not< g_{ij} + g_{jk}$ for some bins i,j and k, we assume that g_{ik} is not too large than $g_{ij} + g_{jk}$. Thus even if one moves a point from the bin i directly to bin k, with out any intermediate bins, the calculated CEMD distance wont be too much off from the optimal value. This simplifies the problem from being a linear programming problem to a greedily solvable one.

To explain the new algorithm, lets view it as the transportation problem. The task is to find the minimum cost flow in the bipartite graph shown in figure 3.2. The bins of the two signatures H_1 and H_2 forms the sets of source nodes and sink nodes respectively. Contents of the signature H_1 are the *supply* at the source nodes. Similarly, contents of H_2 forms the *demand* at the sink nodes. Let E be the set of edges between the source and sink nodes. Each edge $e_i \in E$ has three parameters namely $e.source$, $e.destination$, and $e.cost$. Let d be the dimension of the signature. One can see that the flow through e_{min}, the edge with minimum cost, has to be maximum to get the minimum cost flow. Hence, the flow through the edge e_{min} has to be $min(H_1(e_{min}.source), H_2(e_{min}.destination))$. Once the edge e_{min} is satisfied, the edge

is removed, and the process is repeated with the edge with the next minimum cost, until supply at the source is exhausted or till the demand at the sink is fully met. Note that is could potentially compromise constraint 3.6. The complete steps are given in algorithm 3.

Algorithm 3 Calculate CEMD(H_1, H_2)

Initialize $f_{ij} = 0 \ \forall i, j \leq d$
Initialize $CEMD = 0$
Let $E = \{e_1, e_2, .., e_{d^2}\}$ be the sorted set of edges, based on the edge cost.
for $\forall e_k \in E$ **do**
    $i = e_k.source$
    $j = e_k.destination$
    if $S_1^i > 0$ AND $S_2^j > 0$ **then**
        $f_{ij} = min(S_1^i, S_2^j)$
        $CEMD = CEMD + f_{ij} \cdot e_k.cost$
        $S_1^i = S_1^i - f_{ij}$
        $S_2^j = S_2^j - f_{ij}$
    end if
end for
$CEMD = \frac{CEMD}{\sum_{i=1}^{d} \sum_{j=1}^{d} f_{ij}}$
return $CEMD$

However, the complexity of this algorithm can be brought down even further. Since *ground distance* g between the bins of the signature remains the same for all the points in a dataset, all the pairwise bin distance can be pre-sorted. With the sorted ground distance, the complexity of calculating CEMD between two signatures comes down to $O(d^2)$. Since there would be n^2 bin pairs, the complexity for pre-sorting them would be $O(n^2 log(n))$.

The second phase of reducing the complexity is specific fact that the CEMD would be used in manifold growing/clustering. In manifold growing, we are interested only in the CEMD distance between a point and its immediate k neighbors. Also, $L1$ distance acts as a reasonable measure for CEMD, since $L1$ distance is the same as CEMD, if the flow between two different bins are constrained to be zeros. ie, when $f_{ij} = 0 \forall i \neq j$,$CEMD(H_1, H_2) = L1(H_1, H_2)$. This is a reasonable approximation, since every bin is highly correlated to itself, making the inter-bin flow minimum. Hence, if one wants

to find k neighbors of a point v in CEMD distance, he can reasonably assume that all the k neighbors would be included in K neighbors of the same point v in $L1$ distance where K is reasonably bigger than k. Once these K neighbors are determined, the CEMD distance between v and K neighbors are calculated and the k points with minimum CEMD value are chosen as the required neighbors. If there are n points, then determining K nearest neighbors of a point in $L1$ distance can be done in $O(ndlog(K))$ time. Calculating the k nearest neighbors in CEMD among the already determined K points would take $O(Kd^2log(k))$ time. Thus the total complexity of this two level procedure for calculating CEMD is $O(ndlog(K)) + O(Kd^2log(k))$. Since $K << n$ and usually K is chosen as ck where c is some small constant, the resulting time complexity would be much lesser than the original $O(nd^3log(d)log(k))$.

The efficiency of CEMD over other distance measures are described in chapter 4. See table 4.3 and table 4.4.

4

Object class discovery by manifold clustering

4.1 Introduction

Object recognition has been one of the oldest problems in computer vision. Imparting vision to machines enabling them recognize objects around, makes it possible for them to interact with the world in a more human like way. There has been a whole lot of studies in object recognition using 2D images in the past two decades. Object features like HOG(DT05),SIFT(Low99) and SURF(BTVG06) were powerful enough to make the object detection algorithms (almost) comparable with human abilities of recognition. On the other hand, object recognition using 3D images is mostly an unexplored area. The reason is that,until recently, most of the 3D imaging technologies like range imaging, where intended for mapping large scale terrains and was much costlier for day today usage. But in the past five years, things have changed tremendously. New cheaper technologies like *Microsoft Kinect* were introduced which could capture color 3D images. These 3D images are generally referred as *pointclouds*, which is essentially a set of 3D points with a possible color value associated with each point.

One of the major challenges faced by traditional object recognition systems that uses 2D images for recognition, was that segmenting the object of interest from the background. This is extremely difficult, since in 2D imaging, the third dimension is effectively lost, making occlusion handling with out any prior information very painful. Even though the algorithms in 2D object recognition are pretty powerful, they com-

pletely rely upon the object segmentation algorithm. This problem can be easily solved with the new 3D imaging technologies. Since, along with a RGB value, a depth value would also be defined at every pixel in a 3D image, object segmentation can be done very easily. For example, appendix A shows how objects placed on top of a tabletop can be segmented out from a 3D pointcloud of the scene. The second advantage of 3D images in object recognition is that, besides the traditional 3D object features, one can define 3D object features like VFH(RBTH10) that captures the shape ans structure of the object. Combining these 2D and 3D image features gives object recognition a whole new perspective, and thats exactly what we did here.

4.2 Objective

Our objective is to employ the manifold based clustering algorithm that we presented in chapter 2, in unsupervised discovery of object classes from RGB-D dataset(LBRF11a)(described in section 1.2.1). The RGB-D dataset is an excellent testbed to test our algorithm, since images from each object from this dataset could lie on a one-dimensional circular manifold in a rightly chosen high dimensional space. The dataset consists of 3D pointclouds of 51 classes of common household objects. There are multiple object instances in every class. Each instance has 3 sequences of images captures from 3 different heights. We assume that these sequences of images, would form a one dimensional manifold, since consecutive frames in each sequence would be similar to each other.Refer figure 4.1. Our aim is to use our manifold based clustering algorithm and cluster these objects with out any prior information.

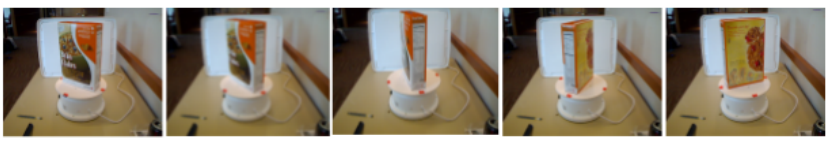


Figure 4.1: Multiple views of a cereal box. We hope that since consecutive frames are similar to each other, these views would form a one dimensional manifold.

4.3 Past attempts in RGBD object detection

As we mentioned before, 3D imaging is just coming into mainstream. Still some interesting works has been done in this area, which are discussed bellow.

One of the early work done in 3D object recognition was by Andrew E. Johnson et al. from NASA, in which they introduced the idea of *spin images* for matching the surface of two objects in question(JH99). This was basically a 3D registration algorithm, but based on the registration error, one could also tell if the two objects are the same or not. Oriented points, 3-D points with associated surface normals, are used to create spin-images. An oriented point defines a partial, object-centered, coordinate system. Two cylindrical coordinates can be defined with respect to an oriented point: the radial coordinate α, defined as the perpendicular distance to the line through the surface normal, and the elevation coordinate β, defined as the signed perpendicular distance to the tangent plane defined by vertex normal and position. Now, in order to find the spin image for a point say x_i, all other points whose surface normal makes an angle less than a threshold *support angle* with the surface normal of the point x_i, are chosen. All the chosen points are represented in the two cylindrical coordinates defined at the point x_i, and (α,β) values are calculated. Now these (α,β) values are binned into a 2D histogram, which in turn is the *spin image* of the point x_i. These spin images are calculated at every point on the object. Refer figure 4.2. If one wants to compare two 3D objects each other, he merely have to find an optimal matching between the points of the two objects based on their spin images.

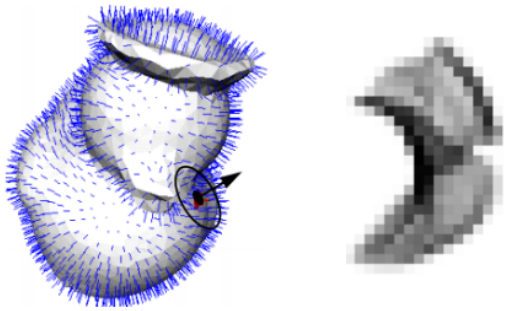


Figure 4.2: A 3D object and a spin image defined on a point on the object.(JH99)

Another major descriptor that was meant to detect 3D objects, was introduced by

Radu B Rusu, called Viewpoint Feature Histogram(VFH)(RBTH10). This captures both the shape and the view point of the 3D image, and is found to be very efficient in detecting objects in our own experiments. A detail discussion on VFH can be seen in section 4.4.1.1

In (LBRF11b), Kevin Lai et al. proposed a method for sparse Instance Distance Learning(IDL): instead of learning per-view distances, they defined a per-instance distance that combines all views of an object instance. The per-instance distances were learned jointly for all views of a particular object. The instance distance function between example x and the j^{th} instance of i^{th} category Y_{ij} was calculated as,

$$f_{ij}(x) = \frac{1}{|Y - ij|} \sum_{y \in Y_{ij}} W^T d(x, y) + b_{ij} \tag{4.1}$$

where W is a set of weight vectors w_y for all $y \in Y_{ij}$ and b_{ij} is a bias term. The weights W are learned from accuracies that each view brings into table on a supervised instance detection. Unlike the nearest instance classifier, this is significantly more expressive distance function since it allows the classifier to assign different weights to each feature and for each view, enabling it to adapt to the data.

Recently, two separate studies by Liefeng Bo et al.(BLRF11a) and Liefeng Bo et al.(BRF10), kernel descriptors were used to generate a rich visual feature set, which he used in detecting objects from the RGB-D dataset. These works presented principled framework to turn pixel attributes (gradient, color, local binary pattern, etc.) into compact patch-level features.

4.4 Object class discovery by manifold based clustering

As we discussed before, the RGB-D dataset is a perfect testbed to test our manifold based clustering algorithm. In order to employ any manifold clustering algorithm, one need to have two things to be defined. A high dimensional space where each data point can be represented, and a distance measure defined over this space. For the high dimensional space, the obvious choice would be the *image space* it self. But there are two main problems with using the images as such. The first problem is the very high dimensionality of the resulting data points. On average, a segmented image of a object in the RGB-D dataset would have around 5000 pixels. Each pixel has four fields defined, three colour components and a depth component. That makes the total dimension

20,000, making any classification algorithm on them very expensive. Even though this may not seem like a big number, this has a major computational implications in our case, because, to compare two points we would be using the CEMD distance we introduced earlier. Even after optimizing, the complexity of calculating CEMD between two points would be $O(d^2)$, where d is the dimensionality of the point(Refer 3.3.1). Thus the computation required increases quadratically with an increase in dimension. The second problem is that, since the size of the image changes from object to object, defining an consistent image space is very difficult. Hence, we decided to extract features from the images, that would have low,consistent, dimensionality. In this case, we define a 1040 dimensional feature vector for a 3D pointcloud. Each object would have this unique *object signature* and thus it defines 1040 dimensional *signature space* for the objects. We use CEMD defined in chapter 3 as the distance measure in this signature space.

4.4.1 Object signature

The most vital part of designing any recognition system is to come up with a "signature" for the objects, that is hopefully powerful enough to uniquely identify the objects in consideration. There are a lot of features defined for 2D images, including but not limited to PHOG(BZM07), SIFT(Low99) and SURF(BTVG06) features. But we are in an advantage here, since we also have the depth information along with the RGB data. To make sure we use all the information that we have in our disposal, we should come up with a signature that would capture the surface shape along with the color information. So the signature that we would be using to describe the object would be a combination three histograms, namely Viewpoint Feature Histogram (VFH)(RBTH10), PHOG, and a simple RGB color histogram.

4.4.1.1 Viewpoint Feature Histogram (VFH)

Viewpoint Feature Histogram (VFH)(RBTH10) is a descriptor for 3D point cloud data that encodes geometry and the viewpoint of the 3D shape. VFH was formulated as an extension of Point Feature Histogram(PFH)(RMBB08)(Rus09). PFH captures five different features of the pointcloud, namely,pairwise pan, tilt and yaw angles between *every* pair of normals on a surface, pairwise distance. As the first step, surface normals are calculated at each point in the cloud. In order to find the normal at a point p,

a principle component analysis is done on p and its k neighbors, and the normal is approximated as a unit vector in the direction of the eigenvector corresponding to the lowest eigenvalue. Now that we have normals defined at each point, construct a local Darboux frame(FLW89) coordinate system for each pair of points in the cloud. Let the pair of points under consideration be p_i and p_j and let n_i and n_j corresponding normals. Then the three coordinates of the Darboux frame for the pair of points are defined as follows.

$$\begin{aligned} u &= n_i \\ v &= (p_i - p_j) \times u \\ w &= u \times v \end{aligned}$$

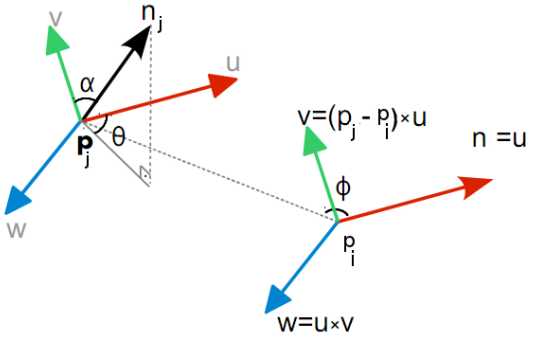


Figure 4.3: Relative angles between the surface normals of the two points in a Darboux coordinate system

Once this local Darboux frame is established for a pair of points, calculate the four features pan, tilt and yaw angles, and the Euclidean distance between the two points using the following formulae.

$$\begin{aligned} d &= ||p_i - p_j|| \\ \alpha &= v \cdot n_j \\ \phi &= u \cdot \frac{(p_j - p_i)}{d} \\ \theta &= arctan(w \cdot n_j, u \cdot n_j) \end{aligned}$$

Calculate the above four parameters for all the n^2 of points in the cloud, and bin them to 45 bins each. Now we have a 180 bin long histogram that represents the shape of the point cloud.

VFH has two major changes from PFH. In VFH, instead of defining a Darboux frame between *every* pair of points on the pointcloud, it is defined only between every point to the the point at the centroid of the pointcloud. Thus, the four parameters explained above, are calculated only once per every point, reducing the complexity from $O(n^2)$ to $O(n)$. As the second change, they added a "viewpoint" component to the histogram, which can capture the angle in which the pointcloud is being viewed. This is calculated by binning the angle that each point normal makes with the central viewpoint direction. These angles are binned to 128 bins. Note that the viewpoint component would be sensitive to the angle from which the object is viewed(hence the name), and hence it is useful in viewpoint identification.

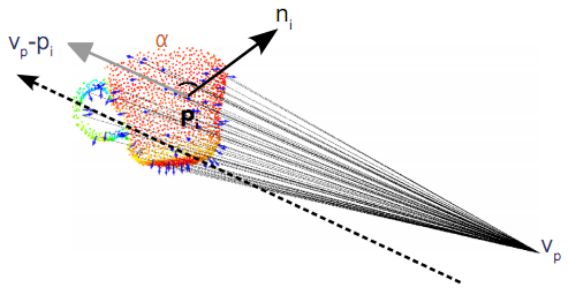


Figure 4.4: Viewpoint part of VFH is calculated by binning relative angles between point normals and central viewpoint direction

VFH histograms are not the only signatures available to capture the 3D shape of an object. Spin images described earlier also does the same thing. But VFH has a computational advantage over spin images. Given a oriented pointcloud with n points, VFH can be calculated in $O(n)$ time, while spin images would take $O(n^2)$ time time. Also, (RBTH10) shows a significant improvement of object detection accuracy when VFH is used over spin images.

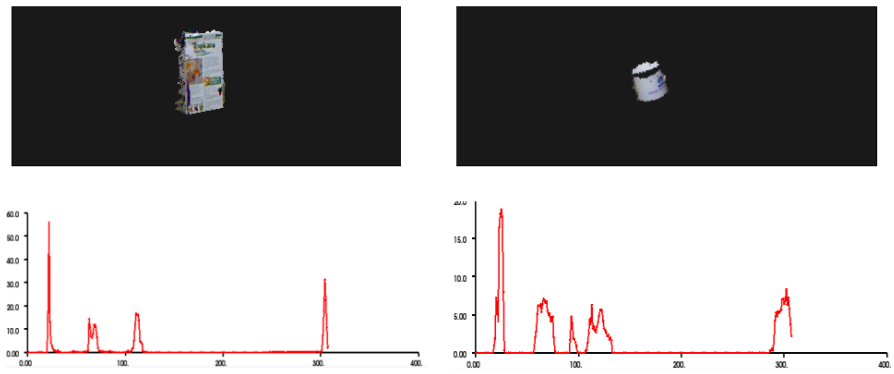


Figure 4.5: VFH signature of a segmented juice bottle and a coffee mug

4.4.1.2 RGB histogram

The second part of the signature is a simple RGB histogram, designed to capture the global distribution of the colors of an object. RGB values of each pixel is binned into one of the 192 bins reserved fro RGB histogram.

4.4.1.3 PHOG histogram

The RGB histograms are usually very weak, since they are not powerful enough to describe how the colors vary locally on the surface of an object. That is, a checker board would have the same RGB histogram as a square colored half black and the other half white. So we need a more powerful signature,that would capture the local color variation and the texture in the image. There are many popular image features like SURF and SIFT, which are local descriptor points, and global features like Histogram of gradients(DT05). Local descriptors like SURF and SIFT are not suitable for the problem at hand, because, in order to initiate a manifold growing algorithm, we have to define a fixed dimensional feature vector for every object to be clustered. But since number of keypoints could vary from object to object, and arranging these points into any particular order is almost impossible, defining a fixed length feature vector using SURF/SIFT is very difficult.

Hence we use a variant of HOG called Pyramid Histogram of Oriented gradients or PHOG(BZM07). In PHOG, local object appearance and shape of an object are characterized by the distribution of local intensity gradients or edge directions. This is implemented by dividing the image window into small spatial regions called cells,and

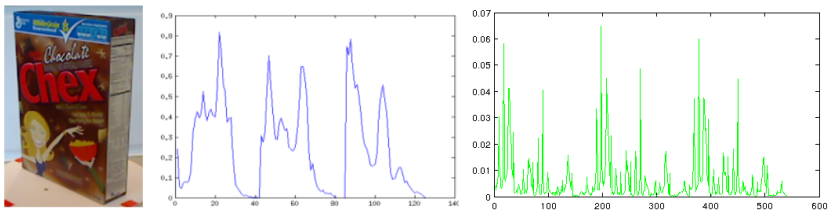


Figure 4.6: The color features: RGB histogram and a PHOG histogram of a cereal box

calculating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. The process is repeated on a pyramid of input image so that the histogram is scale-invariant. In our case, PHOG is calculated separately in the three color components of the image, resulting in a 540 dimensional histogram.

4.4.1.4 Efficiency of the new signature

To test how good is the new signature, used them for training an object class classifier on RGB-D dataset. There were 51 classes of objects in the dataset, and we randomly picked one instance for testing, reserving all the remaining instances of every class for training purposes. We trained just by using one of the three signatures mentioned above, as well as clubbing all three together. Following were the results obtained.

Table 4.1: Object class detection results using K nearest neighbor classifier

Features used	Accuracy
VFH	54.4%
RGB Histogram	34.93%
PHOG	57.90%
VHF+RGB+PHOG	70.68%

The tests showed that combining all the three histograms together resulted in a reasonably high detection rate of 70.68% with 51 object classes. The results obtained are quite satisfying, given the current state of the art accuracy for the RGB-D dataset is 84.1%(BLRF11b). Also it is worth noting that the 70.68% accuracy that we got, is just by using these signatures to train rather naive K nearest neighbor classifier. If one uses these signatures together with more sophisticated classifiers like random forests(Bre01), the accuracy could further improve considerably. But we move on, since we are more interested in the unsupervised learning of the object classes.

4.4.2 Results on RGB-D dataset

While growing the manifold, we used one randomly picked instance of the 51 classes of objects available in the dataset, together with all its three view video sequence. To reduce the complexity, four out of every consecutive 5 frames were skipped. Altogether, there were 7019 images, belonging to one of the 51 classes. The result of applying our algorithm is given in table 4.2, comparing with k means clustering. The parameters used were, $k_{min} = 1$, $k_{max} = 30$, $k_{patch} = 10$ and $\alpha = 0.5$. The efficinecy is quantized using two parameters namely purity and Normalized Mutual Information(NMI). To compute purity , each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned points and dividing by total number of points. Formally,

$$purity(\Omega, C) = \frac{1}{n} \sum_{k} argmax_j(|w_k \cap c_j|) \tag{4.2}$$

where,
$\Omega = \{w_1, w_2, .., w_k\}$ is the set of clusters and $C = \{c_1, c_2, .., c_j\}$ is the set of classes.
High purity is easy to achieve when the number of clusters is large. In particular, purity is 1 if each document gets its own cluster. Thus, we cannot use purity to trade off the quality of the clustering against the number of clusters. Normalized Mutual Information or NMI allows us to make this tradeoff. It is defined as,

$$NMI(\Omega, C) = \frac{I(\Omega, C)}{(H(\Omega) + H(C))/2} \tag{4.3}$$

I is the mutual information defined as,

$$I(\Omega, C) = \sum_{k} \sum_{j} P(w_k \cap c_j) \cdot log \frac{P(w_k \cap c_j)}{P(w_k)P(c_j)} \tag{4.4}$$

where $P(w_k)$, $P(c_j)$, and $P(w_k \cap c_j)$ are the probabilities of a point being in cluster w_k, class c_j, and in the intersection of w_k and c_j, respectively. H is the entropy defined as,

$$H(\Omega) = -\sum_{k} P(w_k) \cdot logP(w_k) \tag{4.5}$$

The confusion matrix is shown in figure 4.7, and the most confused pair of objects and the least confused objects are shown in figure 4.8 and figure 4.9 respectively.

Some of the discovered manifold clusters are showin in figures 4.10, 4.11, 4.12, and 4.13

Table 4.2: Object class discovery: a comparison between our algorithm and K means clustering

	K means clustering	Manifold growing
No of clusters	100	100
NMI	78.42%	90.30%
Purity	69.36%	82.60%

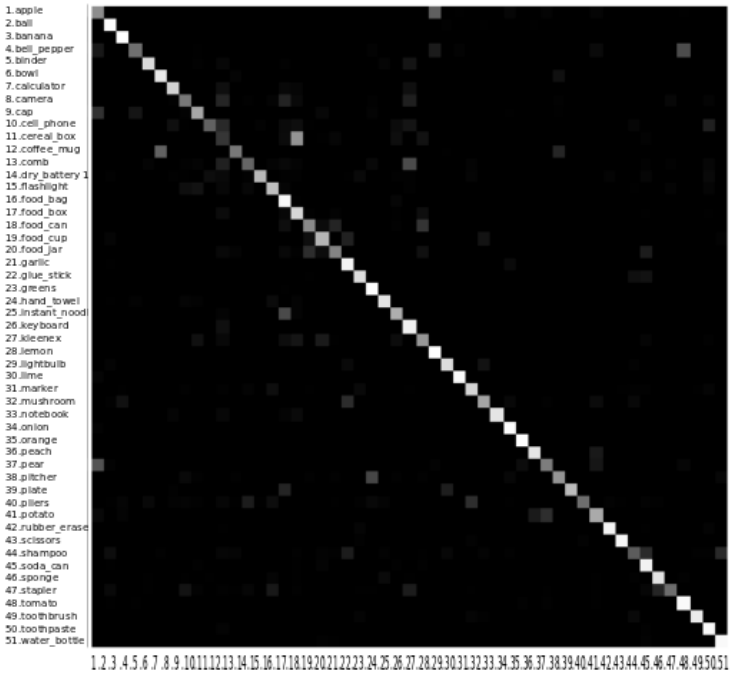


Figure 4.7: The confusion matrix between the objects discovered

4.4.2.1 Effect of dynamic branching factor and CEMD distance measure

Dynamic branching factor and a new CEMD distance measure were two of our major contributions to manifold separating. Thus its only fair to study how the purity would vary with a static branching factor and with a different distance measure than CEMD. But this time we chose only one video sequence for a randomly chosen instance from every class. This was mainly to save the computation time, but also to see how the purity would change from those reported in table 4.2 where we used all the three video sequence while clustering. The results are given in tables 4.3 and 4.4.

As it can be seen from the tables, dynamic branching and CEMD *always* improved

Figure 4.8: The most confused pair of objects

Table 4.3: Results with dynamic branching turned off

Distance metric	$branching factor$	Purity
Euclidean	2	72.26%
Cityblocks	2	86.87%
CEMD	2	92.27%

the performance of the algorithm. The high accuracy compared to the table 4.2 can be attributed to the fact that only one video sequence from the three available was used in manifold growing.

4.4.2.2 Comparison between supervised Vs unsupervised learning

In supervised learning, the algorithm would be learning prior information about data, and how to map the given data point to the corresponding label. This generally done by analyzing data points from a training set along with its corresponding labels. Then these learned priors would be used to predict the class of any newly fed data point. However, in unsupervised learning, we do not have this luxury. Generally, a clustering

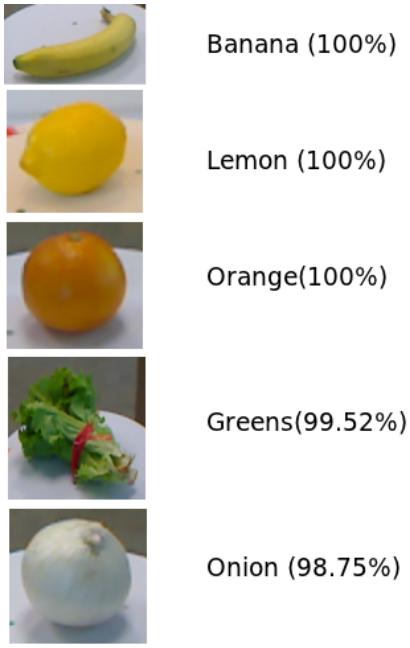


Figure 4.9: The least confused objects

Table 4.4: Results with dynamic branching turned on($k_{min} = 1, k_{patch} = 10, \alpha = 0.5$)

Distance metric	k_{max}	Purity
Euclidean	27	91.32%
Cityblocks	24	98.47%
CEMD	21	98.70%

algorithm is ran on the training dataset that would hopefully identify clusters in data that are significant for the problem at hand. These clusters are labeled according to their plurality, ie the label that is repeated the most in the cluster. Then a classifier is trained based on these learned labels, and is tested on the testing set. Thus, such an unsupervised learning algorithm can never be as accurate as a supervised algorithm, since the labels of learning set used in unsupervised learning would always be just an approximation for the actual labels. But it is interesting to see how much the accuracy falls when one moves from supervised learning to unsupervised one. The closer the accuracy of unsupervised learning to supervised learning, more efficient is the unsupervised algorithm. Note that comparing our results with a supervised method

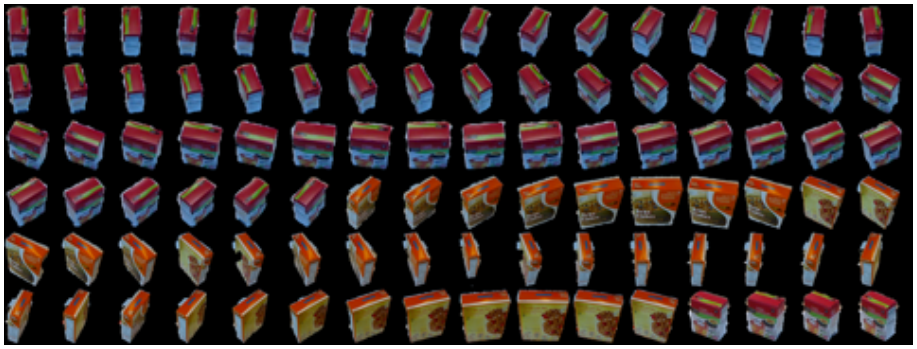


Figure 4.10: One of the manifold discovered. It is mainly composed of boxes.As expected, viewpoint order is preserved in the discovered manifold. There are different types of boxes involved, namely cereal boxes, food boxes etc

requires a little of supervision. That is, we would be needing the label for one data point per the cluster formed, so as to label the cluster with the label of its plurality. Thus, we are reducing the need to label from ALL the input set to only 55.We did two experiments, namely object viewpoint detection and object class detection, and compared the accuracies that we got with our unsupervised manifold learning algorithm with a supervised K nearest neighbor classifier accuracies. The parameters used were, $k_{min} = 1$, $k_{max} = 30$, $k_{patch} = 10$ and $\alpha = 0.5$

Viewpoint independent object detection: In viewpoint independent object detection, we tested how well a classifier can identify objects from a different viewpoint than the ones on which the classifier was trained on. So, images from the same object were used for training and testing, but it was made sure that the testing images were from a different viewpoint than the images on which the classifier was trained on. For every instance of the objects from the first 10 classes, one video sequence was reserved for testing, while the remaining two video sequence were used for training.

A supervised k nearest neighbor classifier gave an accuracy of **99.3%**. The results of using our algorithm for unsupervised learning, are as follows

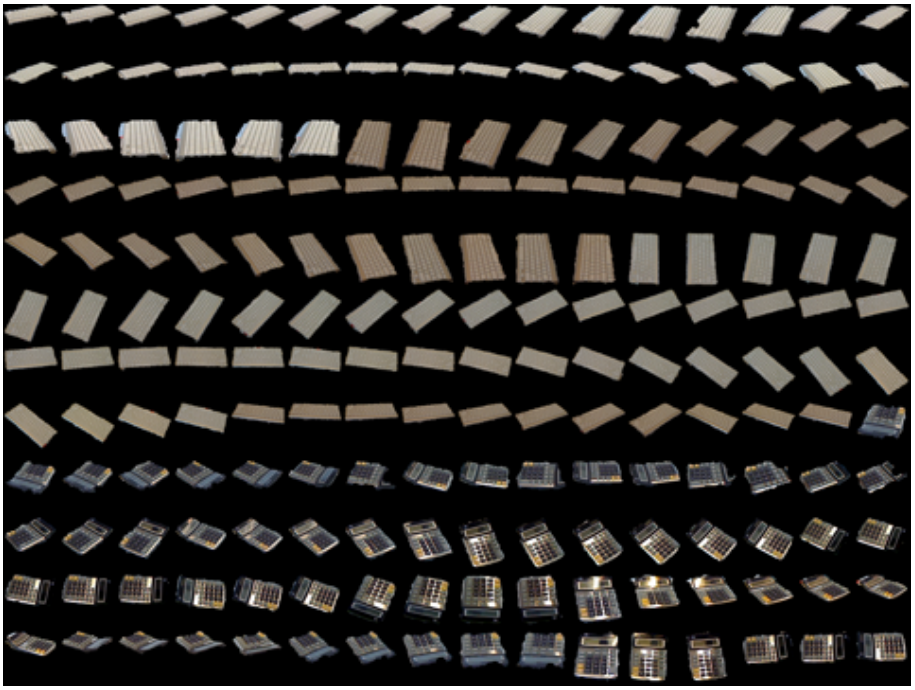


Figure 4.11: This manifold mainly consists of keyboards.But by the end of the manifold contains some samples of calculator as well. This is not a surprise since calculators look a whole lot like keyboards.

Unsupervised learning by manifold growing	
Training	
No of clusters	19
NMI	1.00
Purity	97.03%
Testing	
KNN accuracy(Unsupervised)	93.19%

As one can see, accuracy only dropped by 6% when our algorithm is used for unsupervised learning, which is a very much acceptable margin.

Object class detection: Object class detection is a little more tougher than viewpoint independent object detection, since one would be trying to determine the classes of previously unseen instances of objects.There were 51 classes of objects. Every fifth image from *all* the three video sequence of a one randomly chosen instance of each class were reserved for the testing phase, while every fifth image from the remaining instances were used for training. were chosen for testing. This made sure that, no

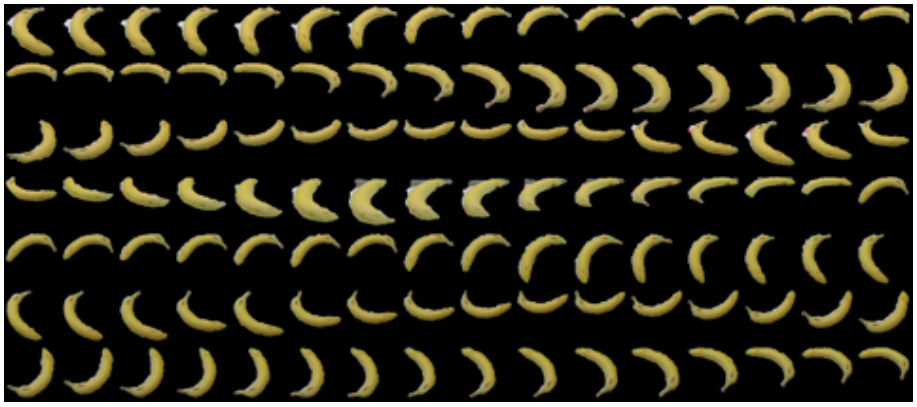


Figure 4.12: Banana manifold. consistent shape and color of bananas resulted in a pure manifold.

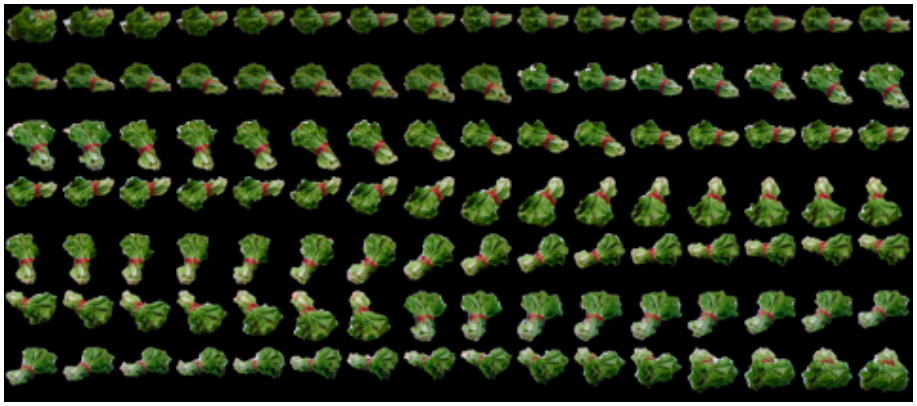


Figure 4.13: Mushroom manifold. Unique shape and consistent green color resulted in a strong, pure manifold

image of the objects tested were included in the training phase. As earlier, we trained a supervised KNN classifier and compared the results with the results of unsupervised classification using our manifold growing algorithm.

The supervised KNN classifier gave an accuracy of **74.04%**. The results of using our algorithm for unsupervised learning, are given bellow.

Unsupervised learning by manifold growing	
Training	
No of clusters	55
NMI	0.952
Purity	91.65%
Testing	
KNN accuracy(Unsupervised)	67.24%

Accuracy has dropped from 74.04% to 67.26% when unsupervised manifold growing algorithm was used in place of supervised KNN.

4.5 Conclusion

The attempt in this thesis was towards unsupervised discovery and not to improve the supervised results per se. Thus, we used our manifold based clustering algorithm in a unsupervised object discovery. Methods such as random forests etc which can be used only in supervised situations were not explored very deeply. Unsupervised learning using our algorithm gave results comparable with standard supervised learning techniques(4.4.2.2). The major contribution to the accuracy came from the CEMD distance measure we introduced, and from the dynamic branching factor of the manifold growing algorithm. As seen from the tables 4.3 and 4.4, there was a consistent improvement in the accuracies with dynamic branching factor turned on, across all the distance measures. It can be also seen that, the usage of CEMD as the distance measure gives better results than other distance measures we tested our algorithm on.

One potential demerit of our clustering algorithm is the high computational time it demands. Most of the computational overhead comes from the usage of CEMD as a distance measure. The complexity of the greedy algorithm3 to calculate CEMD between two points of dimension d is $O(d^2)$ as opposed to $O(d)$ complexity of calculating regular Euclidean distance. This is the major bottleneck in our algorithm, because, since our feature vector has the dimensionality around 1000, making CEMD the calculation 1000 times slower than the time required for Euclidean distance. Feature extraction, CEMD calculation, and running the clustering algorithm on 7019 data points took around 18 hours in a Intel i7 quad core processor with 8GB of RAM. It should be noted that the most of the computation comes from the calculation of CEMD. The clustering algorithm can use any distance measure; and if it had used usual distance matrices like

Euclidean distance, its computational complexity would have been comparable with any popular clustering algorithms like k means clustering.

5

Conclusion

In this thesis, we introduced a new spatial clustering algorithm that was inspired from gestalt perception, and used it in discovering object object categories from a RGB-D dataset. Given a set of points sampled from a space with several(possibly interesting) manifolds, the algorithm attempts to cluster the points into the manifolds they belongs to. Our algorithm does the job with minimal knowledge of the shape and other parameters of the manifolds involved. A major advantage of this algorithm over most of the other similar algorithms is that, one need not have to explicitly mention the total number of manifolds involved and the dimensionality of each manifold. Also, algorithm is designed to be tolerent to non-uniform sampling on manifold, by using dynamic neighborhood assignment. Since the algorithm resembles some of the the gestalt principles of grouping, the results are same for the human grouping for the same data.

We also introduced a distance metric called Correlation based Earth Mover's Distance or CEMD. CEMD is a derivative of the standard Earth Mover's Distance, with bin distances defined in terms of the correlation between histogram bins. Bin distance between two bins i and j is defined as $1 - correlation(i, j)$. This implied that, while calculating the CEMD, when the contents of a bin are moved, they are moved to statistically similar bins. Use of CEMD significantly improved performance of manifold clustering, over other distance measures like euclidean and cityblock distances in manifold clustering.

As an interesting application of the newly proposed algorithm, we used it in unsupervised discovery of object classes from a 3D object dataset of household objects. The dataset that we chose was the Kinect RGB-D dataset. The images were captured

using Kinect, an extra depth information was also available, which better defined the features of objects. The results of the unsupervised approach were impressive, to the standard supervised learning methods. We concluded that the new manifold clustering algorithm together with the new CEMD distance measure, is very efficient in clustering data points that happen to be distributed along some manifold structure, rather than as a spherical blob.

5.1 Plausible future works

As of now, our algorithm is meant only for clustering of data points from high dimensional datasets. Unlike algorithms like isomaps, it does not maps the clustered points onto their intrinsic lower dimensional space. But this is not too difficult to implement, since our algorithm already creates a graph along the manifold surfaces. Now, to implement an isomap like approach (TDSL00), one can easily use this graph to calculate geodesic distance along the manifold. Once the geodesic distance between every points is calculated, one can use dimensionality reduction techniques like MDS(Kru64) to map the points to the respective lower dimensional space.

An improvement of the manifold clustering algorithm could be, eliminating the need of k_{min}, the parameter that defines the nominal branching factor k while growing the graph. Minimum branching k_{min} is usually set according to the nature of the manifold that one is trying to cluster. Hence it can potentially be expressed as a function of the local dimensionality of the manifold. Intuitively, a larger k has to be used if the local dimensionality is very high, and a k can be used if the calculated local dimensionality is very small. This not only removes the need of predefining k_{min}, but also make it dynamic, based on the local structure of the manifold. A potential demerit of this could be the algorithm being more susceptible to the noise.

One of the issues with the current algorithm is that, it assumes that every point that has to be clustered belongs to a single manifold. Even though this can be a useful assumption in some clustering scenarios(For example, in the object class discovery experiment, one may not like an apple to be labeled as both apple and tomato), it may not be that useful in other scenarios(As in figure 1.4, where the central face clearly belongs to both the manifolds). But if one desires, this issue can be fixed quite easily, if it is suitably defined by the user. One just have to relax the manifold growing algorithm

so that it can even grow to the points that have been already identified as the part of some other manifold. The patch to the algorithm 2 to accommodate this change is given bellow.

Change line no 14:

$candidate\_neighbors = \{x_i \in candidate\_neighbors\ s.t\ visited[i] = 0$

to

$candidate\_neighbors = \{x_i \in candidate\_neighbors\ s.t\ visited[i] = 0$ OR $label[i] < manifold\_id$

Appendix A

3D object segmentation

One of the main challenge in visual perception is to extract the objects of interest from the background clutter. Compared to a 2D image, extracting objects from a 3D point cloud data is relatively simple, since we can make use of the one extra dimension. On the other hand, Kinect scans are very noisy, since its sole purpose was to use along with an gaming console. Nevertheless, the simple algorithm outlined in Fig. A.1 was able to extract objects from a relatively less cluttered environment. We assumed that

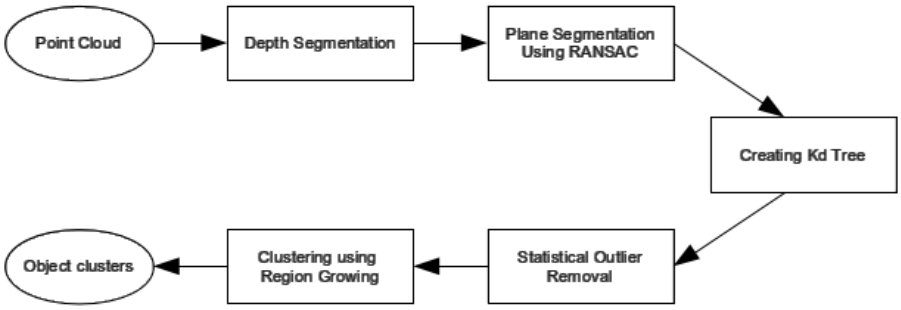


Figure A.1: Object segmentation algorithm outline

the object of interest would be always placed over a planar surface, like floor or a tabletop. The first step is to segment out all the background clutter that fell beyond a threshold distance. This can be easily be done since we know the depth of every point in the pontcloud. After this step, hopefully, we would be left with a planar surface with objects of our interest on top. This planar surface can be identified using the RANSAC algorithm(FB81), which probabilistically determines the parameters of the plane. Once we have the equation of the plane, the points that are on and beneath the

plane were filtered out. A K-d tree(Ben75) was used for the 3D spatial decomposition, so that nearest neighbors of a point can be calculated in $O(logn)$, were n is the number of points in consideration. With the help of the K-d tree, a statistical outlier removal algorithm was ran on the point cloud, which filtered out some of the noises from the data. Now we are left with point clusters, hopefully belonging to the objects of our interest. A region growing algorithm was initiated from random points from the cloud, and with an apt threshold, all the objects in the scene were segmented out. Even though this algorithm is a very crude one, it is efficient enough to segment objects from a controlled environment.

Figure A.2: The input pointcloud, and result after filtering the background and the tabletop

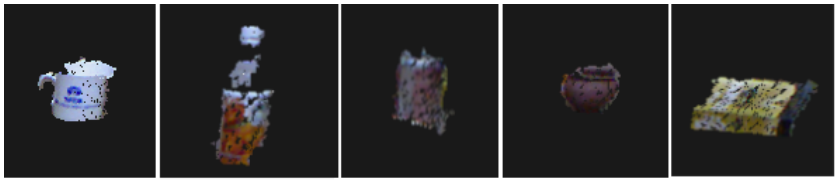


Figure A.3: The final segmented objects after region growing

References

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 66

[BLRF11a] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1729–1736. IEEE, 2011. 46

[BLRF11b] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1729–1736. IEEE, 2011. 51

[BM05] Y. Bengio and M. Monperrus. Non-local manifold tangent learning. *Advances in Neural Information Processing Systems*, 17:129–136, 2005. 7

[Bre01] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 51

[BRF10] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. *Advances in Neural Information Processing Systems*, 2010. 46

[BTVG06] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006. 43, 47

[BZM07] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 401–408. ACM, 2007. 47, 50

[CCM+07] S.K. Chalup, R. Clement, J. Marshall, C. Tucker, and M.J. Ostwald. Representations of streetscape perceptions through manifold learning in the space of hough arrays. In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*, pages 362–369. IEEE, 2007. 4

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. Ieee, 2005. 43, 50

[Dub11] Nandan Dubey. Discovering intersecting smooth manifolds via pursuit algorithm. *M Tech thesis*, 2011. vii, 4, 6, 8, 18, 20, 26

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 65

[FLW89] FP Ferrie, J. Lagarde, and P. Whaite. Darboux frames, snakes, and super-quadrics: Geometry from the bottom-up. In *Interpretation of 3D Scenes, 1989.*

Proceedings., Workshop on, pages 170–176. IEEE, 1989. 48

[Gal09] K.M. Galotti. *Cognitive psychology: In and out of the laboratory*. Cengage Learning, 2009. 25

[GM11] M. Gashler and T. Martinez. Tangent space guided intelligent neighbor finding. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2617–2624. IEEE, 2011. 11, 26

[Goo11] Google. Google goggles official webpage, 2011. 12

[GVL96] G.H. Golub and C.F. Van Loan. *Matrix computations*, volume 3. Johns Hopkins Univ Pr, 1996. 24

[HKO01] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 26. Wiley-interscience, 2001. 4

[JH99] A.E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449, 1999. 45

[Jol02] Jolliffe. *Principal component analysis*, volume 2. Wiley Online Library, 2002. 4, 18, 19

[Kru64] J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. 62

[KV06] B.H. Korte and J. Vygen. *Combinatorial optimization: theory and algorithms*, volume 21. Springer Verlag, 2006. 35, 38

[LBRF11a] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011. 12, 44

[LBRF11b] K. Lai, L. Bo, X. Ren, and D. Fox. Sparse distance learning for object recognition combining rgb and depth information. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4007–4013. IEEE, 2011. 46

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999. 43, 47

[LV07] J.A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, 2007. 5

[Mar74] D. Marr. The computation of lightness by the primate retina. *Vision Research*, 14(12):1377–1388, 1974. 12

[Mic10] Microsoft. Kinect official webpage, November 2010. 1

[Pet77] A.M. Peters. Language learning strategies: does the whole equal the sum of the parts? *Language*, pages 560–573, 1977. 25

[RBTH10] R.B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010. 44, 46, 47, 49

[RMBB08] R.B. Rusu, Z.C. Marton, N. Blodow, and M. Beetz. Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 643–650. IEEE, 2008. 47

[RS00] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. 6, 15

[Rus09] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, October 2009. 47

[SB06] L. Sigal and M.J. Black. Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion. *Brown Univertsity TR*, 120, 2006. 29

[SMM07] R.J. Sternberg, J. Mio, and J.S. Mio. *Cognitive psychology*. Wadsworth Pub Co, 2007. 25

[SP05] R. Souvenir and R. Pless. Manifold clustering. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 648–653. Ieee, 2005. 7

[TDSL00] J.B. Tenenbaum, V. De Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. 5, 15, 26, 62