2-1-2003

A hybrid method for solving the non-rigid point matching problem in 3D

Nahla El Said
The American University in Cairo AUC

The American University in Cairo
School of Sciences and Engineering

A Hybrid Method for Solving the Non-Rigid Point Matching Problem in 3D

A Thesis Submitted to

Computer Science Department

In partial fulfillment of the requirements for
the degree of Master of Science

By
Nahla ElSaid

B.Sc. of Engineering

under the supervision of

Prof. Amr Goneid
Dr. Ahmed Sameh

December 2002

2002/83

The American University in Cairo
School of Sciences and Engineering

A Hybrid Method for Solving the Non-Rigid Point Matching Problem in 3D

A Thesis Submitted by *Nahla Elsaid* to
The Department of Computer Science
December 2002
In partial fulfillment of the requirements for
The degree of Master of Science
has been approved by

Prof. Dr. Amr Goneid
Thesis Committee Chair/Adviser [redacted]
Affiliation AUC

Dr. Ahmed Sameh
Thesis Committee Chair/Adviser [redacted]
Affiliation AUC

Dr. Hoda Hosny
Thesis Committee Reader/Examiner [redacted]
Affiliation AUC

Dr. Sherif El-Kassas
Thesis Committee Reader/Examiner [redacted]
Affiliation AUC

Prof. Dr. Abdel-Badie Salem
Thesis Committee Reader/Examiner [redacted]
Affiliation [redacted]

[redacted] Dec. 31, 02
Department Chair/ Program Director **Date**

[redacted] Jan 8, 2003
Dean **Date**

ACKNOWLEDGMENTS

There are many people who helped me obtain my Master degree. First and foremost, I would like to thank my thesis advisors, Prof. Dr. Amr Goneid, and Dr. Ahmed Sameh who guided me through this work, and with their advice they kept me on track.

During my thesis work, my committee members provided much advice on many topics and their comments on this document greatly improved its final form. In particular Dr. Hoda Hosny and Dr. Sherif El-Kassas.

Finally, I would like to thank my family. My parents have always been a source of constant support and encouragement. From them I developed for knowledge and the desire to work hard to achieve my goals. My last and final thanks go to my fiancé, Mohamed. Without his support I would have had many more stressful moments.

ABSTRACT

This thesis introduces a hybrid method for solving the non-rigid point matching problem in 3D. The method is based on the ICP (Iterative Closest Point) algorithm, which starting from an initial configuration of two non-rigid point sets, iteratively finds their best correspondence. The present method incorporates a non-rigid transformation parameterization, the "Thin-Plate Spline" or TPS as a prior smoothing method.

The convergence of the ICP algorithm towards the global minimum is known to depend largely on the initial configuration of the data and model point sets. The present method also introduces the center of mass computation as a preprocessing step to prevent the algorithm from being trapped in a local minimum and miss the optimum matching.

The performance of the present method, which is represented by the root mean square (RMS) error versus number of iterations, has been tested using synthetic data corresponding to three different templates. The results were obtained using the present method with and without the center of gravity correction, and compared with those obtained using the Robust Point Matching algorithm (RPM). It is found that the present method prevents the ICP from being trapped in a local minimum. Also it made the ICP algorithm work more efficiently with respect to accuracy after a few iterations (i.e. 5-15 iterations), however the RPM algorithm did not produce any reasonable results before a relatively large number of iterations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Chapter 2

Chapter 3

Appendix A

CHAPTER 1

Introduction

Non-rigid point matching is an important problem, as non-rigid transformations are needed for image registration tasks of deformable objects. Analyzing non rigid motion has become a major research problem not only in computer vision and image processing, but also more dominantly in medical imaging, where most of the objects being studied are deformable, such as biological shapes which change non-rigidly- finger bends, heart beats…etc.

A basic non-rigid point matching problem can be defined as follows: given two sets of points (essentially their coordinates), we would like to find the *non-rigid transformation* that best maps one set of points onto the other and/or the set of *correspondence* between the points.

In the few existing literature finidings on the non-rigid matching, most authors are not looking at minimizing a criterion, but their attention was rather focused on the type of smoothing or physical model of deformation to be used. In this thesis we propose to put the non-rigid matching problem into a criterion minimization framework and to use ICP (Iterative Closest Point) as the basic algorithm to be adapted to non-rigid matching.
Before launching into a detailed description of non-rigid point matching, we briefly describe the fundamental concepts in the domain of image registration.

1.1 Image Registration

Registration [9] is a fundamental task in image processing used to match two or more pictures taken, for example at different viewpoints.
To register two images, a transformation (spatial mapping) must be found so that each location in one image can be mapped to a new location in the second. This mapping should "optimally" align the two images wherein the optimality criterion itself depends on the actual structures/ objects in the two images that are required to be matched.

The need for image registration arises in many practical problems in diverse fields. As pointed out in [9], registration is often necessary for:

- Integrating information from different images
- Finding changes in images taken at different times or under different conditions.
- Inferring three-dimensional information from images in which either the camera or the objects in the scene have moved, and
- For model-based object recognition.

Image registration has a significant impact on a wide range of various research fields such as:

- Computer vision and pattern recognition
- Medical image analysis
- Remotely sensed data processing

Those research fields will be introduced separately so that they may be used as examples that give a complete description of the registration task.

1.1.1 Computer Vision and Pattern Recognition – for numerous different tasks such as segmentation, object recognition, shape reconstruction, motion tracking, stereo- mapping, and character recognition.

One goal of computer vision research is to give computers humanlike visual capabilities so that machines can sense the environment in their field of view, understand what is being sensed, and take appropriate actions as programmed. Vision involves the physical elements of illumination, geometry, reflectivity, and image formation as well as intelligence aspects of recognition and understanding.

As an example of a computer vision problem is: Stereomapping to recover depth or shape from disparities. Examples of Pattern Recognition are character recognition, and signature verification.

As this thesis' final goal is to solve the non-rigid point matching in 3D, so we are interested in three-dimensional object recognition [5].

❑ Mathematical Problem Formulation [5]

It is often beneficial to define a problem in a stricter mathematical form to eliminate possible problem ambiguities. For example, we have not yet discussed how a system should respond if several distinct objects appear to be identical from a given viewpoint.

Each object occupies space, and at most one object can occupy any given point in space. It is necessary to describe the spatial relationships between each object and the rest of the world. One way to describe spatial relationships is through the use of coordinate systems. For reference purposes, we assume the existence of a world coordinate system that is placed at any convenient location. Objects are positioned in space relative to this coordinate system by means of translation and rotation parameters. We refer to the translation parameters of an object as the vector α and to the rotation parameters of an object as the vector θ. The number of parameters for each vector depends on the *dimension* of the depth-map recognition problem. For example, the 2-D problem requires a total of three parameters. For the 3-D case, we write the necessary six parameters as follows:

$\alpha = (\alpha, \beta, \gamma)$ and $\theta = (\theta, \phi, \Psi)$.

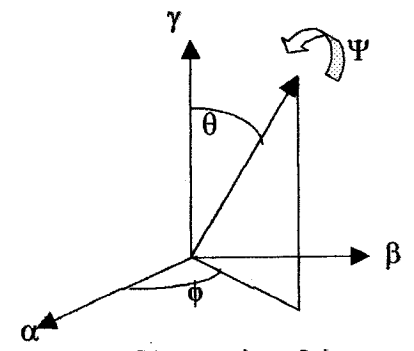

Figure 1.1 Rigid objects in 3-D space have 6 degrees of freedom. Translation = $\alpha = (\alpha, \beta, \gamma)$; rotation and $\theta = (\theta, \phi, \Psi)$

Figure 1.1 explains the meaning of these parameters. We define our world model W as a set of ordered triples (object, translation, rotation):

$$W = ((A_i, \alpha_i, \theta_i))$$

We consider object A_0 to be the sensor object with position , α_0 and orientation θ_0.

If a time-varying world model is required, all objects and their parameters can be functions of time.

- **Object Representation**

The object representations commonly used by contemporary computer-aided-design (CAD) geometric solid-object-modeling systems are categorized as one of the following:

(1) *Wire-frame representation*. A wire-frame representation of a 3-D object consists of a 3-D vertex point list and an edge list of vertex pairs, or it can be formatted as such. This representation became quite common owing to its simplicity. Although fast wire-frame displays are still popular, solid modeling has replaced wire-frame modeling because a wire frame is an ambiguous representation for determining such quantities as the surface area and volume of an object. Figure 1.2 shows an example of a wire-frame model.

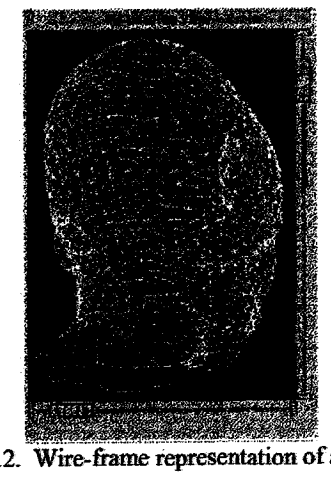

Figure 1.2. Wire-frame representation of a head

(2) *Constructive solid geometry representation* (CSG). The CSG representation of an object is specified in terms of a set of 3-D volumetric primitives (blocks, cylinders, cones, and spheres are typical examples) and a set of Boolean operators: union, intersection, and difference. Figure 1.3(c) is an example of a CSG description of an object. The storage data structure is a binary tree, where the terminal nodes are instances of primitives and the branching nodes represent Boolean set operations and positioning information. CSG trees define object volume and surface area unambiguously and are capable of representing complex objects with a very small amount of data.

However, the boundary evaluation algorithms required to obtain usable surface information are computationally intensive.

Also, general sculptured surfaces, such as the human face surface shown in Figure 1.4, are not easily represented using CSG modelers. A general-purpose modeling system must be able to represent such surfaces.

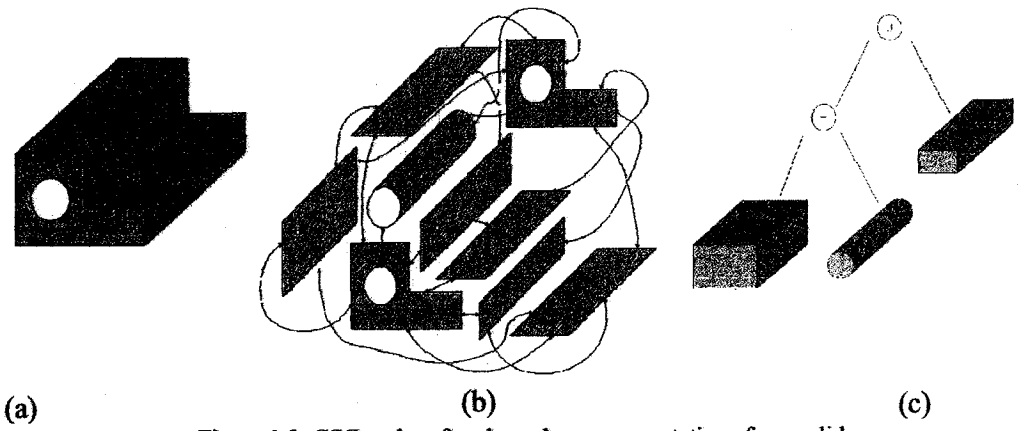

Figure 1.3. CSG and surface boundary representations for a solid
(a) Solid ; (b) a boundary representation ; (c) a CSG Representation

(3) Surface boundary representation (B-Rep). Surface boundary representations define a solid object by defining the 3-D surfaces that bound the object. Figure 1.3(b) shows an example of the boundary representation concept. The simplest boundary representation is the triangle-faced polyhedron, which can be stored as a list of 3-D triangles. Arbitrary surfaces are approximated to any desired degree of accuracy by utilizing many triangles. The human face surface shown in Figure 1.4 was displayed from a list of triangles and quadrilaterals that approximate a smooth surface (which is stored using a rational B-spline surface representation).

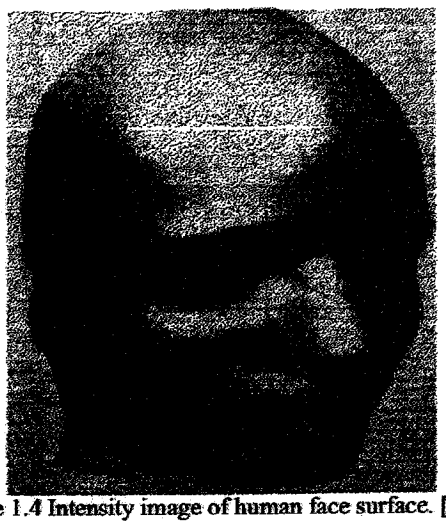

Figure 1.4 Intensity image of human face surface. [38]

A slightly more compact representation allows the replacement of adjacent, connected, coplanar triangles with arbitrary n-sided planar polygons. This type of representation is popular because model surface area and volume are well defined, and all object operations are carried out using piecewise-planar algorithms.

(4) *Spatial-occupancy representation*. Spatial-occupancy representations use nonoverlapping subregions of the 3-D space occupied by an object to define that object. This method unambiguously defines an object's volume. The following single-primitive representations of this type are commonly used:

(a) Voxel Representation.

Voxels are small-volume elements of discretized 3-D space. They are usually fixed-size cubes. Consider the vertebra body shown in Figure 1.5 which can be represented by the list of voxels occupied by the object as shown in Figure 1.6. This representation is memory intensive, but algorithms using it tend to be very simple.

Figure 1.5 Intensity image of a vertebral body

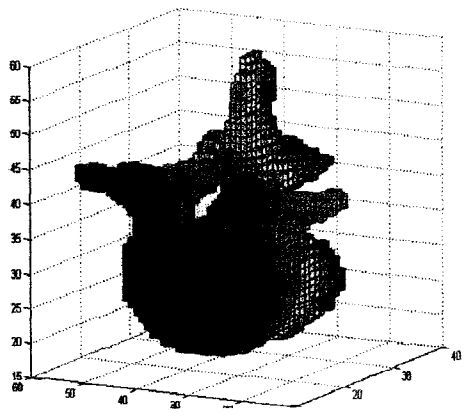

Figure 1.6 Voxels volume Rep

(b) *Octree representation* . An octree is a hierarchical representation of spatial occupancy. Volumes are decomposed into cubes of different sizes as shown in Figure 1.7, where the cube size depends on the distance from the root node.

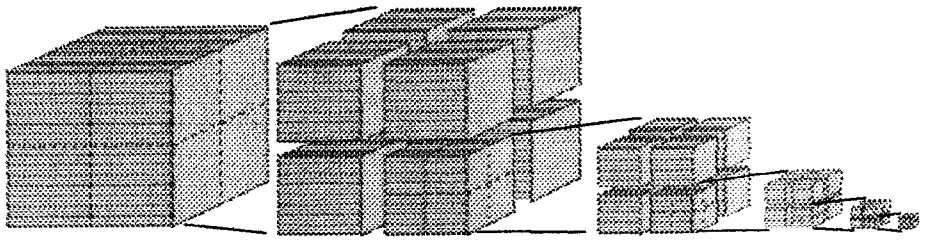

Figure 1.7 Cubes Representing Octree

Each branching node of the tree represents a cube and points to eight other nodes that describe object volume occupancy in the corresponding octant subcubes of the branching node cube. This representation offers the advantages of the voxel description but is more compact. Its compactness requires more complicated algorithms for many computations. The basic idea of octrees is displayed by considering the 2-D analog of octrees (usually referred to as quadtrees). Figure 1.8 shows a simple example of the octree representation.

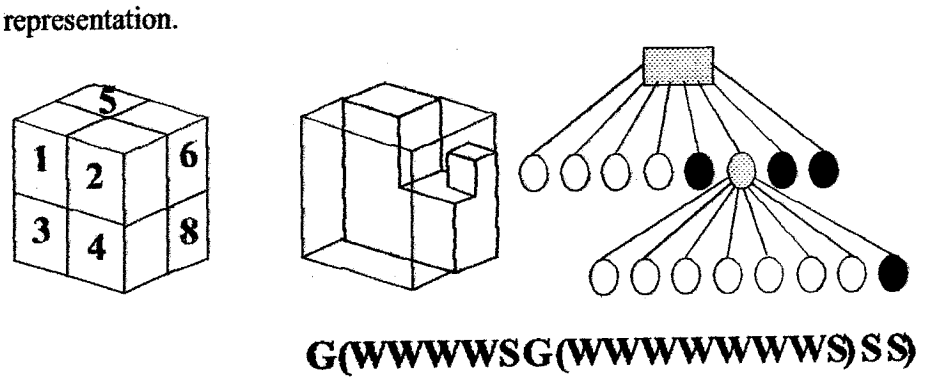

Figure 1.8 Octree Simple Example

(c) *Tetrahedral cell decomposition representation*. Decomposition of 3-D space regions into tetrahedral elements is very similar to the lower dimensional analog of decomposing flat surfaces into triangles as shown in Figure 1.9. (The tetrahedron is a 3-simplex, whereas the triangle is a 2-simplex.) Tetrahedral decompositions define volume and surface area unambiguously and are useful for mathematical purposes.

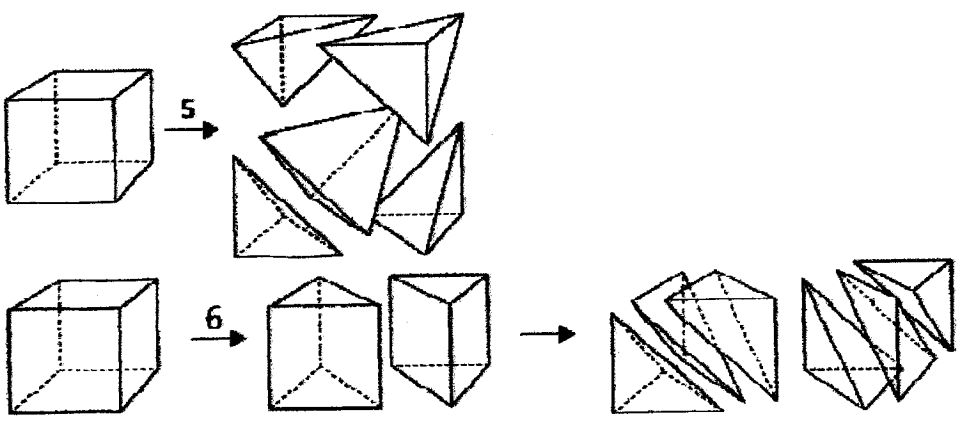

Figure 1.9 Different ways to decompose a cube to 5 or 6 tetrahedra

(d) *Hyperpatch representation.* Each volume element in this representation is a hyperpatch: a generalization of bicubic surface patches. A hyperpatch defines volume, surface area, and internal density variations of a solid element. It is more general than most solid models, which allow only uniform density within a solid primitive; but a price is paid in memory and algorithm complexity (192 scalars are required for each volume element).

1.1.2 Medical Image Analysis

The classification of the used registration methods is based on the criteria formulated by Antone Maintz and Viergever [29].
The main dichotomy of their criteria are :

1) Dimensionality

The main division here is whether all dimensions are spatial *(spatial registration methods)*, or that time is an added dimension *(Registration of time series)*, which can be used for various reasons, such as monitoring of bone growth in children (long time interval), monitoring of tumor growth (medium interval), or post-operative monitoring of healing (short interval).

2) Nature of Registration basis

Image based registration can be divided into *extrinsic*, i.e. based on foreign objects introduced into the image space, and *intrinsic* methods, i.e. based on the image information as generated by the patient.

3) Nature and Domain of the transformation

The nature of the transformation, can be rigid, affine, projective, elastic, or non-rigid transformation as shown in Figure 1.10.

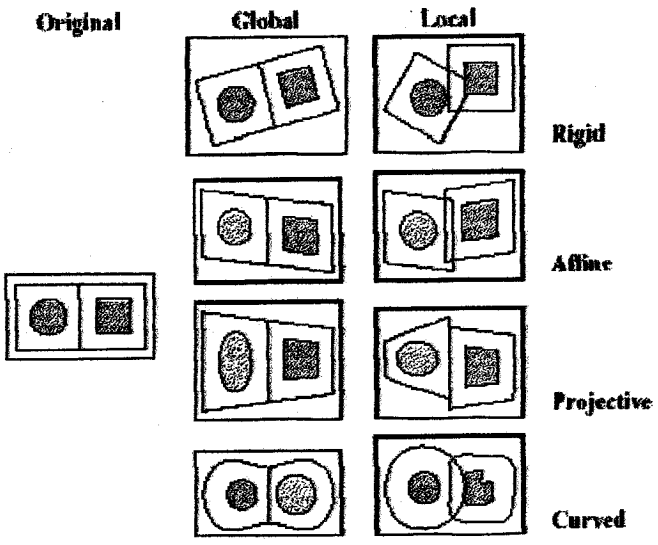

Figure 1.10 Examples of 2D transformations [29].

Domain of the Transformation, A transformation is called "global" if it applies to the entire image, and "local" if subsections of the image each have their own transformation defined.

4) *Optimization Procedure*

- Parameters computed
- Parameters searched for

The parameters that make up the registration transformation can either be computed directly, i.e. determined in an explicit fashion from the available data, or searched for, i.e. determined by finding an optimum of some function defined on the parameter space.

5) *Subject*

- Intra Subject
- Inter Subject
- Atlas

When all of the images involved in a registration task are acquired of a single patient, it is referred to as *intrasubject* registration. If the registration is accomplished using two images of different patients (or a patient and a model), this is referred to as *intersubject* registration. If one image is acquired from a single patient, and the other

image is somehow constructed from an image information database obtained using imaging of many subjects, it is called *atlas* registration. In the literature, many instances of registration of a patient image to an image of a "normal" subject is termed atlas registration. However it is referred to this type of registration as *intersubject*, to keep the class distinctions clear. *Intrasubject* registration is by far the most common of the three, used in almost any type of diagnostic and interventional procedure.

As examples of medical image analysis problems:

1) Integrate structural information from CT or MRI with functional information from radionucleic scanners such as PET or SPECT for anatomically locating a metabolic function.
2) Digital Subtraction Angiography (DSA) – registration of images before and after radio isotope injections to characterize functionality. Digital Subtraction Mammography to detect tumors, early cataract detection.

1.1.3 Remotely Sensed Data Processing [9]

This class includes civilian and military applications in agriculture, geology, oceanography, oil and mineral exploration, pollution and urban studies, forestry and target location and identification.

As an example is the problem of interpretation of well-defined scenes such as airports; locating positions and orientations of known features such as runways, terminals, and parking lots. Another example is the natural resource monitoring, surveillance of nuclear plants, urban growth monitoring.

1.2 Steps of Image Registration

Due to the diversity of the research fields of registration, it is impossible to design a universal method applicable to all registration tasks. Every method should take into account not only the assumed type of geometric deformation between the objects but also application-dependent object representation, required registration accuracy and noise corruption.

Nevertheless, the majority of the registration methods consists of the following three steps:

1. Choosing the basis information best suited for matching.
2. Determining an adequate similarity metric or measure.
2. Searching for optimal or satisfactory values for the unknown variables characterizing the similarity measure.

The first step is to decide what kind of information can be extracted and used to represent the images for the purpose of matching. This information could very well be restricted to the set of image intensities at each pixel (voxel, if in 3D), this type of method is fundamentally characterized by its direct reliance on pixel information; consequently, they are referred to as (*intensity-based* methods).
On the other hand, it could also be certain features extracted from the images. These features can represent salient structures such as strong edges, structural contours, line intersections and points of high curvature within the images, these methods seek to achieve registration indirectly through the alignment of the geometrical features. They are commonly called (*feature based* methods).

In the second step, a similarity metric is chosen to provide a consistent way to evaluate the relative merit of a match. Such a choice is very dependent on the task at hand. The similarity measure is usually a function of some transformation parameters such as rotation, translation etc.

Then, in the third step, certain search strategies are implemented to search through the parameter space characterizing the similarity measure. The search is carried out until an adequate solution (again, whose definition depends on the task at hand) is found.

Implementation of each registration step has its typical problems. In the first step, we have to decide what kind of features is appropriate for the given task. Usually, the physical interpretability of the features is demanded. Detected feature sets in the reference and sensed elements must have enough common elements, even in the situations when the images do not cover exactly the same scene or when there are object occlusions.

In the second step, problems caused by incorrect feature detection or incorrect acquisition can arise. Physically corresponding features can be dissimilar due to different imaging conditions and/or due to the different spectral sensitivity of the sensors. The choices of the feature description and similarity measure have to consider these factors.

Finally in the third step, the type of mapping functions should be chosen according to a *priori* information about the acquisition process and expected image degradations. If no a priori information is available, the model should be flexible and general enough to handle all possible degradations that might appear. The accuracy of the feature detection method, reliability of feature correspondence estimation and the acceptable approximation error need to be considered too. Moreover, the decision as to which differences between images have to be removed by registration has to be taken (i.e. noise that should be rejected). It is desirable not to remove the differences we are searching for if the aim is a change detection. This issue is very important and extremely difficult.

1.3 The Non-Rigid Point Matching Problem [13]

1.3.1 Transformation and Correspondence

A basic non-rigid point matching problem can be defined as follows: given two sets of points (essentially their coordinates), we would like to find the *non-rigid transformation* that best maps one set of points onto the other and/or the set of *correspondence* (including possible outliers[1]) between the points.

The point-sets can be aligned with a good transformation/spatial mapping, which takes one set of points and warps them closely onto the other set. The correspondence basically informs us as to which point in one set is the counterpart of a point in the other set. Usually an answer for the transformation/correspondence is considered to be a good answer when identical or similar structures presented within the two point-sets

1 Points in one point set that have no suitable counterparts in the other and hence need to be rejected during the matching process.

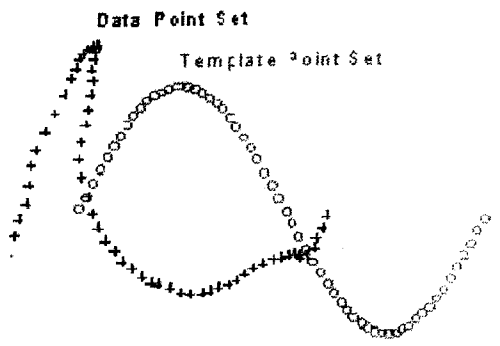

Figure 1.11: A simple non-rigid point matching problem. Two point-sets are shown. The goal is to align these two point-sets by deforming one of them (the template point-set depicted using dots) onto the other (the data or target point-set depicted using crosses)

The transformation and the correspondence are normally regarded as the two unknown variables in a point-matching problem. They share an intimate relationship. Once one variable is known, the solution for the other is actually mostly trivial. Given the set of correspondences (including the set of outliers), finding a good transformation is often a straightforward least-squares problem. On the other hand, given a transformation, we can apply it to one point-set and determine the set of correspondences using some proximity criteria. Consequently, if either variable is deemed known, the point matching problem is considered solved. This is the main reason why the point matching problem can be represented as a problem using either variable (transformation or correspondence), or both. While it may seem simpler to define the problem using a single variable, we will see that it is not necessarily the case for non-rigid point matching.

1.3.2 Difficulty of the Non-rigid Point Matching Problem

A search through the variables' parameter spaces is required in order to obtain a reasonable solution to non-rigid point matching. The search becomes computationally expensive when the dimensionality of the parameter space is high.

Correspondence is one of such high dimensional variables. Even without bringing outliers into the picture, a simple problem of matching of a set of N points to another set of N points would lead to a total of N! permutations.

Any of these permutations can be the possible best solution for correspondence. As the value of N increases, it quickly leads to a combinatorial explosion, making the problem impossible for any simple exhaustive search strategy. Once outliers are included, the situation can be even worse due to the increased number of possibilities. Consequently, as we will discuss later in this chapter, the solution for correspondence has always been carried out with some simplifying assumptions to cut down the search space in all previous methods.

Turning to the transformation, insofar as only rigid mappings are considered, the search often gets restricted to four parameters (rotation, translation and scale) in 2D and nine parameters in 3D. However, we are primarily interested in non-rigid transformations; in this case, the number of parameters grows with the number N of points that are being matched. Furthermore, the non-rigid mapping parameters are real-valued making the search more difficult.

In short, the high dimensionality of the two unknown variables, the non-rigid transformation and the correspondence, composes the main intrinsic[2] factor that makes non-rigid point matching a hard problem. In addition to this intrinsic difficulty, there are other factors that can be encountered in real-world situations that pose further challenges.

The ideal situation for point matching is when the data sets are "*exactly matchable*". There are two ways of construing that term. First, it means that each point in one set has exactly one counterpart in the other. There are no spurious points (outliers) in either set. The second sense of the term is that there is no noise. Thus, through an appropriate non-rigid transformation, we can map a point exactly onto its counterpart. The residue error will be zero in this case, which is an ideal case.

In real-world situations, since the feature points themselves are obtained through feature detectors from possibly corrupted images, there usually is a certain irreducible amount of point "jitter" and outliers. The feature points are therefore corrupted and are not "exactly matchable".

2 It is "intrinsic" because these variables are indispensable for the non-rigid point matching proble m. Other factors such as data quality (e.g. noise) are application dependent and not considered intrinsic.

One common problem is noise that arises from the processes of image acquisition and feature extraction. Noise adds jitter (disturbance) to the feature point location. The consequence is that the exact mapping in the ideal point matching scenario no longer holds. To prevent overfitting (fitting not only the data but also the noise), the algorithm needs to figure out where the true locations of the data points are and conduct the estimation of the non-rigid mapping accordingly. Deciding on the extent to which the data points should be matched is not an easy problem specially when taking into account of the enormous flexibility of the non-rigid transformations involved here.

Another even more difficult factor is the presence of outliers — points in one point-set that have no suitable counterparts in the other and hence need to be rejected during the matching process.

To be able to handle outliers, the algorithm needs to match a subset of one point-set with an appropriate subset of the other. The size of the subsets is not known is advance. This further increases the already enormous parameter space of the correspondence, thus making the problem more difficult.

1.3.3 A General Definition for the Non-Rigid Point Matching Problem

Taking these real-world considerations into account, we define a general non-rigid point matching problem to be the following: given two point-sets with only the points' location/coordinate information, we are required to find the correspondences between the two point-sets, reject possible outliers and determine a good non-rigid transformation that can map one point-set onto the other, while the mapping should not be adversely affected by the possible existing noise.

1.4 Thesis Objectives and Organization

A major goal in this work is to develop a general purpose non-rigid point matching algorithm which is also well suited to real-world registration tasks.

We are mainly interested in one type of feature-based registration — non-rigid point matching. More specifically, in this work,

1. We focus on the feature-based approach.
2. We intend to use the point feature.
3. We seek to develop an algorithm for the feature point registration problem that is capable of handling non-rigid transformations.

The point feature was chosen for the following reasons:

- The point feature is the simplest form of all features. In fact, it often serves as the basis upon which other more sophisticated feature representations, such as curves and surface, are built to incorporate additional information besides the coordinate information.
- With point feature being the foundation of all features, the point matching problem can be regarded as the most fundamental problem in the domain of feature-based registration as well. A careful investigation of point matching in the abstract setting should not only result in improved point feature-based registration but should also provide opportunities for advances in other feature-based registration methodologies as well.

This thesis consists of four chapters. The following statements summarize the scope of each chapter.

- Chapter 1 introduces the registration and relevant topics and emphasizes its applications of the 3D registration in computer vision, pattern recognition, medical imaging, and remotely sensed data processing and finally the definition of the thesis problem.

- Chapter 2 contains an overview of the 3D matching research in general.

- Chapter 3 introduces the ideas implemented and the non-rigid matching algorithm using ICP with other incorporated algorithms, with experiments and results.

- Chapter 4 summarizes the thesis results and contains conclusions and areas of further research.

CHAPTER 2

3D Matching Problem: Literature Review

A considerable amount of research has been conducted on 3D matching. The approaches used to solve the problems can be classified into two categories according to methodology. Approaches in the first category try to create some form of representation for input 3D objects and transform the problem of comparing the input 3D objects to the simplified problem of comparing their representations. These approaches are used most often in model-based object recognition. In contrast, approaches in the second category work on the 3D objects directly without any kind of representation. One data set is aligned to the other by looking for the best transformation, using optimization techniques to search the six-dimensional pose space. These approaches are mainly used for surface registration.

According to the manner of representing the shape of an object, existing representations of 3D free-form objects may be regarded as either global or local. The global methods assume one particular coordinate system attached to an object and represent the object as an implicit or parametric function in this coordinate system. The resulting representation is global in that the implicit function represents the entire shape of the object or of a large portion of the object.

Examples of global representations spherical representations such as EGI (extended Gaussian Image)[24], SAI (Spherical Attribute Image)[22], and COSMOS (Curvedness-Orientation-Shape Map On Sphere)[14], HOT (High Order Tangent) curves [27]. Although global representations can describe the overall shape of an object, they have difficulties in representing objects of arbitrary topology or arbitrary complexity in shape. Moreover, global representations have difficulty in handling clutter and occlusion in the scene.

Many local representations are primitive-based. In [36], super-segments and splashes are proposed to represent 3D curves and surface patches with significant structural changes.

In [12], a three-point-based representation is proposed to register 3D surfaces and recognize objects in cluttered scenes. On the scene object, three points are selected with the requirement that (1) their curvature values can be reliably computed; (2) they are not umbilical points; and (3) the points are spatially separated as much as possible. Then, under the curvature, distance, and direction constraints, different sets of three points on the model surface are found to correspond to three points on the scene objects. The transformations computed using those scene-model correspondences are verified to select the best one. A similar approach was proposed in [40], using a representation called SPS (Surface Point Signature), as special points on the surface are identified, then surface registration is performed by matching SPS images of different surfaces and hence finding corresponding points in each surface.
Another a local representation called Spin-Images is proposed in [26]. HSI (Harmonic Shape Images) in [41]. Although local representations can not provide an overall description of the object shape, they have advantages in handling clutter and occlusion in the scene.

Among the other class of approaches that attempts to match set of points directly without any prior surface fitting (3D surface registration algorithms) , Iterative Closest Point (ICP) [6] in which the distance between point sets is computed and minimized to find the best transformation between model and scene. This approach does not require any surface segmentation or surface fitting. The main drawback of this approach is that, like any minimization technique, it requires an initial guess of the transformation between model and scene or data. We will focus on this approach in the next chapter. A K-D tree structure was also used in [41] to speed up the process of finding the closest point. More recently, an approach called RPM (Robust Point Matching) was proposed in [13], which is capable of handling the non-rigid point matching problem.

Most of the previously mentioned methods work well only for rigid transformations. When it comes to non-rigid mappings, the huge number of transformation parameters usually renders these methods ineffective. For non-rigid matching, most authors are not looking to minimize a criterion, but attention has rather focused on the type of smoothing or physical model of deformation to be used.

In the subsequent sections, we will review many of the above approaches in detail.

2.1 Extended Gaussian Image (EGI)

The Extended Gaussian Image (EGI) of an object records the variation of surface area with surface orientation. The EGI is a unique representation for convex objects.
Before we go through the Extended Gaussian Image (EGI) we first define what is the Gaussian Image.

2.1.1 The Gaussian Image

Surface normal information for any object maybe mapped onto a unit (Gaussian) sphere by finding the point on the sphere with the same surface normal:
This mapping is called the Gaussian Image of the object when the surface normals for each point on the object are placed such that:

- Tails lie at the center of the Gaussian sphere.
- Heads lie on the sphere at the matching normal point

So, in areas of convex objects with positive curvature, no two points will have the same normal, and rotations of the object correspond to rotations of the sphere.

We can *extend* this process so that

- a weight is assigned to each point on the Gaussian sphere equal to the area of the surface having the given normal
- This mapping is called the *extended Gaussian image* (EGI).
- Weights are represented by vectors parallel to the surface normals, with length equal to the weight.

An example of such an extended Gaussian image is shown in Fig. 2.1

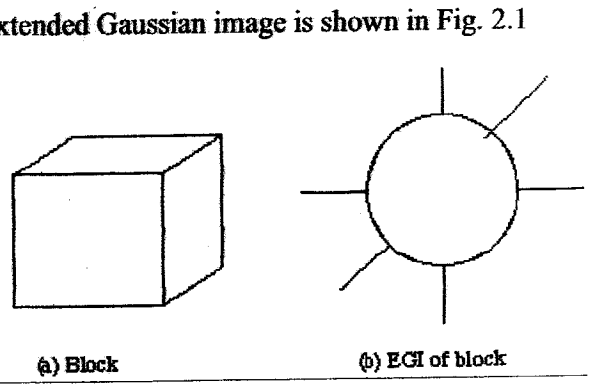

Figure 2.1 The EGI of a block

Disadvantages:

- EGIs only uniquely define convex objects.
- An infinite number of non-convex objects can possess the same EGI as shown in Figure 2.2.

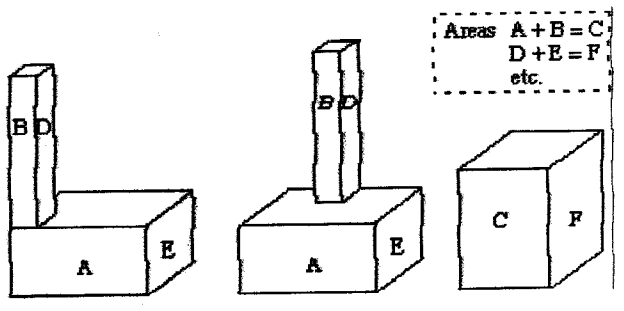

Figure 2.2 Examples of objects with the same EGI

2.2 Spherical Attribute Image (SAI)

This approach [22] begins with a combination of the point set matching and the original EGI approach. As in the case of the point set matching that avoid fitting analytical surfaces to represent an object. Instead a representation was used that simply consists of a collection of points, or nodes, arranged in a mesh covering the entire surface of the object. This has the advantage that the object can have any arbitrary shape, as long as that shape is topologically equivalent to the sphere.

As in the EGI algorithms, each node of the mesh is mapped onto a regular mesh on the unit sphere, and a quantity that reflects the local surface curvature at the node is stored at the corresponding node on the sphere. Instead of using a discrete approximation of the curvature, another measure of curvature was used, the simplex angle, which is entirely defined from a node and its neighbors in the mesh without any reference to the underlying continuos surface. The corresponding spherical representation is called the spherical attribute image (SAI).

To determine whether two objects are the same, we only need to compare the corresponding spherical distributions. The overall approach is illustrated in Fig 2.3

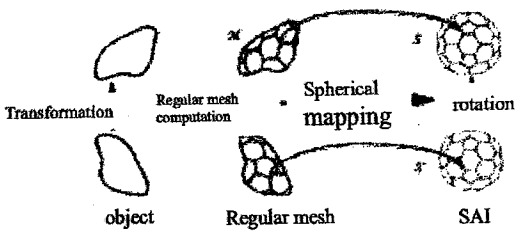

Figure 2.3 Object Recognition using SAI [22]

A regular mesh is computed from input sensor data; a simplex angle is computed at each node of the meshes and the meshes are mapped onto a sphere, the SAI. A fundamental difference between the SAI and other global representations is that a unique mesh, up to rotation, translation, and scale, can be reconstructed from a given SAI. In the case of the EGI, for example, this property is true only for convex objects. Another fundamental difference is that the SAI preserves connectivity in that patches that are connected on the surface of the input object are still connected in the spherical representation. The latter is the main reason why this approach can handle arbitrary non-convex objects and in the presence of occlusion.

2.3 Curvedness-Orientation-Shape Map On Sphere (COSMOS)

This representation scheme [14] is mainly for recognizing 3D free form objects. Figure 2.4 shows some examples for free form objects

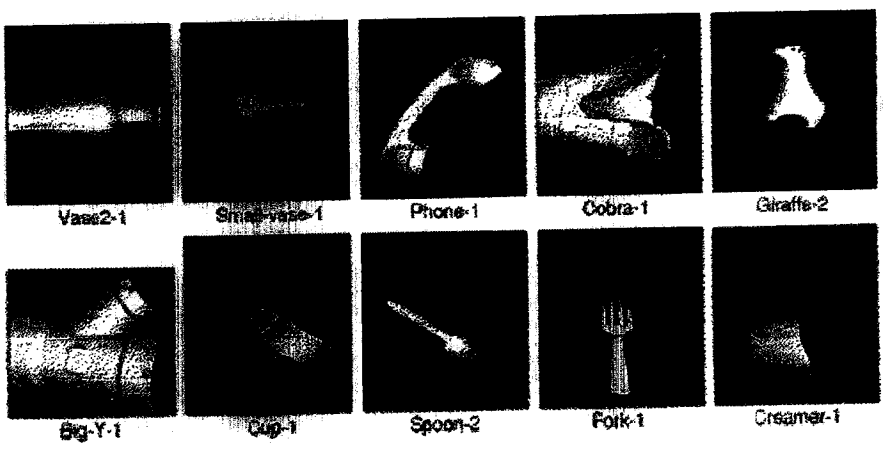

Figure 2.4 Range images of 3D free form objects [14]

Free Form Surfaces Definition

A free-form surface *S* is defined to be a smooth surface, such that the surface normal is well defined and continuous almost everywhere, except at vertices, edges, and cusps.

The goals of the COSMOS (Curvedness–Orientation-Shape Map on Sphere) representation scheme were; first, to be a general representation scheme that can be used to describe sculpted objects as well as objects composed of simple analytical surface primitives. Second, to be as compact and expressive as possible for accurate recognition of objects from a single range image.

The representation uses the *shape index* to represent complex objects for their recognition. Koenderink and Van Doorn originally proposed the shape index for graphical visualization of surfaces [28]. An object is concisely characterized by a set of maximally sized surface patches of constant shape index and their orientation dependent mapping onto the unit sphere. The patches that get assigned to the same point on the sphere are aggregated according to the shape categories of the surface components.

The points on the unit sphere are further characterized by a set of support functions describing the shape, average curvedness, and area of the mapped patches. The average curvedness of a surface patch specifies whether it is highly or gently curved; the surface area quantifies its extent in three-dimensional space; the orientation (mean surface normal) of the patch describes how it is directed in 3D space. The relative spatial arrangement of the various patches as encoded by their connectivity is also built into the representation.

The concept of *shape spectrum* is also included in the COSMOS framework. This allows free-form objects views to be grouped in terms of the shape categories of the visible surfaces and the surface areas.

For the recognition purpose, COSMOS adapted a feature representation consisting of the moments computed from the shape spectrum of an object view. This eliminated

unlikely object view matches from a model database of views. Once a small subset of likely candidate views has been isolated from the database, based on the views' spectral features, a detailed matching scheme that exploits the various components of the COSMOS representation is performed to derive a matching score and to establish view surface patch correspondences. This matching scheme is inspired by traditional graph matching algorithms and exploits the natural structural information that is being made explicit in the COSMOS representation.

2.4 High Order Tangent Curves (HOT)

Smooth curved surfaces are described by HOT curves [27], a discrete, non-parametric representation anchored in different geometry. These curves determine the structure of the image contours and its catastrophic changes, and there is a natural correspondence between some of them and monocular contour features such as inflections and bitangents.

We say that a tangent vector at some point has contact of order n with the surface (or more concisely is of order n) when the derivative of order i of the surface equation in the direction of the tangent is zero for all $i < n$, and non-zero for $i = n$ .While all surface points have an infinity of tangents of order two (or greater) in their tangent plane, only hyperbolic points may have third order tangents. Contact of order four or higher only occurs along certain surface curves (i.e. parabolic and flecnodal curves), and fifth order contact only occurs at isolated points along these curves [36].

Additionally, there are other surface curves where a line grazes the surface in multiple discrete points with at least second order contact in some exceptional manner: the limiting bitangents, the asymptotic bitangents, and the tritangents. A limiting bitangent touches the surface at two points sharing a common (bi)tangent plane; an asymptotic bitangent is an asymptotic direction at one of the two contact points. Each of these curves has a corresponding ruled surface which grazes the original surface along two curves. Finally, a tritangent grazes the surface in three distinct points and has a corresponding ruled surface.

The parabolic, ecnodal, limiting and asymptotic bitangent, and tritangent curves form the basis for a shape representation aimed at automatic model construction and object

recognition. We propose to maintain an explicit discrete representation of these _ve HOT curves (where the surface admits High Order Tangents), recording the position of each curve point on the surface, the direction of its surface normal, and the direction of its "special" (bi)tangent.

There is a close relationship between the three-dimensional HOT curves and the two-dimensional contour inflections and bitangents (Fig. 2.5).

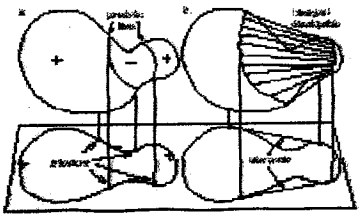

Figure 2.5 Parabolic curves, limiting bitangent developables, and their projections [27]

2.4.1 HOT Curves Reconstruction

The algorithm for reconstructing the asymptotic lines at parabolic points and the bitangent line of the limiting bitangent developable surface was implemented, by placing the object to be modelled on a turntable; this scene is viewed from a nearby camera in general position (i.e. the image plane is not parallel to the axis of rotation). In the example shown, a special glass bottle was chosen (which is approximately a solid of revolution), since the parabolic and bitangent developable lines are readily interpreted. In particular, these curves should be circles centered on the object's axis while the corresponding visual event curves should sweep lines of latitude on the view sphere. These methods do not make use of the fact that this object is a solid of revolution. 280 images were taken at one degree increments.

Figure 2.6.a-c shows a cropped image of the bottle, the detected contours, inflections and bitangent lines, and the temporal feature tracks. Figure 2.6.d shows the recovered parabolic curves. In Fig. 2.6.e the asymptotic directions are drawn along the parabolic lines. Fig. 2.6.f shows the recovered limiting bitangent developable surface that lies on the convex hull. The recovered asymptotic directions, which correspond to lip and beak events in an aspect graph, are drawn on the view sphere in Fig. 2.6.g. The

tangent crossing events (the direction of limiting bitangent lines) are shown in Fig. 2.6.h.

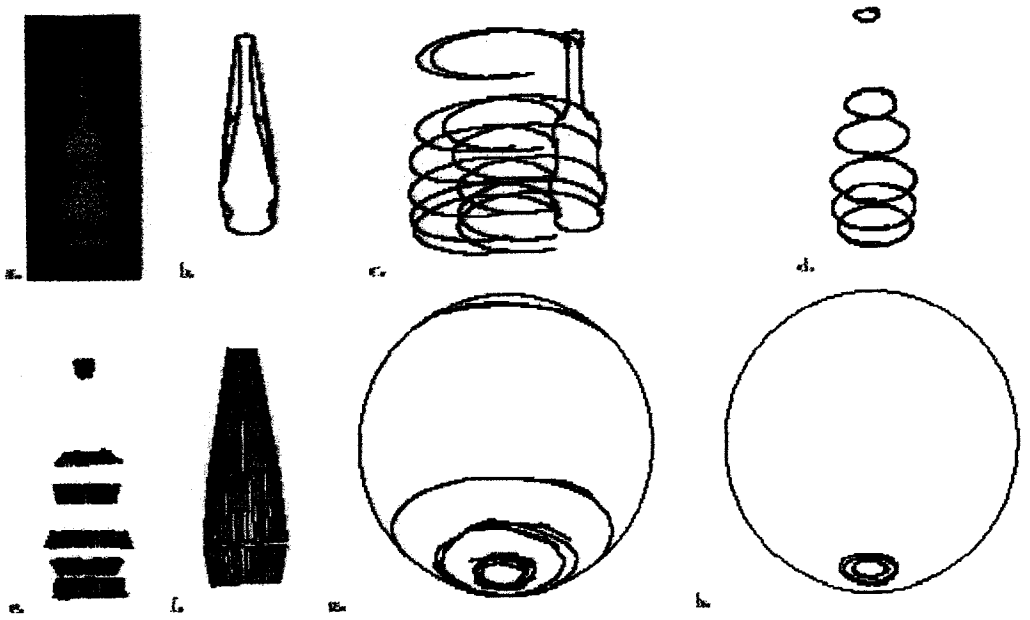

Figure 2.6 HOT curve reconstruction [27]: a. An image. b. Extracted features. c. Tracked inflections. d. Reconstructed parabolic curves. e. Asymptotic directions along the parabolic curve (from a side view). f. Reconstructed limiting bitangent developable surface that lies on the convex hull (from a side view). g. Beak and Lip transitions on the view sphere. h. Tangent crossing events.

2.5 Splash Representation

Geometric indexing has been one of the most used surface indexing techniques because it used the geometrical relationships between invariant features. However another form of indexing that uses local shape information has become more popular. As it is based on structural information local to the neighborhood of a point, this indexing method is called "Structural indexing".

Circle of points was first used to describe the underlying surface structure around a given point. This can be done by decomposing the local surface patch around a specific point into a series of contours, each of which is the locus of all points at a certain distance from the specific point.

Stein and Medioni [36] extended this idea further. Instead of decomposing a surface patch into a series of contours of different radii, a few contours at prefixed radii are extracted as shown in Fig. 2.7. On each contour of points, surface normals are

computed. This contour is called a "splash". A 3-D curve us constructed from the relationship between the splash and the normal at the center point. This curve is converted into piecewise linear segments. Curvature angels between these segments and torsion angels between their binormals are computed. These two features are used to encode the contour.

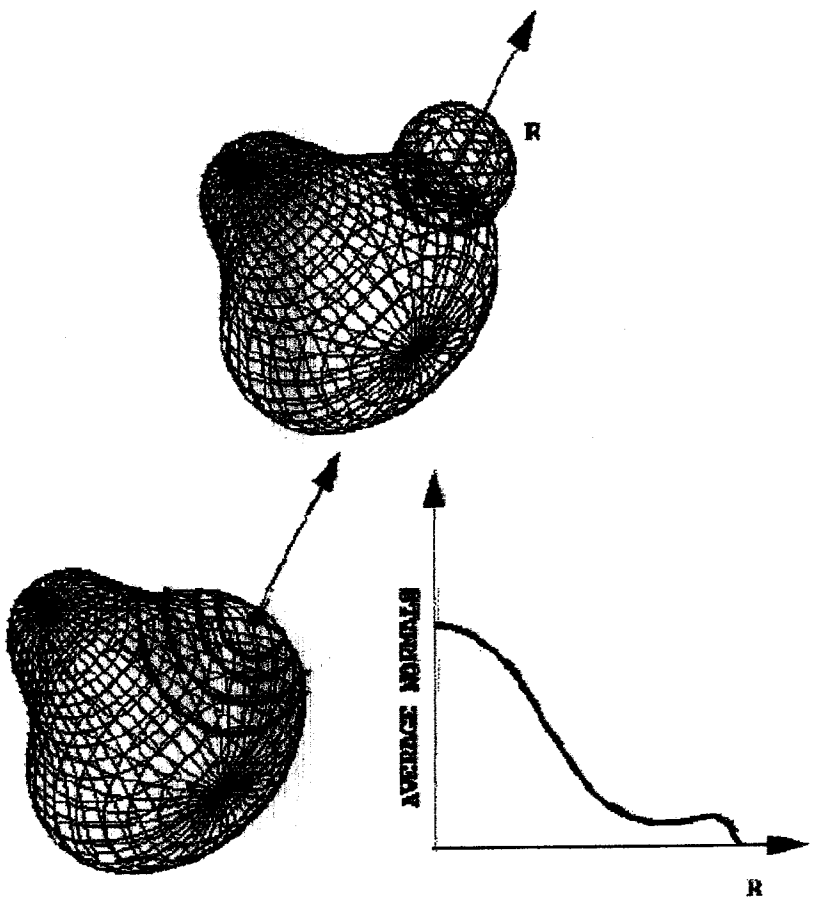

Figure 2.7 An illustration if the "splash" representation scheme. At specific points on the surface, the intersections of the surface patches and the spheres of pre-fixed radii are obtained. For each intersection a curve representing the average normal of the points in the intersection and the point in study is obtained. These curves are further used for matching

Matching is performed using the contour codes of points on the two surfaces. Fast indexing was achieved by hashing the codes for all models in the library into an index table.

At run-time recognition, the splashes of highly structured regions are computed and encoded using the same encoding scheme. Models that contain similar codes as the

splashes appearing in the scene are extracted. Verification is then performed for each combination of three correspondences.

2.6 The Spin Image Representation

Spin Images [26] is an approach that can be used for recognition of complex objects in cluttered 3D scenes. The name *spin image* was chosen; *image* because the representation is a 2D array of values, and *spin* because the image generation process can be visualized as a sheet spinning about the normal of a point. The spin image, comprises descriptive images associated with oriented points on the surface

Through correlation of images, point correspondences between a model and scene data are established. Geometric consistency is used to group the correspondences from which plausible rigid transformations that align the model with the scene are calculated. The transformations are then refined and verified using a modified ICP algorithm.

2.6.1 Oriented Points

A fundamental component of surface matching representation is an *oriented point*, a three dimensional point with an associated direction. Oriented points are static versions of oriented particles. Oriented points are used to create spin-images. We define an oriented point O at a surface mesh vertex using the 3-D position of the vertex $\boldsymbol{p}$ and surface normal at the vertex $\boldsymbol{n}$. The surface normal at a vertex is computed by fitting a plane to the points connected to the vertex by the surface mesh. Specifically, surface normal at a vertex is the eigenvector with the smallest eigenvalue of the inertia matrix of vertex and the other vertices directly connected to it by the edges of the surface mesh.

The above eigenvector computation does not determine the inside/outside direction of the surface normal; for spin-image generation, oriented points should be oriented to the outside of the object surface. If the surface mesh was created from a sensor with a single viewing direction, then the normal direction can be chosen as the one pointing toward the sensor. Otherwise, surface normals of a mesh must be oriented to the

outside of the object using the following heuristic. First, a vertex is chosen and the orientation of its normal is spread to the normals of its adjacent vertices. This process is repeated until the normals of all of the vertices are consistently oriented to the inside or outside of the object. Next the orientation (inside/outside) of all of the normals on the surface is determined by calculating the scalar products of the surface normal at each vertex and the vector from the centroid of the object to the vertex. If the majority of scalar products are positive, the normals have been oriented to the outside. Otherwise, the normals have been oriented to the inside, so they are inverted. If the object has multiple connected components, this normal orientation procedure is applied separately to each connected component. To date, we have never encountered an object where this heuristic would not generate outside oriented surface normals, although objects can be constructed where it will fail. Given this method for computing surface normal, an oriented point can be constructed at each vertex of a surface mesh using the position of the vertex and its surface normal.

As shown in Figure 2.8, an oriented point defines a 5 degree of freedom (DOF) basis ***(p,n)*** (i.e., local coordinate system) using the tangent plane P through p oriented perpendicularly to n and the line L through p parallel to n.

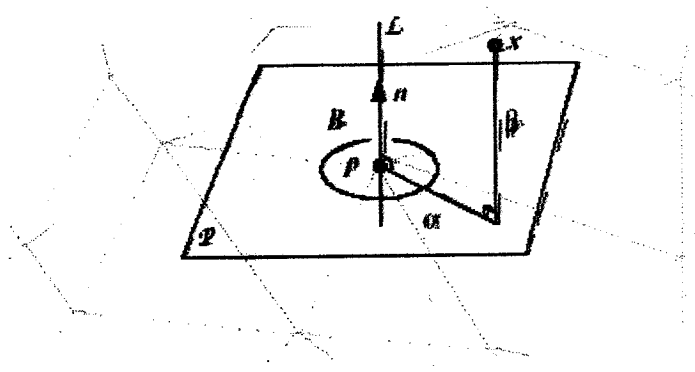

Figure 2.8 An oriented point basis created at a vertex in a surface mesh [26]. The position of the oriented point is the 3D position of the vertex, and the direction of the oriented point is the surface normal at the vertex. Two coordinates can be calculated given an oriented point: α the radial distance to the surface normal line L and β the axial distance above the tangent plane P.

The two coordinates of the basis are α, the perpendicular distance to the line L, and β the signed perpendicular distance to the plane P. An oriented point basis is a cylindrical coordinate system that is missing the polar angle coordinate (because this

coordinate cannot be determined using just surface position and normal). Using an oriented point basis *O*, we can define a *spin-map SO* as the function that projects 3D points x to the 2D coordinates of a particular basis (p,n) corresponding to oriented point *O*.

$$S_o : R^3 \to R^2$$

$$S_o(x) \to (\alpha, \beta) = \left(\sqrt{\|x-p\|^2 - \left(n \cdot (x-p)\right)^2}, n \cdot (x-p) \right) \qquad (2.1)$$

2.6.2 Spin Image Generation

Each oriented point *O* on the surface of an object has a unique spin-map S*o* associated with it. When S*o* is applied to all of the vertices of a surface mesh *M*, a set of 2D points is created. In this case, the vertices on the surface of *M* are the pre-image of the spin-map, and the resulting 2D points are the image of the spin-map. We will use the term *spin-image* $I_{O,M}$ to refer to the result of applying the spin-map S*o* to the vertices of *M*. A spin-image is a description of the shape of an object because it is the projection of the relative position of 3D points that lie on the surface of an object (vertices) to a 2D space where some of the 3D metric information is preserved. Figure 3.9 shows three 2D point sets that result from applying the spin-map at a point on the surface of a rubber duckie mesh model to all of the vertices in the mesh.
Once all the points on the surface have been processed, the 2D array is converted into a gray image

Then spin images can be compared using linear correlation coefficients. The magnitude of the correlation coefficients is used as well as the confidence in the correlation results which is measured by the variance of the correlation coefficient. Since the linear correlation coefficient is a function of the number of pixels used to compute it, the amount of overlap between spin images will have an effect on the correlation coefficients obtained. The more pixels used to compute a correlation coefficient, the more confidence there is in its value.

An experimental analysis of recognition rate versus the clutter and occlusion shows that surface matching using spin images degrades gracefully, not catastrophically,

with the addition of clutter and occlusion to a scene, making spin images an effective representation for object recognition in complex scenes.

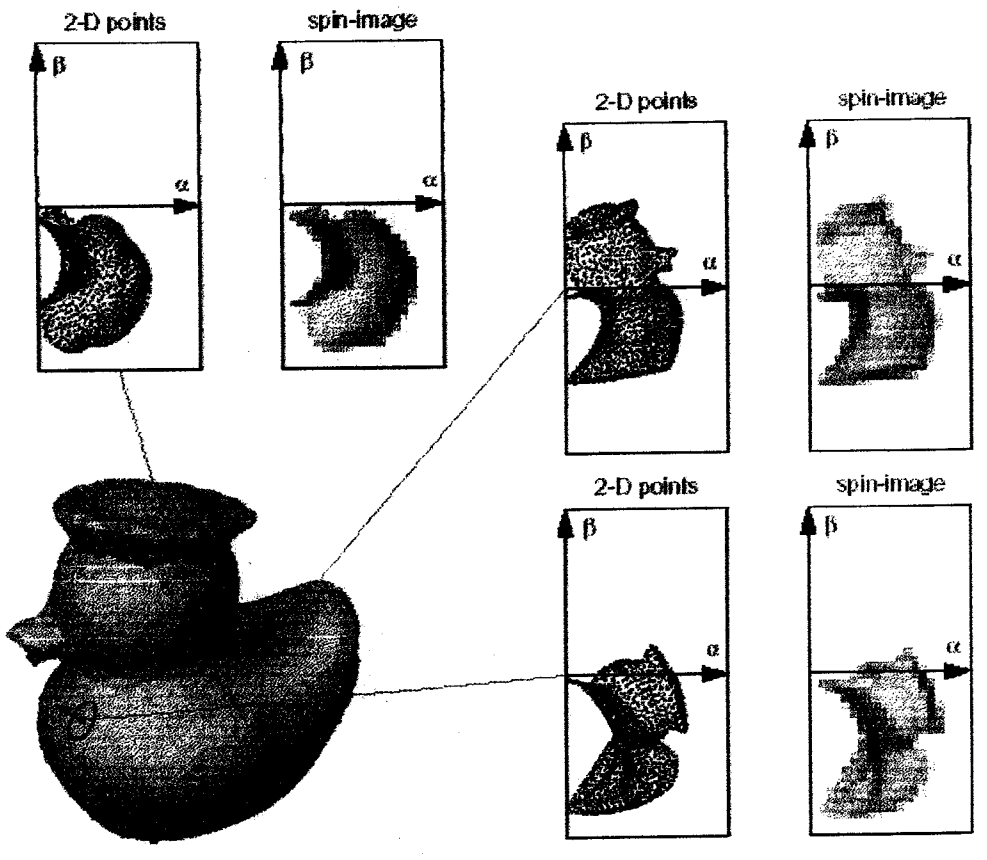

Figure 2.9 Spin-images for three oriented points on the surface of a rubber duckie model [26]. The 3D position of vertices in the mesh are mapped into 2-D using the spin-map for each oriented point basis. By accumulating 2-D points in discrete bins, spin-images are generated.

2.7 Harmonic Shape Images (HSI)

The key concept of Harmonic Shape Images [41] is to compare surfaces by comparing patches on them. A surface patch is defined to be a connected region without holes on a given surface. Harmonic Shape Images are 2D shape representations of surface patches. Using this representation, the problem of 3D surface matching is reduced to 2D image matching. The paradigm in Figure 3.10 illustrates this idea.

Given two surfaces *S1* and *S2*, most of the previous approaches conduct point-based matching as shown in Figure 2.10(a). As a result, only point-to-point correspondences can be established when the representations of two points match. In contrast, the

approach proposed in [41] matches patches on two surfaces as shown in Figure 2.10(b). Correspondences between the two patches, i.e., correspondences between every pair of points in the two patches, can be established immediately without any extra cost once the representations of the two patches match each other.

Making use of surface continuity allows the natural establishment of correspondences between two surfaces after the matching process. This means that the difficult problem of finding correspondences on two surface patches becomes trivial. The matching process does not only provide a matching score, but also the correspondences, i.e., the correspondences are obtained without any extra computation. In contrast, if surface continuity is discarded, then the correspondence between only one pair of points (usually the centers of patches) can be established.

In order to find more pairs of correspondences, more matching needs to be conducted and heuristic-guided post-processing has to be done to organize the matched points into mutually consistent correspondences.

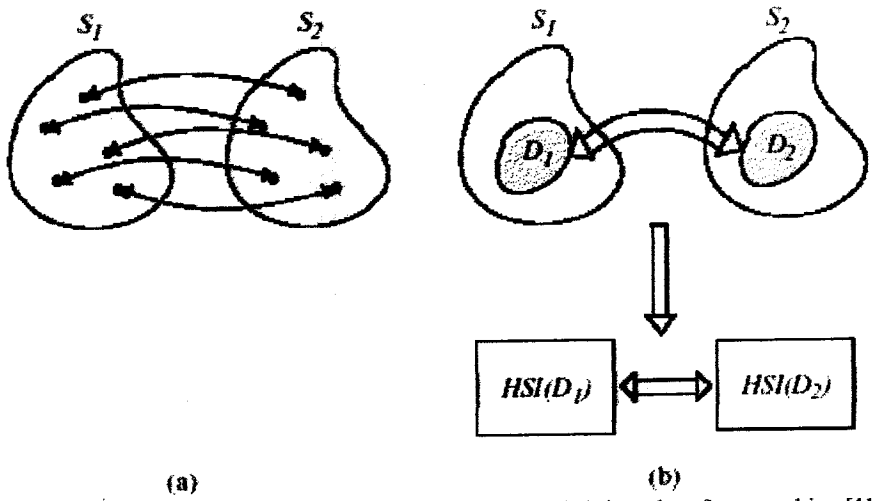

Figure 2.10 Illustration of point-based surface matching and patch-based surface matching [41].

This approach describes patch-based surface matching conducted by matching Harmonic Shape Images of those patches as shown in Figure 2.10(b). Therefore, creating those images is the core of the patch-based matching.

2.7.1 Generation of Harmonic Shape Images

Given a 3D surface S as shown in Figure 2.11(a1), let v denote an arbitrary vertex on S. Let $D(v, R)$ denote the surface patch which has the central vertex v and radius R. R

is measured by distance along the surface. *D(v, R)* is assumed to be a connected region without holes. *D(v, R)* consists of all the vertices in *S* whose surface distances are less than, or equal to, *R*. The overlaid region in Figure 2.11(a1) is an example of *D(v, R)*. Its amplified version is shown in Figure 2.11(b1). The unit disc *P* on a 2D plane is selected to be the target domain. *D(v, R)* is mapped onto *P* by minimizing an energy functional. The resultant image *HI(D(v, R))* is called the harmonic image of *D(v, R)* as shown in Figure 2.11(c1).

As can be seen in Figure 2.11(a1) and (c1), for every vertex on the original surface patch *D(v, R)*, one, and only one, vertex corresponds to it in the harmonic image *HI(D(v, R))*. Furthermore, the connectivities among the vertices in *HI(D(v, R))* are the same as that of *D(v, R)*. This means that the continuity of *D(v, R)* is preserved on the harmonic image *HI(D(v, R))*.

The preservation of the shape of *D(v, R)* is shown more clearly on the Harmonic Shape Image *HSI(D(v, R))*(Figure 2.11(d1)) which is generated by associating shape descriptor at every vertex on the harmonic image(c1). The shape descriptor is computed at every vertex on the original surface patch(b1). On *HSI(D(v, R))*, high intensity values represent high curvature values and low intensity values represent low curvature values. The reason for Harmonic Shape Images' ability to preserve the shape of the underlying surface patches lies in the energy functional which is used to construct the mapping between a surface patch *D(v, R)* and the 2D target domain *P*. This energy functional is defined to be the shape distortion when mapping *D(v, R)* onto *P*. Therefore, by minimizing the functional, the shape of *D(v, R)* is maximally preserved on *P*.

Another surface patch is shown in Figure 2.11(a2) and (b2). Its harmonic image and Harmonic Shape Image are shown in (c2) and (d2), respectively. In this case, there is occlusion in the surface patch(Figure 2.11(b2)). The occlusion is captured by its harmonic image and Harmonic Shape Image(Figure 2.11(c2), (d2)). The latter's ability to handle occlusion comes from the way the boundary mapping is constructed when mapping the boundary of *D(v, R)* onto the boundary of *P*; because of the boundary mapping, the images remain approximately the same in the presence of occlusion.

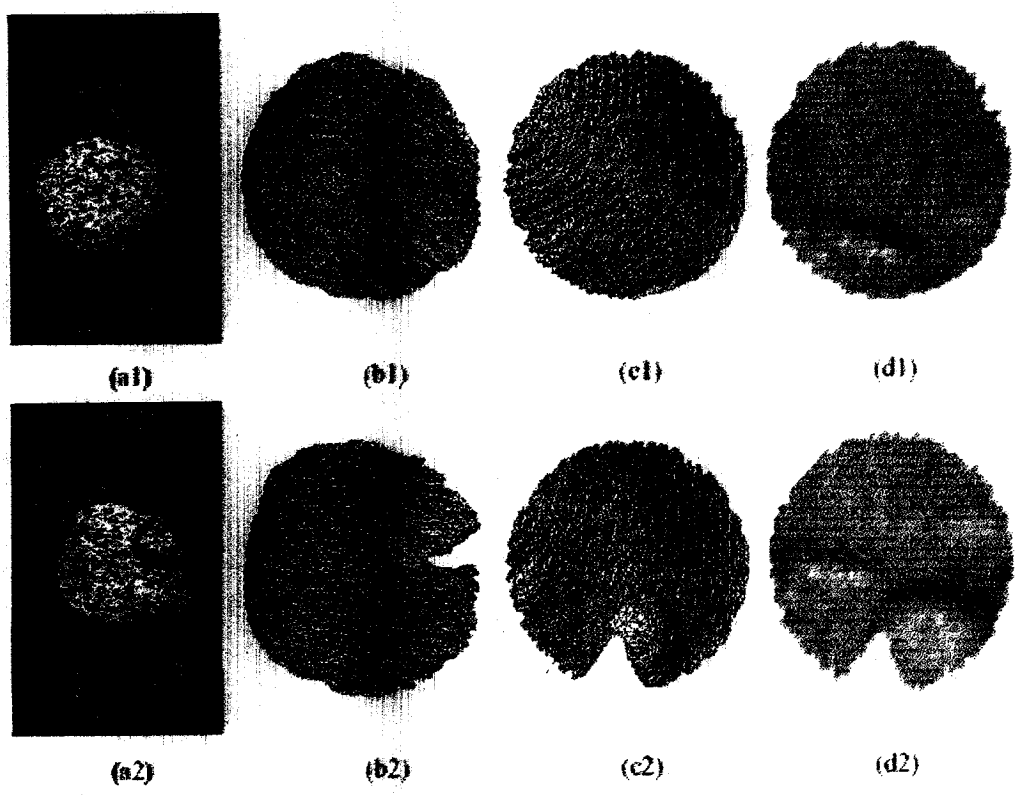

Figure 2.11 Examples of surface patches and Harmonic Shape Images [41]. (a1), (a2) Surface patches on a given surface; (b1), (b2) the surface patches in wireframe; (c1), (c2) their harmonic images; (d1), (d2) their Harmonic Shape Images.

From the above generation process, it can be seen that the only requirement imposed on creating Harmonic Shape Images is that the underlying surface patch is connected and without holes. This requirement is called the topology constraint.

2.7.2 Properties of Harmonic Shape Images

Harmonic Shape Images are a patch-based representation for 3D free-form surfaces. For a given surface, a surface patch with radius R can be generated for every vertex on that surface.

This means that a Harmonic Shape Image can then be generated for each surface patch as long as that surface patch satisfies the topology constraint. The size of the a surface patch is defined by the radius R. When R increases, the area of the surface patch increases and its Harmonic Shape Image consists of more

information of the underlying surface. Another point that needs to be noticed about the patch representation is that every vertex on a given surface is treated equally. No feature points or special points need to be selected in order to create Harmonic Shape Images. This is in contrast to previous feature-based (or primitive-based) representations in which the robustness of feature extraction is a difficult issue to cope with.

Harmonic Shape Images are defined on a simple domain, which is a unit disc. This simplifies the 3D surface matching problem to a 2D image matching problem. Furthermore, because the unit disc is specified as the domain for any given surface patches, the Harmonic Shape Images of all those patches are defined in the same coordinate system regardless of the actual positions of those patches. This means that Harmonic Shape Images are pose invariant.

Harmonic Shape Images capture both the shape and the continuity information of the underlying surface patches. It can be seen easily from Figure 2.11 that there is one-to-one correspondence between the vertices on the surface patch and its harmonic image. In fact, the mapping from the surface patch to the disc domain is a well-behaved mapping. It is one-to-one and onto; it is continuous; it is unique; and it is intrinsic to the original surface patch. This property allows the natural establishment of correspondences between two surface patches once their Harmonic Shape Images match. Finally, Harmonic Shape Images are robust to occlusion and sampling resolution.

The comparison of some surface representations previously proposed is listed in Table 2.1.

Table 2.1 Comparison of some surface representations

Comparison Items / Representation	Object Domain	Type of Representation	Mapping	Comparison of Representation
SAI	Objects with Spherical topology	Global	Spherical mapping of surface curvature at all points	Complete, both surface shape and continuity are represented
Splashes	Objects with free-form surfaces	Local	Gaussian map of surface normals along a geodesic circle	Partial
COSMOS	Objects with free-form surfaces	Global	Spherical mapping of orientation of CSMPs	Partial for non-convex objects
Spin-Images	Objects with free-form surfaces	Local	2D histogram of distances to the reference tangent plane and surface normal at all points	Partial, surface continuity is not represented
HSI	Objects with free-form surfaces	Local	Harmonic map of the underlying surface onto a unit disc surface curvature is stored on the map for all points	Complete, both surface shape and continuity are represented.

2.8 Robust Point Matching (RPM)

RPM [13] is an algorithm that jointly estimates a one-to-one correspondence and a non-rigid transformation between two sets of points. It is built upon the heuristic of *softassign*, which allows for multiple partial correspondences between points. With the help of the *deterministic annealing* technique, this new heuristic enables the algorithm to overcome many local minima that can be encountered in the matching process.

2.8.1 Point Matching as Joint Estimation of Correspondence and Transformation

Notations

Suppose we have two point-sets V and X (in $\Re^2$or in $\Re^3$) consisting of points $\{v_a,\ a = 1, 2,\ldots ,K\}$ and $\{x_i;\ i = 1, 2, \ldots ,N\}$ respectively. Note that the total number of points can be different so it is possible that K does not equal N. For the sake of simplicity, we will assume for the moment that the points are in 2D. We always apply the transformation to one point-set V so that it can be better aligned with the other point-set X. For this reason, we call point-set V the model and point-set X the target.

We represent the non-rigid transformation by a general function f with parameters α. A point v_a is mapped to a new location $u_a = f(v_a, \alpha)$. The whole transformed point-set V is then U or $\{ u_a \}$. We use a *prior smoothness* measure to place appropriate constraints on the mapping to prevent it from behaving too arbitrarily. To this end, we introduce an operator L and our chosen smoothness measure is $\|Lf\|^2$.

Formally, given two point-sets V and X, point matching can be regarded as the minimization of the following *binary* linear assignment-least squares energy function:

$$E_1(Z,\alpha) = \sum_{i=1}^{N}\sum_{a=1}^{K} z_{ai}\left\|x_i - f(v_a,\alpha)\right\|^2 + \lambda\|Lf\|^2 - \xi\sum_{i=1}^{N}\sum_{a=1}^{K} z_{ai} \qquad (2.2)$$

Where the parameters λ and ξ are just constants and the variable Z or $\{z_{ai}\}$, with only possible binary values 0 and 1, is subject to the following constraints,

$$\sum_{i=1}^{N+1} z_{ai} = 1, \quad for \quad a \in \{1,2,...,K\}$$

$$\sum_{a=1}^{K+1} z_{ai} = 1, \quad for \quad i \in \{1,2,...,N\}$$

The energy function can be easily dissected. The first term is essentially a similarity measure to evaluate how good a fit is with a set of specific correspondence and transformation. The second term is the prior constraint on the transformation. The third term is the robustness control term preventing rejection of too many points as outliers. The parameters , and are the weight parameters to balance these terms.

The matrix Z or $\{z_{ai}\}$ is called the *correspondence matrix* [18]. It consists of two parts. The inner K x N part of Z defines the correspondence. If a point v_a corresponds

to a point x_i, $z_{ai} = 1$: If $z_{ai} = 0$; then the points are unmatched and their distance does not contribute to the total energy. The row and column summation constraints guarantee that the correspondence is one-to-one. The outer extra $(N+1)^{th}$ row and $(K+1)^{th}$ column of Z are introduced to handle the outliers. When a point is rejected as an outlier, the related inner entries all become zero. Then the extra outlier entries will start taking non-zero values to satisfy the constraints.

2.8.2 Softassign

The basic idea of softassign [18] is to relax the binary correspondence variable Z to be a continuous valued matrix M in the interval of [0; 1], while still enforcing the row and column constraints.

The continuous nature of the correspondence matrix M basically allows fuzzy, partial matches between the point-sets. The correspondence becomes more like a probability measure. Now, one point does not necessarily just correspond to only one other point; it could have multiple possible matching partners, while maybe preferring the ones which are closer to itself a little bit more. This allows the point to "see" further away to find a potentially ideal match. From an optimization point of view, this fuzziness makes the resulting energy function better behaved because the correspondences are able to improve gradually and continuously during the optimization without jumping around in the space of binary permutation matrices (and outliers). In more formal terms, making the correspondence fuzzy smoothes the energy function and gets rid of poor local minima.

Though allowed to be fuzzy, the correspondence matrix still has to satisfy the row and column constraints. This can be enforced via the iterative row and column Sinkhorn balancing normalization [35] of the matrix M.

There are different degrees of fuzziness. The extremely fuzzy case happens when all mai are equal, i.e., any point thinks any point else is probably its partner without any preference at all. At the other end of the spectrum is the least fuzzy case when a point only choose one partner, which is most likely the closest, and disregard the rest. This is the same as the binary correspondence. So in fact the binary correspondence can be regarded as one extreme case of the fuzzy correspondence. Between these two

extremes, there can be many intermediate degrees of fuzziness when a point would only choose a portion of possible points (neither all of them, nor a single one) as its partners. The idea of different degrees of fuzziness is illustrated in Figure 2.12.

Figure 2.12 Different degrees of fuzzy correspondence [13]. From left to right, i) two point-sets are shown; ii,iii,iv) The degrees of the fuzziness gradually decrease from the extremely fuzzy to the least fuzzy (binary).

In the search mechanism using different degrees of fuzziness, each point is always actively searching for its potential partners while the searching is regulated by some "*search range*".

In the extremely indistinguishable case, one point is being matched to almost all the points in the other set, even including those that are quite far away from itself. At this stage, any single point search range can be interpreted as being so large that it puts all other possible points within its searching scope, causing the resulting correspondence to be very fuzzy.

In the least fuzzy case where a point only choose the closest partner, it is as if the "search range" has shrunk to so small a value that only the closest candidate can get in.

With an intermediate degree of fuzziness, the search range is neither large enough to accommodate all candidates, nor so small that the point can consider no more than the closest candidate. A scale-space strategy can be easily conceived at this point by manipulating such a search range, thus effectively controlling the degree of fuzziness. A point can start with searching globally over long ranges for its possible partners, then slowly reduce the range to refine the search to more and more local scales. The idea is illustrated in Figure 2.13.

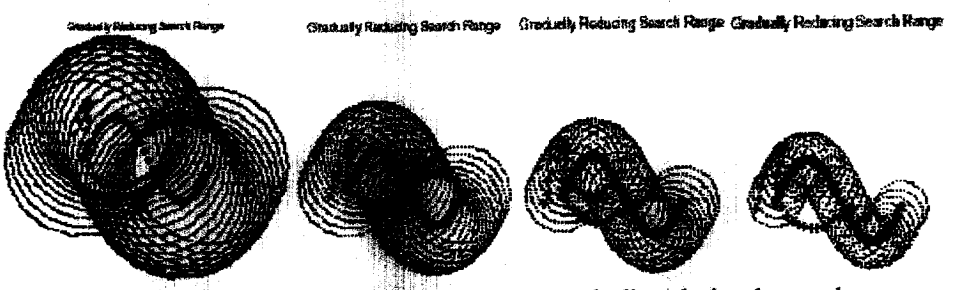

Figure 2.13 A global-to-local search strategy [13]. By gradually reducing the search range, a point can first search global for all possible partners and slowly refine the matching. Two point-sets are shown in each panel. Circles are placed around the point-set, to which the transformation is being applied (V), to indicate the current scale of search range. From left to right, the search range is gradually decreasing.

2.8.3 Deterministic Annealing

Deterministic Annealing (DA) [18] is done by adding an entropy term in the form of $H = \sum_{i=1}^{N}\sum_{a=1}^{K} m_{ai} \log m_{ai}$ to the original assignment energy function (2.2). The newly introduced parameter T is called the temperature parameter.

The name "temperature" originates from the fact that as you gradually reduce T, the energy function is minimized by a process similar to physical annealing. The annealing in DA refers to the procedure of tracking the minimum of the energy function (with the entropy term) while gradually lowering the temperature. In other words, the minima obtained at each temperature are used as initial conditions for the next stage as the temperature is lowered. With the entropy term, the energy function tends to be more convex, enabling the algorithm to overcome many local minima. The influence of the newly added entropy term will gradually decrease as T approaches zero. So it is guaranteed that the minimum tracked with DA will also always be a minimum of the original energy function (but not necessarily the global minimum). Once the annealing schedule for the temperature is specified, the whole process is determined, hence the name deterministic annealing.

Some intuitive notion of the workings of deterministic annealing can be obtained from observing the effect of the entropy term in the annealing evolution. Since all m_{ai} are smaller than 1; the entropy term is minimized when all m_{ai} are equal, i.e., the correspondence is most fuzzy. At higher temperatures, the entropy term dominates the energy function. The attempt to minimize the energy function forces the

correspondence to be more fuzzy and hence becomes a factor in "convexifying" the objective function. As T is lowered, the influence of the entropy decreases and less fuzzy configurations of M are allowed. In short, the entropy term always encourages a certain level of fuzziness. The level is determined by the temperature parameter T. The larger the T, the higher the level of fuzziness. The temperature parameter T is acting as the "search range" discussed just above. The deterministic technique basically allows us to control the "search range", hence control the degree of fuzziness.

When T goes to zero, we will get back to the original energy function. At this point, the correspondence will also approach binary. This is a consequence of the following extension to the well-known Birkhoff-von Neumann theorem [7]: the set of doubly substochastic matrices is the convex hull of the set of permutation matrices and outliers. So it can be ensured that we will always achieve a one-to-one correspondence.

2.9 Iterative Closest Point (ICP)

Iterative closest point registration (ICP) is an accurate and reliable method for registration The ICP algorithm always converges monotonically to the nearest local minimum of a mean-square distance metric, and experience shows that the rate of convergence is rapid during the first few iterations. Therefore, given an adequate set of initial rotations and translations for a particular class of objects with a certain level of "shape complexity," one can globally minimize the mean-square distance metric over all six degrees of freedom by testing each initial registration. For example, a given "model" shape and a sensed "data" shape that represents a major portion of the model shape can be registered in minutes by testing one initial translation and a relatively small set of rotations to allow for the given level of model complexity. ICP is efficient, with average case complexity of $O(n \log n)$ for n point images.

Now, we will review the original algorithm proposed by Besl and Mckay [6]. For variants of the ICP algorithm, refer to Appendix A.

The ICP algorithm can be described in terms of an abstract geometric shape X whose internal representation must be known to execute the algorithm but is not of concern for this discussion. Thus, all that follows applies equally well to 1) sets of points, 2) sets of line segments, 3) sets of parametric curves, 4) sets of implicit curves, 5) sets of triangles, 6) sets of parametric surfaces, and 7) sets of implicit surfaces.

In the description of the algorithm, a "data" shape P is moved (registered, positioned) to be in best alignment with a "model" shape X. The data and the model shape may be represented in any of the allowable forms. For our purposes, the data shape must be decomposed into a point set if it is not already in point set form. The number of points in the data shape will be denoted Np. Let N, be the number of points, line segments, or triangles involved in the model shape. As described above, the curve and surface closest-point evaluators implemented in our system require a framework of lines or triangles to yield the initial parameter values for the Newton's iteration; therefore, the number N, is still relevant for these smooth entities but varies according to the accuracy of the approximation.

The distance metric d between an individual data point $\vec{p}$ and a model shape X will be denoted

$$d(\vec{p}, X) = \min_{x \in X} \|\vec{x} - \vec{p}\| \tag{2.3}$$

The closest point The closest point in X that yields the minimum distance is denoted $\vec{y}$ such that $d(\vec{p}, \vec{y}) = d(\vec{p}, X)$, where $\vec{y} \in X$. Note that computing the closest point is $O(N_x)$ worst case with expected cost $\log(N_x)$. When the closest point computation (from $\vec{p}$ to X) is performed for each point in P, that process has a worst case $O(N_p N_x)$. Let Y denote the resulting set of closest points, and let C be the closest point operator:

$$Y = C(P, X). \tag{2.4}$$

Given the resultant corresponding point set Y, the least squares registration is computed as:

$$(\vec{q}, d) = Q(P, Y) \tag{2.5}$$

The positions of the data shape point set are then updated via $P = \vec{q}(P)$.

2.9.1 ICP Algorithm Statement

The ICP algorithm can now be stated:

- The point set P with N_p points $\{\vec{p_i}\}$ from the data shape and the model shape X (with N_x supporting geometric primitives: points, lines, or triangles) are given.
- The iteration is initialized by setting $P_0 = P$, $\vec{q_0} = [1,0,0,0,0,0,0]^T$ and $k = 0$.

The registration vectors are defined relative to the initial data set P_0 so that the final registration represents the complete transformation. Steps 1, 2, 3, and 4 are applied until convergence within a tolerance τ. The computational cost of each operation is given in brackets.

a. Compute the closest points: $Y_k = C(P_k, X)$ (cost: $O(N_p N_x)$ worst case, $O(N_p \log N_x$ average).

b. Compute the registration: $(\vec{q_k}, d_k) = Q(P_0, Y_k)$ (cost: $O(N_p)$).

c. Apply the registration: $P_{k+1} = \vec{q_k}(P_0)$ (cost: $O(N_P)$).

d. Terminate the iteration when the change in mean-square error falls below a preset threshold $\tau > 0$ specifying the desired precision of the registration: $d_k - d_{k+1} < \tau$

If a dimensionless threshold is desired, one can replace τ with $\tau\sqrt{tr(\sum_x)}$, where the square root of the trace of the covariance of the model shape indicates the rough size of the model shape.

2.9.2 Convergence Theorem

The key ideas of the convergence theorem of the ICP are that 1) least squares registration generically reduces the average distance between corresponding points during each iteration, whereas 2) the closest point determination generically reduces the distance for each point individually. Of course, this individual distance reduction also reduces the average distance because the average of a set of smaller positive numbers is smaller. A more elaborate explanation in the proof is shown below.

Theorem: The iterative closest point algorithm always converges monotonically to a local minimum with respect to the mean-square distance objective function.

Proof: Given $P_k = \{\vec{p}_{ik}\} = \vec{q}_k(P_0)$ and X, compute the set of closest points $Y_k = \{\vec{y}_{ik}\}$ given the internal geometric representation of X. The mean squared error e_k of that correspondence is given by

$$e_k = \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|\vec{y}_{ik} - \vec{p}_{ik}\right\|^2 \tag{2.6}$$

The Q operator is applied to get $\vec{q}_k$ and d_k from the correspondence:

$$d_k = \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|\vec{y}_{ik} - R(\vec{q}_{kR})\vec{p}_{i0} - \vec{q}_{kT}\right\|^2. \tag{2.7}$$

It is always the case that $d_k \le e_k$. Suppose that $d_k > e_k$. If this were so, then the identity transformation on the point set would yield a smaller mean square error than the least squares registration, which cannot possibly be the case. Next, let the least squares registration $\vec{q}_k$ be applied to the point set P_0, yielding the point set P_{k+1}. If the previous correspondence to the set of points Y_k were maintained, then the mean square error is still d_k, that is

$$d_k = \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|\vec{y}_{ik} - \vec{p}_{i,k+1}\right\|^2. \tag{2.8}$$

However, during the application of the subsequent closest point operator, a new point set Y_{k+1} is obtained: $Y_{k+1} = C(P_{k+1}, X)$. It is clear that

$$\left\|\vec{y}_{i,k+1} - \vec{p}_{i,k+1}\right\| \le \left\|\vec{y}_{i,k} - \vec{p}_{i,k+1}\right\| \text{ for each } i = 1, N_p \tag{2.9}$$

because the point $\vec{y}_{ik}$ was the closest point prior to transformation by $\vec{q}_k$ and resides at some new distance relative to $\vec{p}_{i,k+1}$. If $\vec{y}_{i,k+1}$ were further from $\vec{p}_{i,k+1}$ than $\vec{y}_{ik}$, then

this would directly contradict the basic operation of the C closest point operator. Therefore, the mean square errors e_k and d_k must obey the following inequality:

$$0 \le d_{k+1} \le e_{k+1} \le d_k \le e_k \text{ for all } k. \tag{2.10}$$

The lower bound occurs, of course, since mean-square errors cannot be negative. Because the mean-square error sequence is nonincreasing and bounded below, the algorithm must converge monotonically to a minimum value.

Experimentally, it was found that fast convergence during the first few iterations that slows down as it approaches the local minimum. Even at this slow pace, somewhere between 30 and 50 iterations yields excellent results: $d_k \approx 0.1\%$ of model shape size. Next we will describe the advantages and disadvantages of the ICP algorithm as stated by Besl and Mckay [6].

2.9.3 The advantages of the ICP matching algorithm are as follo ws:

- It handles the full six degrees of freedom.
- It is independent of shape representation.
- The surface patch or curve segment partitioning of parametric or implicit entities is essentially ignored by the matching procedure. This is important for using CAD data in its native form without elaborate user-guided preprocessing.
- It does not require preprocessing of 3-D point data, such as smoothing, as long as the number of statistical outliers is near zero. This is usually the case with accurate noncontact sensors used for inspection purposes.
- It does not require any derivative estimation or any local feature extraction.
- Almost all aspects of the algorithm are ideally suited for coarse-grain or fine-grain parallel architectures. For large problems, even remote execution procedures and distributed file systems on networks of workstations can provide worthwhile speedup without significant overhead.
- Global matching is achieved at predictable cost based on shape complexity.
- Local matching is achieved at predictable cost based on shape complexity and the percentage of allowable occlusion.

- It can handle a reasonable amount of normally distributed vector noise, with standard deviations of up to 10% of object size.
- It can easily be used in conjunction with other algorithms, such as the covariance matrix alignment, which preorient the data so that fewer initial rotation states are necessary.
- Shapes with three sufficiently distinct principal moments (eigenvalues) can be globally matched at a cost of only four initial rotation states.
- It is relatively insensitive to minor data segmentation errors as indicated by the performance of the registration of points with the African mask parametric surface model.
- The results of the last iteration of closest point registration can be used directly as inspection results since the distance to the closest point on a surface is computed as a byproduct.

2.9.4 The disadvantages of the ICP matching algorithm are as follows:

- It is susceptible to gross statistical outliers unless a statistically robust method is substituted at some point (either in preprocessing or registration computation) for outlier detection.
- The cost of local matching can get quite large for small allowable occlusion percentages, e.g., 10% or less. It is not recommended to use the ICP method if feature extraction techniques will successfully solve the problem.

- As an extension of the outlier rejection issue, the stated algorithm does not solve the segmentation problem, of course. If data points from two shapes are intermixed and matched against the individual shapes, the registrations will be wrong, and the mean square distance metric will be large. This is a problem with almost all of the shape matching algorithms in the literature.
- For any given fixed initial set of rotations, the global shape matching capability can be defeated even without sensor noise by constructing "sea urchin" or "planetoid" shaped objects based on the set of rotations such that correct registration cannot be guaranteed. On the other hand, for a fixed set of objects and no sensor noise, one can determine an initial set of registrations in a finite amount

of time such that one can guarantee registration in a finite amount of time with a sufficiently small probability of error.

- In the limit of very complicated sea urchins or perfectly spherical planets with a single 1 μm bump or in the limit of very localized matching (1% of object shape or less) on any object, the ICP algorithm degenerates to brute-force 3-D template matching. Feature extraction techniques, if possible, are preferable in such circumstances.

2.10 Discussion

After reviewing most of the related previous work, we see that the "non-rigid" point matching has remained largely unresolved. Most of the matching methods available only solve for rigid transformations. Improving upon the independent approaches still an open area of research. This is what motivated us to enhance the ICP algorithm as it is still the most robust reliable algorithm available, to be used for the non-rigid point matching problem. Although the original algorithm and all its variants were mainly used for the registration of rigid body transformations, none of them mentioned the use of ICP with non-rigid bodies. As they assume that there is an adequate set of initial poses available in case of rigid transformation, such an assumption is no longer valid in case of non-rigid transformations specially when the deformation is large.

CHAPTER 3

A New Non-rigid Point Matching Algorithm

Based on the critique and analysis used in the previous chapters, Non rigid point matching is an important problem as non rigid transformations are needed for image registration tasks for deformable objects. Analyzing non-rigid motion has become a major research problem not only in computer vision and image processing but also, more dominantly, in medical imaging, where most of the objects being studied is deformable. As demonstrated in the previous chapters, the few existing algorithms that are capable (to different extents) of handling the non-rigid point matching problem have various shortcomings.

Consequently the objective of the thesis is to develop a general-purpose non-rigid point matching algorithm which is also well suited to real-world image registration tasks. The algorithm is based on the iterative closest point algorithm (ICP), as for the methods based on the ICP algorithm; the initial positions of the point sets need to be close enough for the registration to work properly. So if they are deformed heavily, its performance will degenerate quickly as it will be trapped in local minima and cannot be recovered to get the most accurate results, so to improve its performance, it would be a good idea if we entered the data partially transformed (although not very accurate). This can generate best results with respect to accuracy, and can be achieved through the center of mass algorithm.

ICP relies on experimental heuristics to handle outliers. For example, outliers are rejected through a dynamic thresholding mechanism [15], which depends on the task at hand. Distances between nearest point pairs are first fit to a Gaussian distribution. Points with distance measure values larger than the mean plus L (usually set to 3) times the standard deviation are then rejected as outliers.

3.1 Incorporation of Different Transformations

Different models of transformation have their different properties and, hence are suitable for different applications. An algorithm's ability to accommodate different transformation models can make it a general tool for many problems. Our algorithm is designed with this in mind. At this point, the ICP algorithm is a general framework, i.e., any specific non-rigid transformation can be put in to replace the general notion of the algorithm. As an example, here we discuss the incorporation center of mass algorithm as well as one specific non-rigid transformation parameterization —the thin plate spline (TPS) [8].

3.1.1 Center of mass

This method originated in physics [1]. The idea is to bring the centers of mass of the two point sets to coincide to correct translational mismatch. The centers of mass of the two point sets are to coincide in order to achieve rough translational correction before any alignment is performed. The center of mass $(\hat{x}, \hat{y})$ of an image I is computed as

$$\hat{x} = \sum_{i,j} g_{ij} i \Big/ \sum_{i,j} g_{ij} \qquad g_{ij} \in I \qquad (3.1)$$

$$\hat{y} = \sum_{i,j} g_{ij} j \Big/ \sum_{i,j} g_{ij} \qquad g_{ij} \in I \qquad (3.2)$$

where g_{ij} is the gray scale value of the image pixel at the location (i,j). The pixels corresponding to the background of the image are set to zero and so they do not affect the computation of the center of mass. The center of mass provides us with the information about the global location of the data. The result is not very accurate but it is automatic, very fast and easy in implementation and would prevent ICP from being trapped in local minima. We will use the term CG to stand for Center of Mass (sometimes referred to as Center of Gravity) throughout the rest of the chapter.

3.1.2 The Thin-Plate Spline (TPS) [8,39]

TPS provides us with a good example to explain the reason why we need to include a prior smoothing term for a non-rigid transformation model. As most other splines, TPS's main task is to generate an appropriate spatial mapping given two sets of landmark points. The constraint that these two sets of landmark points are to be mapped exactly onto each other is not strong enough to specify a unique non-rigid transformation. If we think of a non-rigid spatial mapping/transformation as a spatial displacement field, such a requirement only tells us the displacements at certain particular locations, where those landmarks happen to be at. Apart from those special locations, we need to fill in what the displacements are for the rest of the space. Since the mapping is allowed to be non-rigid, there are an infinite number of different choices. To narrow down the choices, we need to add extra information about how the non-rigid mappings should behave. More often than not, we would prefer the mapping to be somewhat "smooth", i.e., it should not distort the space too much, especially if it is unnecessary. This kind of preference can be enforced via a prior penalty term that discourages mappings that are too arbitrary. Since the penalty is often measured by some kind of "smoothness" of the mapping, we call it the "smoothness prior/constraint/measure". We can control the behavior of the mapping by choosing a specific smoothness measure, which basically reflects our prior knowledge on the transformation.

One of the simplest forms of the smoothness measures is the space integral of the square of the second order derivatives of the mapping function. More formally, we are looking for a mapping function $f(va)$ between corresponding point-sets $\{ya\}$ and $\{va\}$ that minimizes the following energy function:

$$f_{TPS} = \arg\min_{f} E_{TPS}(f) \tag{3.3}$$

$$= \arg\min_{f} \left(\sum_{a=1}^{K} \| y_a - f(v_a) \|^2 + \lambda \int\int [(\frac{\partial^2 f}{\partial x^2})^2 + 2(\frac{\partial^2 f}{\partial x \partial y})^2 + (\frac{\partial^2 f}{\partial y^2})^2] dx dy \right) \tag{3.4}$$

Suppose the points are in 2D (D=2). We use *homogeneous* coordinates for the point-set where a point y_a is represented as a vector (1, y_{ax} ,y_{ay}). With a fixed weight parameter λ, there exists a unique minimizer f : We call it the thin plate spline (TPS). This minimizer is parameterized by α which comprises two matrices d and c, (α={d, c}).

$$f_{TPS}(x,\alpha) = f_{TPS}(x,d,c) = x \cdot d + \sum_{b=1}^{K} \phi(\|x - v_b\|) \cdot c_b \tag{3.5}$$

Where d is a $(D + I)$ x $(D + I)$ matrix representing the affine transformation and, c is a K x $(D + I)$ warping coefficient matrix representing the non-affine deformation. The **kernel function** $\phi(\|x - v_b\|)$ **is a 1 x** K **vector for each point** x, where each entry $\phi_b(x) = \|x - v_b\|^2 \log \|x - v_b\|$.

If we substitute the solution for f(3.5) into (3.4), the TPS energy function becomes,

$$E_{TPS}(d,c) = \|Y - V_d - \Phi_c\|^2 + \lambda \operatorname{trace}(c^T \Phi_c) \tag{3.6}$$

Where Y and V are just concatenated versions of the point coordinates y_a and v_a, and Φ is a (K x K) matrix formed from the $\phi(\|v_a - v_b\|)$. Each row of each newly formed matrix comes from one of the original vectors. The matrix Φ represents the TPS *kernel* . Loosely speaking, the TPS kernel contains the information about point-set's internal structural relationships. When it is combined with the warping coefficient c, a non-rigid warping is generated.

A nice property of the TPS is that it can always be decomposed into a global affine and a local non-affine component. Consequently, the TPS smoothness term in 3.4 is solely dependent on the non-affine components. This is a desirable property, especially when compared to other splines, since the global pose parameters included in the affine transformation are not penalized.

3.1.2.1 Closed Form Solution for TPS

The separation of the affine and the non-affine warping space is done through a QR decomposition [39].

$$V = [Q_1 \mid Q_2] \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{3.7}$$

Where Q_1 and Q_2 are K x $(D$ x $1)$ and N x $(K - D - 1)$ orthonormal matrices, respectively. The matrix R is upper triangular.

With the QR decomposition in place, (3.6) becomes,

$$E_{TPS}(\gamma, d) = \left\| Q_2^T Y - Q_2^T \Phi Q_2 \gamma \right\|^2 + \left\| Q_1^T Y - Rd - Q_1^T \Phi Q_2 \gamma \right\|^2 + \lambda \gamma^T Q_2^T \Phi Q_2 \gamma \tag{3.8}$$

Where $c = Q_2\gamma$ and γ is a $(K - D - 1)$ x $(D + 1)$ matrix. Setting $c = Q_2\gamma$ (which in turn implies that $V^T c = 0$) enables us to cleanly separate the first term in (3.6) into a non-affine term [first and second terms in (3.8) respectively].

The least squares energy function in (3.8) can be first minimized with respect to γ and then with respect to d. The final solution for c and d are,

$$\hat{c} = Q_2 (Q_2^T \Phi Q_2 + \lambda I_{(K-D-1)})^{-1} Q_2^T Y \tag{3.9}$$

$$\hat{d} = R^{-1} (Q_1^T Y - \Phi \hat{c}) \tag{3.10}$$

We call the minimum value of the TPS energy function obtained at the optimum $(\hat{c}, \hat{d})$ the "bending energy":

$$E_{bending} = \lambda \operatorname{trace}\left[Q_2 (Q_2^T \Phi Q_2 + \lambda I_{(K-D-1)})^{-1} Q_2^T Y Y^T \right]. \tag{3.11}$$

3.2 ICP Pseudo-code:

We normally start the algorithm's alternating update process by setting the transformation parameters α to be zeros (so that the transformation is an identity transformation and points and points stay at their original place). Then apply center of mass to make the centers of mass of the two point sets to coincide. Then we run the correspondence update and the transformation at each iteration, until optimum solution is reached or maximum number of iterations is reached.

The ICP Algorithm Pseudo-code

Initialize TPS parameters α (e.g. α =0, Identity matrix).

Bring the centers of mass of the two point sets to coincide.

Begin A: Iterative Matching

Begin B: Alternating Update

Step I: Update correspondence parameter *M* based on current transformation parameter α based on the closest neighbor criteria (with outlier rejection).

Step II: Update transformation parameter α based on current correspondence parameter *M*.

End B

Until convergence or maximum number of iterations is reached

End A

3.3 Experiments

We used in our experiment Synthetic data developed by Dr. Haili Chui in the University of Yale. Although they are 2D point sets, yet they can be used as an initial step to test the algorithms used. They represent non- rigid point matching examples. These examples will provide a careful evaluation of the algorithm and a comparison to previous algorithms such as RPM.

We then set out to evaluate the algorithm's performance or efficiency using the number of iterations using these synthetic data.

The ground truth (for both the correspondence and the transformation) is known for the synthetic data so that we can carry out quantitative evaluations of the algorithm. We conducted 1170 synthetic experiments covering three templates and comparing our algorithm with a similar algorithm but without the contribution of the center of mass algorithm and also comparing to the RPM algorithm.

We present some typical examples from these synthetic experiments and error statistics as a measure of evaluation of different algorithms to be compared.

3.3.1 A simple example

We start with a simple case in 2D point matching. Two point sets are shown in Figure 3.1.

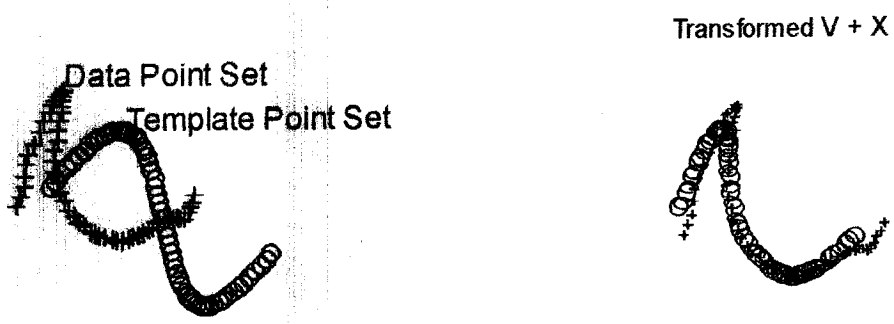

Figure 3.1 A simple 2D example. **Left:** Initial position of two point sets V (circles) and X (crosses). **Right:** Final position using our algorithm.

Global matching using center of mass occurs first and the output is fed into the ICP algorithm with the TPS mapping. What we are doing is basically trying to deform the template point set to fit the data point set.

3.3.2 Setup of the synthetic experiments

As we mentioned before, we used the synthetic data made by Dr. Haili Chui. Three different templates used here are shown in Figure 3.2. The first template is a simple one, which can be a footprint, while the second template comes from the

outer contour of a tropical fish. It mainly consists of a closed contour with some very sharp corners. To define a continuos curve, those sharp corners may pose some challenges because geometrical measures such as the curvature are discontinuous at those locations. The third comes from a Chinese character. It is a more complex pattern with multiple curve like strokes that may or may not be connected to each other.

Modeling such a complex pattern with high level feature representation, such as curve, might be quite messy, while the simple point representation can avoid these problems all at once.

We then used both ICP with and without the center of mass as a preprocessing step to found the TPS mappings that warp the template set onto the target set. And we compared to the RPM algorithm as well. The errors are computed as the root mean squared distance between the point sets.

Note that although we know the ground truth correspondence between the template and data point sets; such information is not being used in matching. We intend to recover both the correspondence as well as the non-rigid transformation through our point-matching algorithm.

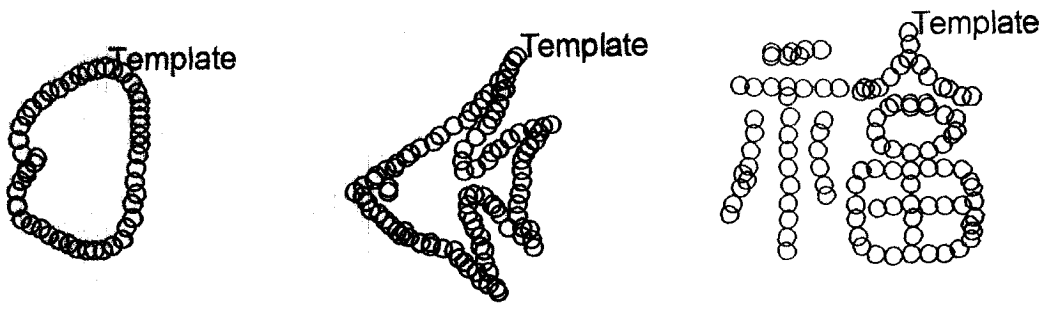

Figure 3.2 Template point sets for synthetic experiments

We conducted a series of experiments on each type of templates. In each series we changed the number of iterations to be used and we repeated each of those series ten times to take the average value of the RMS error (root mean square).

3.3.3 Results of the synthetic experiments

After we explained how the synthetic experiments are set up, we now examine the results by looking at some examples and the error statistics.

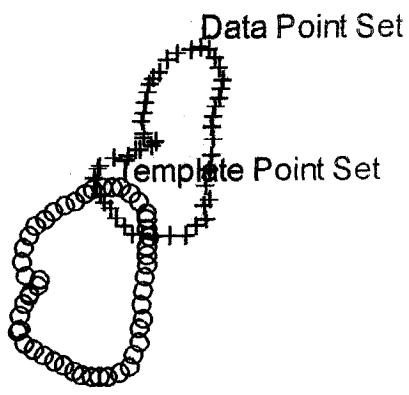

Figure 3.3 Original point sets of the first template

The following is Table 3.1 showing the results using the different techniques used in the experiments on the template in Figure 3.3.

Number of iterations	RMS error of ICP with TPS	RMS error of ICP with TPS and CG	RMS error of RPM
10	0.122342	0.067810	0.199179
15	0.099882	0.056730	0.203814
20	0.095666	0.056763	0.206080
25	0.089309	0.056377	0.202850
30	0.086253	0.055964	0.164970
35	0.086128	0.055385	0.139995
40	0.085989	0.054845	0.126533
43	0.085908	0.054419	0.104823
46	0.085842	0.054111	0.092452
49	0.085492	0.053402	0.083868
51	0.084719	0.053216	0.077455
53	0.084471	0.053070	0.070130

Table 3.1 Results of different algorithms with the first template

The graph representing the RMS error with respect to the number of iterations, is shown in Figure 3.4.

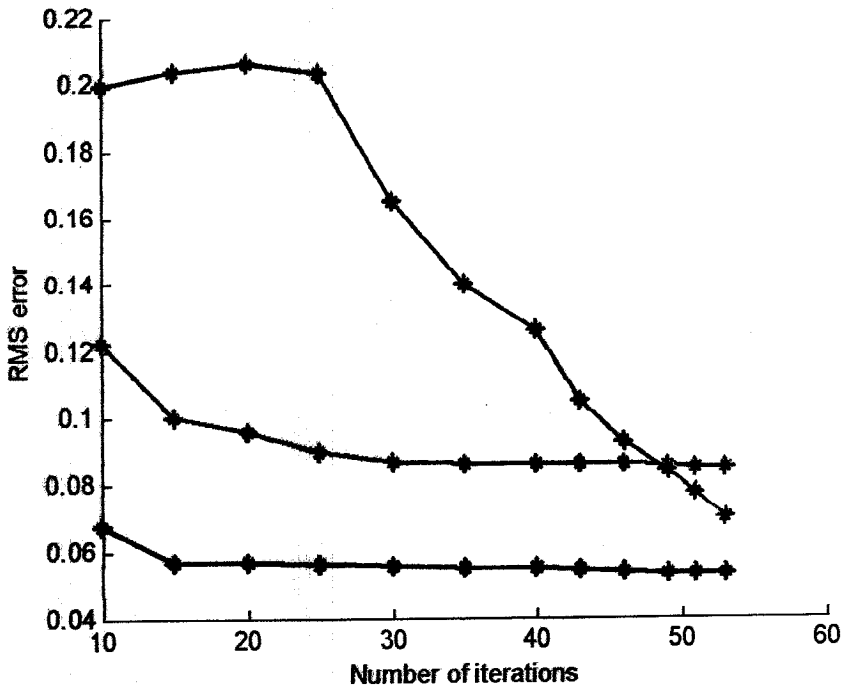

Figure 3.4 A graph showing the results in Table 3.1

—— ICP with TPS

—— ICP with TPS and CG

—— RPM

As can be seen from the graph, the best results with respect to RMS error are for our algorithm. Although RPM is working well at the highest number of iterations, but it is not reliable in lower stages.

Now we will repeat the experiments to the second template shown in Figure 3.5.

If we applied ICP with TPS mappings on the two point sets, the result would be poor as the RMS error was computed to be 0.190238 using 53 iterations (normally, an error

beyond 0.1 indicate a poor matching). As ICP was trapped in local minima as shown in Figure 3.6.

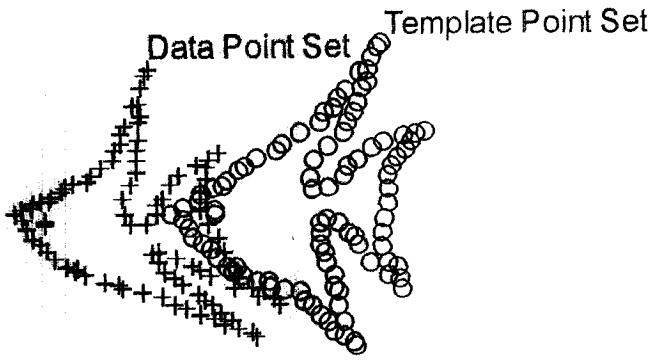

Figure 3.5 Original point sets of the second template

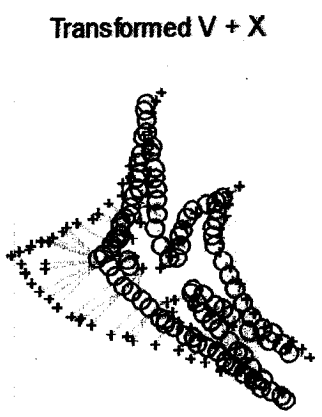

Figure 3.6 ICP with TPS using 53 iterations

If we apply center of mass, to the original point sets, they coincide and hence rough alignment is produced as shown in Figure 3.7.

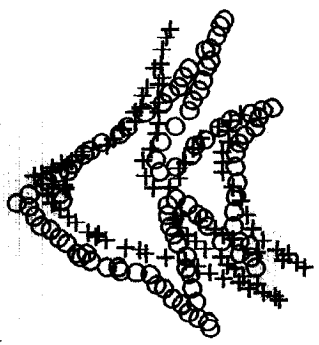

Figure 3.7 Center of Mass applied to the original point sets in Figure 3.5

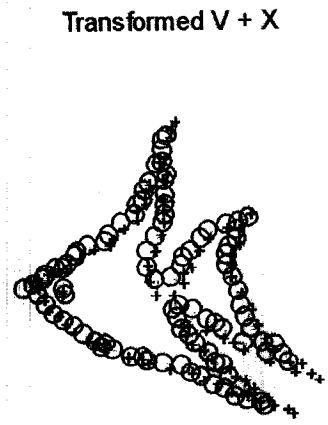

Figure 3.8 ICP with TPS and CG using 53 iterations

In Figure 3.8 we applied ICP and TPS using 53 iterations on the output shown in Figure 3.7. So the two point sets were nearly "exactly matchable".

Table 3.2 shows the results with respect to the second figure, shown in Figure 3.5. It gives an overall picture for all the three algorithms' performance or efficiency against the number of iterations.

Number of iterations	RMS error of ICP with TPS	RMS error of ICP with TPS and CG	RMS error of RPM
5	0.316173	0.095008	0.393020
10	0.222866	0.082544	0.351857
15	0.196807	0.077977	0.252412
20	0.187654	0.077150	0.171334
25	0.189114	0.076168	0.115613
30	0.190221	0.075458	0.086319
35	0.190566	0.070439	0.072572
40	0.189749	0.069702	0.070350
43	0.189753	0.068793	0.068256
46	0.190637	0.068291	0.066224
49	0.190739	0.067793	0.061299
51	0.190404	0.067439	0.059170
53	0.190238	0.066996	0.057871

Table 3.2 Results of different algorithms with the second template

The graph showing the RMS error with respect to the number of iterations used in each algorithm is plotted in Figure 3.9.

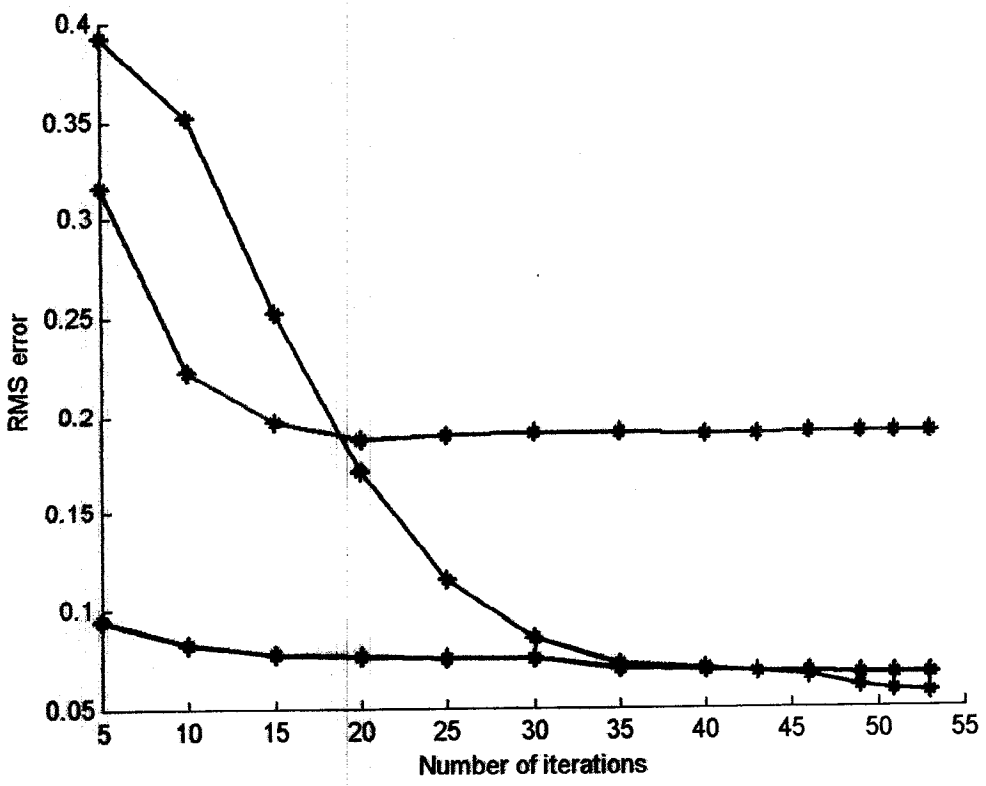

Figure 3.9 A graph showing the results in Table 4.2

ICP with TPS

ICP with TPS and CG

RPM

Similarly, by analyzing the graph, we can see that using CG with ICP and TPS mappings, makes the algorithm work efficiently after a few iterations, however the RPM algorithm does not produce any reasonable results before 35 iterations. Also we notice the wide gap between the efficiency of the ICP with TPS alone relative to our algorithm, although they behave similarly but the accuracy of our algorithm is much better through all the iterations.

Although RPM produces better results after 45 iterations yet we should note that, each iteration in RPM is a composite iteration as it is further decomposed into a number of iterations according to the setting of the temperature in the annealing process (refer to

section 2.8.3). And this can be revealed through the time taken to finish each iteration so we will include some other tables related to time later in this section.

Now we will examine our algorithm on the third template shown in Figure 3.10 which is the most complex point set as it is composed of 105 points.

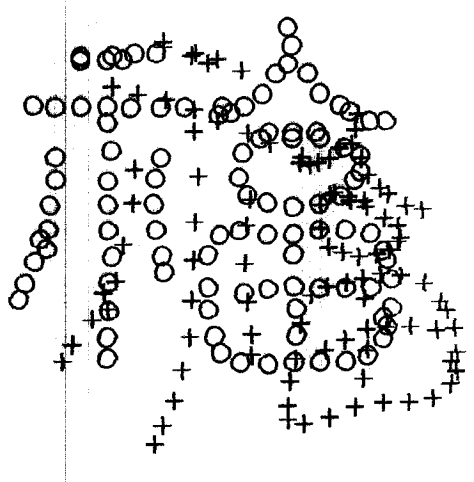

Figure 3.10 Original point sets of the third template

By applying ICP with TPS mappings, using 53 iterations, the result will be poor as in Figure 3.11. with 0.089232 RMS error at this snap shot. Note that we carried out the same experiment ten times to take the average and used that average in Table 3.3.

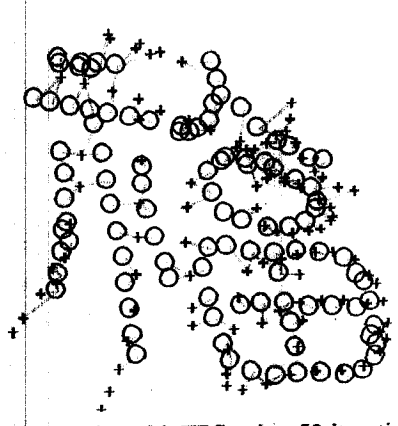

Figure 3.11 ICP with TPS using 53 iterations

By applying the center of mass algorithm to the original point sets, the result will be rough alignment as shown in Figure 3.12.

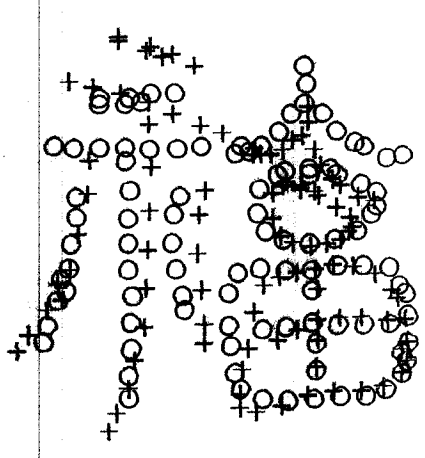

Figure 3.12 Center of Mass applied to the original point sets in Figure 3.10

Now if we applied ICP and TPS on the output of Figure 3.12, the result will be as shown in Figure 3.13 with 53 iterations and 0.054924 RMS error, which is better than the result of applying ICP and TPS mappings on the original point sets directly.

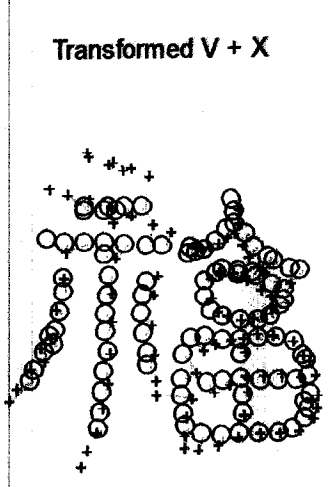

Figure 3.13 ICP with TPS and CG using 53 iterations

The results of different algorithms applied on the third template in Figure 3.10 is shown in Table 3.3, which is consistent with the analysis extracted from the previous tables and graphs.

Number of iterations	RMS error of ICP with TPS	RMS error of ICP with TPS and CG	RMS error of RPM
5	0.134906	0.065367	0.255769
10	0.115658	0.059264	0.262750
15	0.110010	0.059114	0.266250
20	0.100553	0.058931	0.262324
25	0.088624	0.058541	0.213520
30	0.086921	0.058169	0.145293
35	0.087083	0.057662	0.095455
40	0.086233	0.056919	0.082713
43	0.086169	0.056939	0.077772
46	0.086857	0.056445	0.073500
49	0.086694	0.055602	0.069766
51	0.086318	0.055276	0.060945
53	0.086036	0.054920	0.052823

Table 3.3 Results of different algorithms with the third template

By plotting the results in Table 3.3, we obtain the graph in Figure 3.14.

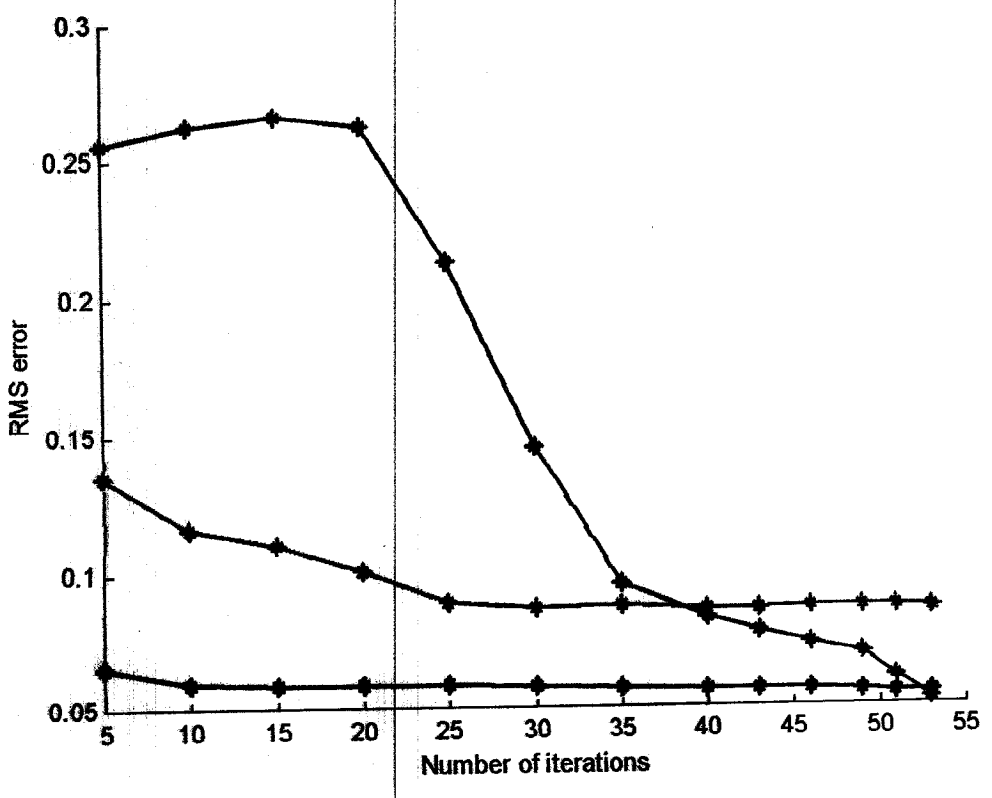

Figure 3.14 A graph showing the results in Table 3.3

—— ICP with TPS

—— ICP with TPS and CG

—— RPM

As the third template is the most complex template with respect to both the type of curvatures or discontinuities and the number of points, the results show that ICP can be a robust technique for solving the non- rigid point matching problem.

Also we conducted some experiments using time as a parameter instead of the number of iterations, although we cannot rely on those values in our analysis, but it can be used only as a rough indicator to the complexity of each algorithm. We carried out those experiments on a Pentium II 300 MHz CPU. Also we should consider the delay due to the GUIs in each experiment but those GUIs are equal in time in all the cases, the results of running the three algorithms on the three templates for 23 and 42 iterations are shown in Tables 3.4 and 3.5, respectively.

Template	First (50 points)	second (98 points)	Third (105 points)
RMS with ICP & TPS / Time	0.092297/ 10 sec	0.188997/ 15 sec	0.100549 / 20 sec
RMS with ICP, TPS and CG / Time	0.067412/ 11 sec	0.076493/16 sec	0.058724 / 21 sec
RMS with RPM/ Time	0.204712/ 40 sec	0.123672/145 sec	0.230337 /180 sec

Table 3.4 RMS error and Time of Different algorithms with 23 iterations

Template	First (50 points)	second (98 points)	Third (105 points)
RMS with ICP & TPS / Time	0.085944/ 16 sec	0.189762/ 30 sec	0.086620/ 35 sec
RMS with ICP, TPS and CG / Time	0.062526 / 17 sec	0.069202/ 31 sec	0.057246/ 36 sec
RMS with RPM/ Time	0.109486/ 65 sec	0.060518 / 230 sec	0.058632 /305 sec

Table 3.5 RMS error and Time of Different algorithms with 42 iterations

3.4 Discussion

We conducted 1170 experiments using three different templates that represent non-rigid point matching examples which can be a sample to the real world data of non-rigid bodies, as one of them was a simple one and the other two constitutes various complexities. Although we knew in advance the true correspondence between the template and data point sets, such information was not used in the matching experiments, as we recovered both the correspondence as well as the non-rigid transformation through our point matching algorithm.

The results from the above experiments were evaluated according to two criteria: accuracy and speed of convergence. As these are synthetic data, we measured the accuracy using root mean square error (RMS). The other measure of comparing the different registration algorithms is the speed of convergence. This is simply the number of iterations needed to achieve convergence. We used two algorithms as benchmarks ICP with TPS mapping and RPM. We did not use the original version of the ICP algorithm, as it won't be a fair judgment to use an algorithm for rigid bodies on non-rigid point sets

Experimental results demonstrate that our algorithm prevents the ICP to a great extent from being trapped in a local minimum. Also, it makes the ICP algorithm work more efficiently after a few iterations, however the RPM algorithm did not produce any reasonable results before 35 iterations or more (According to the complexity of the point sets at hand). Even though it gave better results than the ICP algorithm after 45 to 50 iterations according to the template used, but each iteration in RPM is a composite iteration as we mentioned before, and this can be revealed easily through the tables showing the time taken by each algorithm.

Although our algorithm is expected to work better on points in a closed form, due to the usage of the center of mass algorithm, it demonstrated reasonable results with the third template, which is not in a closed form. This fact proves its reliability and that it can be used as a general-purpose algorithm.

CHAPTER 4
Conclusions

4.1 Conclusions and Contributions

Non-rigid point matching is an important problem, as non-rigid transformations are needed for image registration tasks of deformable objects. However most of the authors of the non-rigid matching algorithms are not looking to minimize a criterion, but their attention has rather focused on the type of smoothing or physical model of deformation to be used. In this thesis we developed a general purpose non-rigid point matching algorithm which is also well suited to real world registration tasks.

Point matching becomes much more difficult when the transformation to be estimated is non-rigid rather than rigid. We have demonstrated in this thesis that the non-rigid point matching can be solved by ICP after incorporating some other algorithms such as the center of mass algorithm and, the Thin Plate Spline (TPS). This new algorithm is capable of estimating both the correspondence and the transformation between two sets of points, and it prevents the ICP algorithm from being trapped in a local minimum as the basic algorithm. Also this new algorithm can produce reliable results as it stabilizes after a few iterations relative to the other algorithms.

Synthetic test data used in the experiments were of different natures, the first of them was smooth and the second had very sharp corners, while the third was a complex one with multiple curve like strokes, that are connected in some cases and disconnected in others. Also the first two were in closed form while the third was not in a closed form. This variety guarantees the reliability of our algorithm with all types of data and they represent a good sample of the real world.

Image intensity based registration and feature based registration have long been regarded as two totally different approaches. This does not have to be so anymore. That is why we are using an intensity-based method, which is the center of mass with the point feature as a representation, and together they produced better results.

4.2 Future Work

There is still much room for further improvement. Better feature representation can definitely be explored. While the point feature representation has the advantage of being simple and flexible, more detailed shape information (e.g. local or global shape context) would certainly enhance the representative ability and make the matching problem easier. Also implementing the algorithm on real examples, and adding some obstacles like noise and outliers and monitoring their behavior would be worth further work.

Image intensity based registration and feature based registration need not be regarded as two totally different approaches. As a digital image is treated as nothing more than a set of intensity values defined over a regular grid of pixels, the image gray level intensity is just a third dimension. The algorithm used in this thesis, which is originally designed for feature based registration, can be easily extended to intensity based registration as well, in which case center of mass can be more effective. The possible merging of these two seemingly different registration fields may invoke a lot of other innovations that can greatly benefit the research fields of medical imaging and computer vision.

Further work is also needed to find better transformation models for the non-rigid deformations. Most current deformation models used are very generic and not problem specific. Physically based information is needed to build more realistic models. However, such information can hardly be captured during image acquisition. If experimental faces about the objects' properties when in motion are known, these can certainly be used to build better transformation models.

Finally, our algorithm has been developed as a general framework. It has enormous potential applicability to a variety of registration problems in medical imaging as well as shape matching and recognition problems in computer vision. We hope that some of this potential can be realized in the years to come.

Appendix A

ICP Variants

A.1 ICP using Invariant features (ICPIF)[33]

A.1.1 Notation

The term *ICP using invariant features* (ICPIF) is used to describe the use of invariant features in a modified distance function for correspondence selection. Each data point is represented as the concatenation of its three positional coordinates with k feature coordinates. Points are matched using the L_2 norm in the k+3 dimensional space. The positional components shall be denoted P_c and its feature components $(p_i)_f$. That is,

$$p_c = (p_x, p_y, p_z) \in \mathrm{R}^3$$
$$p_f = (p_{f_1}, p_{f_2}, ..., p_{f_k}) \in \mathrm{R}^k.$$
$$p = (p_c, p_f) \in \mathrm{R}^{3+k}.$$

Where p_{f_1} through p_{f_k} are the k invariant features describing point **p**. When necessary, the notation $(p_i)_e$ and $(p_i)_f$ will be used to refer to the (Euclidean) positional and features components of point p_i. The combined positional and feature distance between **p** and **q** shall be denoted as

$$d(p,q) = d_e(p,q) + d_f(p,q). \qquad \text{(A.1)}$$

where

$$d(p,q) = \|p-q\|^2$$
$$d_e(p,q) = \|p_e - q_e\|^2$$
$$d_f(p,q) = \|p_f - q_f\|^2$$

The weighted feature distance is defined as

$$d_\alpha(p,q) = d_e(p,q) + \alpha^2 d_f(p,q) \qquad \text{(A.2)}$$

Where α controls the relative contribution of the positions and features. The closest point in M, which is the geometric model to a scene point s according to the distance

measure d_e shall be denoted CP(s, M), and the closest point according to the distance measure d_α shall be denoted $CP_\alpha(s, M)$.

A.1.2 ICPIF Algorithm

The ICPIF algorithm performs ICP using closest point correspondences using $CP_\alpha(s, M)$. At this point, we shall assume that the user has heuristically selected an appropriate value for α.

Algorithm (Iterative Closest Point Registration using Invariant Features)

Let S be a set of N_s points, $\{s_1,...,s_{N_s}\}$, and let M be the model.

1. Let T_0 be an initial estimate of the transformation.
2. Repeat for $k = 1..k_{max}$ or until termination criteria are met.
   (a) Build up the set of correspondences $C = \bigcup_{s_i \in S}\{(T_{k-1}(s_i), CP_\alpha(T_{k-1}(s_i), M))\}$
   (b) Compute the new transformation T_k that minimizes mean square error between point pairs in C.

A.1.3 Analysis of ICPIF

Theoretical results show that ICPIF chooses the correct pointwise correspondences at least as well as ICP for all possible scenes, and better than ICP for most scenes. In addition, the property of monotonic convergence to a local minimum is preserved. Experimental results on real and synthetic images suggest that ICPIF converges to the ground truth registration in fewer iterations than traditional ICP, and that ICPIF is much less likely to be trapped in a local minimum compared to traditional ICP. Although ICPIF converges in fewer iterations, there is additional overhead involved in searching for nearest neighbors in the higher dimensional space.

A.2.1.1 SIC-range of 2D objects

For 2D objects, the configuration space is three dimensional. A configuration c_i is defined for example by the coordinates (x_i,y_i,ω_i) referring to the translation (x_i,y_i), and relative orientation ω_i of the two objects.

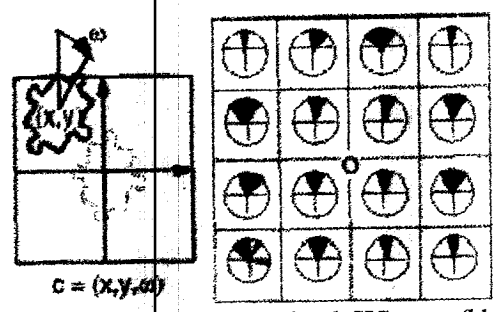

Figure A.1 Configuration (x,y,ω) and associated SIC-range (black sectors) [25]

To assess the SIC-range of the puzzle of Figure A.1, we consider sixteen translations of the test in a regular grid around the model and, for each translation, 36 evenly spaced relative rotations ω. Test and model represent the same object, i.e. $f(c)=f_{mm}(x,y,\omega)$. Figure A.1 plots the obtained SIC-range. The center of each circle indicates the translation. The black sectors at every grid position indicate the angles ω for which the matching is successful.

We observe that the SIC-range is limited in the rotation range ω and that translation between two puzzles to be matched is of minor influence. Very roughly, and for small initial translations, the SIC-range of this puzzle can be described by $\omega \in [+30, -30]$ degrees.

A.2.1.2 SIC-range of 3D objects

With 3D objects, the configuration space C is 6-dimensional and so is the SIC-range. The analysis of this general case is not simple. On one hand, limited computational resources impose a limited number of configurations to be tested, and do not allow a dense cover of C. On the other hand, a representation and a fortiori an interpretation of a 6-dimensional SIC-range is not simple, anyhow. Therefore, we search for a

meaningful SIC-range representation of lower order. The space of initial configurations can then be defined by the triple (φ,θ,ω) where φ and θ are respectively zenith and azimuth angles of the view axis in the model spherical reference system and ω designates the camera rotation angle around this axis.

Starting from 3D model and test objects, we assess their SIC-range by exploring the (φ,θ,ω) space at discrete locations, testing ICP for successful matching and setting $f(\varphi,\theta,\omega)$ accordingly.

Being 3-dimensional, the SIC-range can now be visualized by a map similar to the one used for 2D objects. This SIC-map is defined in Figure A.2 and examples are shown in Figure A.3. The small circles span the (φ,θ) space of the spherical coordinates. Each circle holds for the (φ,θ) position of the view axis and represents by its black sector the associated SIC-subrange for w. The spherical coordinate system is projected on a plane tangent to the pole. View points with same zenith angle lie on the same circle around the pole and those with same azimuth lie on the same radius. View points from the lower hemisphere are omitted.

The SIC-map assessment includes several steps. Start is from the pole configuration, which is the pose obtained after a first successful match of the model and test.

From this pole configuration, the complete $(\varphi,\theta),\omega)$ parameter space is explored in discrete steps varying azimuth angle θ from 0 to 360, zenith angle θ from 0 to 180 and view axis rotation angle ω from 0 to 360 degrees. This procedure orientates the test at different view points and then rotates it around the view axis.

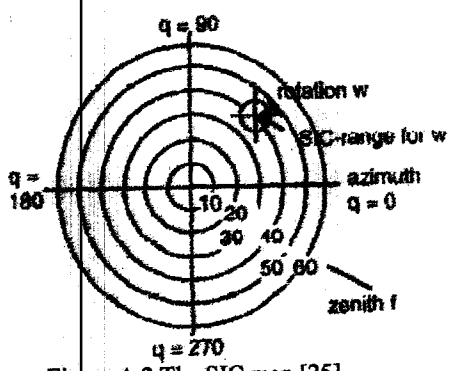

Figure A.2 The SIC map [25]

The exact sequence brings the pole axis to the spherical coordinates (φ,θ) by a rotation of the test object by zenith angle φ, around the rotation axis, which is perpendicular to the great circle of azimuth angle θ. Then, follows the rotation ω around the view axis.

For every initial configuration the ICP matching algorithm is launched, running enough (40) iterations to ensure convergence. Note that the criteria for successful convergence can be chosen differently. It could be a comparison of pose, it can also be a decision on the final error.

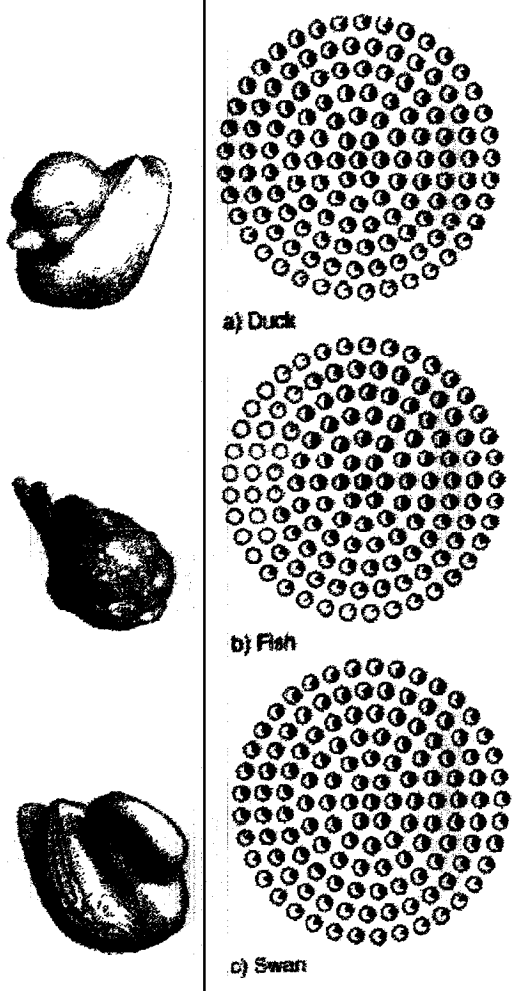

Figure A.3 Examples of some SIC-maps [25]

A.3 ICP using Z-buffers

This method is based on the ICP algorithm and on a segmentation of the sampled points in an optimized set of z-buffers [4]. This multi-z-buffer technique provides a 3D space partitioning which greatly accelerates the search of the nearest neighbors in the establishment of the point-to-point correspondence between overlapping surfaces. Then a randomized iterative registration is processed on the surface set.

Its main interest is in the first step of the algorithm where all the overlapping parts of the digitized surfaces are segmented in a set of optimized z-buffers by using Gauss spheres. This space partitioning strongly accelerates the point-to-point correspondence between all pairs of overlapping surface parts. In the second step, we use for the global registration, an iterative optimization. As for all the methods based on the ICP algorithm, the initial positions of the surfaces need to be close enough for the registration to work properly.

A z-buffer partitioning of the 3D space already greatly accelerates the determination of the point-to-point correspondence between two surfaces. However, the problem of choosing the best direction of projection for the z-buffer is not obvious when the shape of the overlapping part of the two surfaces is strongly bent. In such a case the use of a single z-buffer appears already inappropriate. This problem becomes clearly critical when we have to register a greater number of partially overlapping surfaces. Our multi-z-buffer technique provides an efficient solution to this problem. One of its principal properties is to quickly detect all the overlaps and to allow the registration process to concentrate on them even when the surfaces overlap each other only to a very small extent.

However, for the z-buffer or multi-z-buffer technique to work properly, we have to assume that the surfaces have been sampled with a sufficiently high and homogeneous density. This condition is not too strong a limitation as nowadays the available range finders provide high-density data.

A.3.1 ICP algorithm acceleration by using a z-buffer

In order to accelerate the ICP algorithm and by assuming that the density of the 3D points is homogeneous enough, the proposed method based on a twin z-buffer structure, which provides an explicit space partitioning. It is similar to two depth images, one per set of points, and allows applying 2D image-processing techniques on the 3D data. The correspondence between the two sets of 3D points can then be efficiently obtained by using the 2D connectivity of the twin z-buffer cells.

Figure A.4 Two renderings of an ancient coin before (left) and after (right) the mono-z-buffer registration [4]

To build the twin z-buffer structure, we first chose a direction of projection and two orthogonal directions. And then compute the minimal box aligned with these directions and which includes $S_1 \bigcup S_2$. The direction of projection and the width and height of the minimal enclosing box define the orientation and size of the twin z-buffers. Then partition each set by projecting its points in the cells of the associated z-buffer. And keep only one point per cell, the nearest one along the direction of projection, and store all its geometrical information. This selection allows us, by adjusting the size of the cells where data are oversampled, to save memory space and to reduce CPU time for solving the surface correspondence problem.

Then, to match a point P_1 in $cell_2(\omega, h)$ of the first z-buffer, we look for the closest point P_2 of P_1 belonging to an (n x n) window centered on $cell_2(\omega, h)$ of the second z-buffer. This way the ICP algorithm is heavily accelerated. To reinforce its accuracy and robustness, two matching criteria are added: a minimal density of points to test inside the (n x n) window and a minimal distance to be satisfied by P_1P_2.

This approach works well especially when the overlap between the two surfaces is sufficiently planar as illustrated in the example of Figure A.4. In the case of strongly curved overlaps, the experience shows that the matching algorithm does not converge properly because the digitized surfaces are not uniformly resampled in the twin z-buffer structure.

A.4 Levenberg-Marquardt Optimization for ICP (LM-ICP)

The Levenberg-Marquardt (LM) algorithm [17] is an optimization procedure, which is used to minimize the model-data fitting error via nonlinear minimization.

There now follows a derivation of the Levenberg-Marquardt algorithm. The error function $E(a)$ can be written as the sum of N_d residuals as follows:

$$E(\mathbf{a}) = \sum_{i=1}^{N_d} E_i^2(\mathbf{a}), \qquad E_i(\mathbf{a}) = \sqrt{\omega_i} \min_j \in (\|m_j - T(\mathbf{a}, d_i)\|)$$

An important concept in the derivation if LM will be the vector of residuals so that $E(\mathbf{a}) = \|e(\mathbf{a})\|^2$.

The Levenberg-Marquardt algorithm combines the gradient descent and Gauss-Newton approaches to function minimization. Using the notation above, the goal at each iteration is to choose an update to the current estimate $\mathbf{a}_k$, say x, so that setting $\mathbf{a}_{k+1} = \mathbf{a}_k + \mathbf{x}$ reduces the error $E(\mathbf{a})$. Expanding $E(\mathbf{a} + \mathbf{x})$ around a, we obtain

$$E(\mathbf{a}+x) = E(\mathbf{a}) + (\nabla E(\mathbf{a}) \cdot \mathbf{x}) + \frac{1}{2!}((\nabla^2 E(\mathbf{a}) \cdot \mathbf{x}) \cdot \mathbf{x}) + \text{h.o.t} \quad (A.3)$$

$$(\{\mathbf{m}_j\}_{j=1}^{N_m}, \{\mathbf{d}_i\}_{i=1}^{N_d}, \mathbf{a}_0)$$

$$\|e(\mathbf{a}_k)\|^2$$

Expressing this in terms of e, we have

$$E(\mathbf{a}) = \mathbf{e}^T\mathbf{e}$$
$$\nabla E(\mathbf{a}) = 2(\nabla \mathbf{e})^T\mathbf{e}$$
$$\nabla^2 E(\mathbf{a}) = 2(\nabla^2 \mathbf{e})\mathbf{e} + 2(\nabla \mathbf{e})^T \nabla \mathbf{e}$$

We shall denote that N_d x p *Jacobian* matrix ∇e by J with ij^{th} entry $J_{ij} = \frac{\partial E_i}{\partial a_j}$.

Introducing the Gauss-Newton approximation, i.e. neglecting $(\nabla^2 e)e$, we arrive at

$$E(a+x) \approx e^T e + x^T J^T e + x^T J^T J x \qquad (A.4)$$

The task at each iteration is to determine a step x which will minimize $E(a + x)$. Using the approximation to E which we have just derived, we differentiate with respect to x and equate with zero, yielding

$$\nabla_x E(a+x) = J^T e + J^T J x = 0 \qquad (A.5)$$

solving this equation for x yields the Gauss-Newton update, and gives the algorithm for one iteration of Gauss-Newton ICP:

1. Compute the vector of residuals $e(a_k)$, and its $N_d \times p$ matrix of derivatives J with respect to the components of a. (For a 2D rigid-body transformation, a has 3 components, and J is Nd × 3).
2. Compute the update $x = -(J^T J)^{-1} J e$.
3. Set $a_{k+1} = a_k + x$.

The above strategy does not guarantee that the step taken will result in a reduced error at $E(a_{k+1})$. Whether or not it does so depends on the accuracy of the second-order Taylor series expansion at a_k, and on the validity of the Gauss-Newton approximation. However, it can be shown that when these approximations are good, as they tend to be when near the minimum, convergence is rapid and reliable.

By comparison, an accelerated gradient descent approach as used by some previous registration algorithm [6] is obtained by replacing step 2 with

2. Compute the update $x = -\lambda^{-1} J^T e$.

where the value of λ controls the distance traveled along the gradient direction. For small λ, the iteration moves a long way along the downhill direction; large λ implies a short step. In contrast to Gauss-Newton, gradient descent does guarantee to reduce E, providing λ is sufficiently large. However, its convergence near the optimum is dismally slow.

The Levenberg-Marquardt algorithm combines both updates in a relatively simple way in order to achieve good performance in all regions. Step 2 is replaced by

2.Compute the update $x = -(J^T J + \lambda I)^{-1} J^T e$.

Now large λ corresponds to small, safe, gradient-descent steps, while small λ allows fast convergence near the minimum. The art of a good Levenberg-Marquardt implementation is in tuning λ after each iteration to ensure rapid progress even where the Gauss-Newton approximations are poor.

The LM-ICP algorithm, summarized as follows:

function a = **lmicp** $(\{m_j\}_{j=1}^{N_m}, \{d_i\}_{i=1}^{N_d}, a_0)$
Set λ to an initial value
Set $a = a_0$
repeat
Compute ek = e(a) —*One closest-point computation*
Compute J —p *closest-point computations*
Modify λ until $a_k = a - (J^T J + \lambda I)^{-1} J^T e_k$ reduces the error $\|e(a_k)\|^2$.
—*One or more closest-point computations*
Set $a = a_k$
until λ is large —*So only a small gradient descent step reduced the error.*

Appendix B

Experiments' code

All the experiments were carried out on MATLAB with Image Processing toolbox. Note that the code for implementing RPM as a benchmark (from Dr. Haili Chui) was embedded..

```
% Non_rigid Point Matching (ICP) Demo:
% ------------------------------
% icp_demo.m
% ------------------------------------------------------------------------
%
% Purpose:
%     1. A small GUI.
%     2. load a few non-rigid point matching examples.
%     3. run CG and ICP and RPM.
%
% Usage: [] = icp_demo;
%     1. Click the buttons in figure(2) to run the examples.
%     2. The matching process and results are displayed in figure(1).
%     3. You can resize figure(1), if it is too small.
%
% Notes: There are a total of 4 examples included to demonstrate
%        the non-rigid point matching algorithm.
% ------------------------------------------------------------------------

function [] = icp_demo (cmd_str);

global x y  frac T_init T_finalfac disp_flag m_method lam1 lam2
perTmaxit
global c d m

% Init the command
if nargin < 1
  figure(1); delete(1); % Clean up previously opened figures.
  figure(2); delete(2);
  cmd_str = 'init';
end;

if strcmp (cmd_str, 'init')
  % Init the figure windows.
  h1 = figure(1); set(gcf,'position', [10 10 600 500]);
  set(gcf, 'color', [0 0 0]);
  h2 = figure(2); set(gcf,'position', [10 10 620 40],
'menubar','none');

  % Init the command buttons.
  h_fig   = h2;
  col_ex0 = 10;
  row_ex0 = 5;
  col_ex1 = 10+100;
  row_ex1 = row_ex0;
  col_ex  = 30;
```

```
  row_ex  = 30;
  h_jnk = uicontrol (h_fig, 'style', 'text', ...
      'position', [col_ex0 row_ex0 100 row_ex], ...
      'string', 'Load Data:','fontsize',15);
  h_ex1 = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''load_ex1'');',...
      'position', [col_ex1 row_ex0 col_ex row_ex], ...
      'string', '1');
  h_ex2 = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''load_ex3'');',...
      'position', [col_ex1+(col_ex*1) row_ex0 col_ex row_ex], ...
      'string', '2');
  h_ex3 = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''load_ex4'');',...
      'position', [col_ex1+(col_ex*2) row_ex0 col_ex row_ex], ...
      'string', '3');
  h_ex4 = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''load_ex5'');',...
      'position', [col_ex1+(col_ex*3) row_ex0 col_ex row_ex], ...
      'string', '4');
% h_ex5 = uicontrol (h_fig, 'style', 'pushbutton', ...
%      'callback', 'icp_demo(''load_ex5'');',...
%      'position', [col_ex1+(col_ex*4) row_ex0 col_ex row_ex], ...
%      'string', '5');

  col_run0 = 10+100+30*5+20;
  row_run0 = 5;
  col_run1 = col_run0+100;
  row_run1 = row_run0;
  col_run2 = col_run1+80;
  row_run2 = row_run1;
  col_run  = 80;
  row_run  = 30;
  h_jnk = uicontrol (h_fig, 'style', 'text', ...
      'position', [col_run0 row_run0 100 row_run], ...
      'string', 'Run:','fontsize',15);
  h_run_cg = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''run_cg'');',...
      'position', [col_run1 row_run0 col_run row_run], ...
      'string', 'CG');
h_run_rpm = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''run_rpm'');',...
      'position', [col_run2+(col_run*1)  row_run1 col_run row_run],
...
      'string', 'RPM');


h_run_icp = uicontrol (h_fig, 'style', 'pushbutton', ...
      'callback', 'icp_demo(''run_icp'');',...
      'position', [col_run1+(col_run*1) row_run0 col_run row_run],
...
      'string', 'ICP');

  % Init all the parameters:
  load demodata_ex1;
  x = x1; y = y1;
  frac       = 1;
  T_init     = 0.5;
  T_finalfac = 500;
  disp_flag  = 1;
  m_method   = 'mix-rpm';
```

```
  lam1        = 1;
  lam2        = 0.01;
  perTmaxit   = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load data:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp (cmd_str, 'load_ex1')
  icp_demo('reset_all'); load demodata_ex1;
  x = x1; y = y1;



  demo_disp;
elseif strcmp (cmd_str, 'load_ex2')
  load demodata_ex2;
  x = x2; y = y2;


  demo_disp;
elseif strcmp (cmd_str, 'load_ex3')
   load demodata_ex3;
   x = x3; y = y3;

  demo_disp;
elseif strcmp (cmd_str, 'load_ex4')
  load demodata_ex4;
  x = x1*2; y = y2a*2;
demo_disp;


elseif strcmp (cmd_str, 'load_ex5')
   %x = load('bun000.dat');
   %y = load('bun045.dat');

   load demodata_ex5;
  x = x1; y = y2a;
  %     [siz1,tmp]=size(x);
%  [siz2,tmp]=size(y);

  demo_disp;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp (cmd_str, 'run_cg')
temp=x
theta = 0;
%maxm = sum(sum(x.*y))
arr=zeros(12,1);
%maxm=sqrt((y(1,:)-temp(1,:))^2 + (y(2,:)-temp(2,:))^2);
%fx=fftshift(fft2(fftshift(x)));
%fy=fftshift(fft2(fftshift(y)));

%arr(1,1)=maxm;
[siz1,tmp]=size(x);
```

```
[siz1,tmp]=size(x)
[siz2,tmp]=size(y)

%  xtemp=x;

%  valx1=sum(xtemp(:,1))
%  valy1=sum(xtemp(:,2))
%  valz1=sum(x(:,3))

% refx1=sum(valx1)/siz1
% refy1=sum(valy1)/siz1
% refz1=sum(valz1)/siz1

%  valx2=sum(y(:,1));
%  valy2=sum(y(:,2));
%  valz2=sum(y(:,3));

% refx2=sum(valx2)/siz2
% refy2=sum(valy2)/siz2
% refz2=sum(valz2)/siz2
% xshift=refx1-refx2;
% yshift=refy1-refy2;
% for i=1:siz1
%   xtemp(i,1)=x(i,1)-xshift;
%   xtemp(i,2)=x(i,2)-yshift;
  % x(i,3)=x(i,3)-zshift;
%end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   valx1=sum(x(:,1))
   valy1=sum(x(:,2))
%  valz1=sum(x(:,3))

 refx1=sum(valx1)/siz1
 refy1=sum(valy1)/siz1
% refz1=sum(valz1)/siz1

   valx2=sum(y(:,1));
   valy2=sum(y(:,2));
%  valz2=sum(y(:,3));

 refx2=sum(valx2)/siz2
 refy2=sum(valy2)/siz2
 %refz2=sum(valz2)/siz2
  xshift=refx1-refx2;
 yshift=refy1-refy2;
 for i=1:siz1
   x(i,1)=x(i,1)-xshift;
   x(i,2)=x(i,2)-yshift;
  % x(i,3)=x(i,3)-zshift;
end


  demo_disp;


elseif strcmp (cmd_str, 'run_rpm')

   [c,d,m]=cMIX2(x,y,frac,T_init, T_finalfac);
```

```
  disp ('RPM point matching done ...');

   elseif strcmp (cmd_str, 'run_icp')
   [c,d,m]=cMIX2 (x,y,frac,T_init, T_finalfac,1,'icp3');
   fid = fopen('match.txt','w');

     fprintf(fid,'%6.8f \n',m);

  % fprintf(fid,'%6.8f \n',v, '%3.2f\n',i,'%3.2f\n',j);
   fclose(fid);
  disp ('ICP point matching done ...');

end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------
% ------------------------------------------------------------------------
% demo_disp.m    (cMIX)
% ------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dispaly two point sets:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [] = demo_disp;
global x y
global axis_save

figure(1); clf;
cplot(x,y); hold on;



jnk  = axis; fac = 10;
xmin = jnk(1); xmax = jnk(2); ymin = jnk(3); ymax = jnk(4);
xmin = xmin - (xmax-xmin)/fac;
xmax = xmax + (xmax-xmin)/fac;
ymin = ymin - (ymax-ymin)/fac;
ymax = ymax + (ymax-ymin)/fac; axis_save = [xmin xmax ymin ymax];



[jnk,itmp] = max(x(:,2));
pttmp      = x(itmp,:);
htmp       = text (pttmp(1,1), pttmp(1,2)+(ymax-ymin)/fac/2,
'Template Point Set');
set        (htmp, 'color', 'g', 'fontsize', 12);
[jnk,itmp] = max(y(:,2));
pttmp      = y(itmp,:);
htmp       = text (pttmp(1,1), pttmp(1,2)+(ymax-ymin)/fac/2, 'Data
Point Set');
set        (htmp, 'color', 'r', 'fontsize', 12);

axis(axis_save); axis('off');
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------------
% ------------------------------------------------------------------
% cMIX1.m   (cMIX)
% ------------------------------------------------------------------
---
%%
% Usage:
%
% [c,d,m] = cMIX (x, y, frac, Tinit, Tfinalfac);
%
% Optional ones:
%
% [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac);
% [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac, disp_flag);
% [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac, disp_flag, m_method);
%
% [w]   = cMIX (x, y, z, sigma, frac, Tinit, Tfinalfac);
% [w]   = cMIX (x, y, z, sigma, frac, Tinit, Tfinalfac, disp_flag);
% [w]   = cMIX (x, y, z, sigma, frac, Tinit, Tfinalfac, disp_flag,
m_method,
% icp_sigma);
%
% Notes: the program will set transformation type automatically.
%        ['tps', 'rbf'].
%
%        'icp' -- 'icp0', 'icp3', 'icp5'. --> to set k_sigma.
%

function [o1,o2,o3] =  cMIX (in1,in2,in3,in4,in5,in6,in7,in8,in9);

figure(1); clf; whitebg('k'); set(gcf,'color',[0 0 0]);
% set(gcf,'DoubleBuffer','on')

% Init control parameters:
% ------------------------
perT_maxit  = 7;
relax_maxit = 1;
anneal_rate = 0.93;

lamda1_init = 1;
lamda2_init = 0.01;

% default:
% --------
disp_flag = 1;
m_method  = 'mixture';

debug_flag = 0;


% check input:
% ------------

% --- [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac) -------------------
--------
if (nargin == 5)
  x          = in1;
  y          = in2;
  frac       = in3;
  T_init     = in4;
```

```
   T_finalfac = in5;

   trans_type = 'tps';
   disp ('TPS');

   z          = x;
   sigma      = 1;

% --- [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac, disp_flag) --------
------
elseif (nargin == 6) & (length(in3) == 1)
   x          = in1;
   y          = in2;
   frac       = in3;
   T_init     = in4;
   T_finalfac = in5;
   disp_flag  = in6;

   trans_type = 'tps';
   disp ('TPS');

   z          = x;
   sigma      = 1;

% --- [c,d] = cMIX (x, y, frac, Tinit, Tfinalfac, disp_flag,
m_method) ------
elseif (nargin == 7) & (length(in3) == 1)
   x          = in1;
   y          = in2;
   frac       = in3;
   T_init     = in4;
   T_finalfac = in5;
   disp_flag  = in6;
   m_method   = in7;

   trans_type = 'tps';
   disp ('TPS');

   z          = x;
   sigma      = 1;


% --- [w]    = cMIX (x, y, z, sigma, frac, Tinit, Tfinalfac) ---------
--------
elseif (nargin == 7) & (length(in3) > 1)
   x          = in1;
   y          = in2;
   z          = in3;
   sigma      = in4;
   frac       = in5;
   T_init     = in6;
   T_finalfac = in7;

   trans_type = 'rbf';
   disp ('RBF');

   disp_flag = 1;


% --- [w]    = cMIX (x, y, z, sigma, frac, Tinit, Tfinalfac,
disp_flag) -----
```

```
elseif (nargin == 8) & (length(in3) > 1)
  x          = in1;
  y          = in2;
  z          = in3;
  sigma      = in4;
  frac       = in5;
  T_init     = in6;
  T_finalfac = in7;
  disp_flag  = in8;

  trans_type = 'rbf';
    disp ('RBF');


% --- [w]    = cMIX
(x,y,z,sigma,frac,Tinit,Tfinalfac,disp_flag,m_method); ---
elseif (nargin == 9) & (length(in3) > 1)
  x          = in1;
  y          = in2;
  z          = in3;
  sigma      = in4;
  frac       = in5;
  T_init     = in6;
  T_finalfac = in7;
  disp_flag  = in8;
  m_method   = in9;

  trans_type = 'rbf';
    disp ('RBF');


% --- [theta,tx,ty] = cMIX (x, y, frac, Tinit, Tfinalfac) -----------
-----------
elseif (nargin == 5) & (nargout == 3)
  x          = in1;
  y          = in2;
  frac       = in3;
  T_init     = in4;
  T_finalfac = in5;

trans_type = 'r+t';
  disp ('R+T');

  theta = 0; t = zeros (2,1); s = 1;

  disp_flag = 1;
  z         = x;
  sigma     = 1;

% --- [theta,tx,ty] = cMIX (x, y, frac, Tinit, Tfinalfac, disp_flag)
---------
elseif (nargin == 6) & (nargout == 3)
  x          = in1;
  y          = in2;
  frac       = in3;
  T_init     = in4;
  T_finalfac = in5;
  disp_flag  = in6;

trans_type = 'r+t';
  disp ('R+T');
```

```
  theta = 0; t = zeros (2,1); s = 1;

  z          = x;
  sigma      = 1;

% --- [theta,tx,ty] = cMIX (x, y, frac, Tinit, Tfinalfac,disp_flag,
m_method); ---
elseif (nargin == 7) & (nargout == 3)
  x          = in1;
  y          = in2;
  frac       = in3;
  T_init     = in4;
  T_finalfac = in5;
  disp_flag  = in6;
  m_method   = in7;

trans_type = 'r+t';
  disp ('R+T');

  theta = 0; t = zeros (2,1); s = 1;

  z          = x;
  sigma      = 1;

else
  disp ('# ERROR #: cMIX -- wrong input!');
  help cMIX; return;
end;

% take care of 'icp' k_sigma stuff.
if (strcmp(m_method(1:3), 'icp'))
  if length(m_method) == 3
    k_sigma  = 0;
    m_method = 'icp';
  else
    k_sigma  = str2num(m_method(4));
    m_method = 'icp';
  end;
else
  k_sigma = 0;
end;


% Init:
% -----

% init x,y,z:
[xmax, dim] = size(x); x = x (1:frac:xmax, :,:); [xmax, dim] =
size(x);
[ymax, tmp] = size(y); y = y (1:frac:ymax, :,:); [ymax, tmp] =
size(y);
[zmax, tmp] = size(z);

if strcmp(trans_type, 'tps')
  z = x;
end;
x

xmax
ymax
```

```
% init m:
m               = zeros(xmax, ymax);

T0              = max(x(:,1))^2;
moutlier        = 1/sqrt(T0)*exp(-1);       % /xmax *0.001;
% moutlier        = 1/xmax*0.01;
m_outliers_row = ones (1,ymax) * moutlier;
m_outliers_col = ones (xmax,1) * moutlier;

% init transformation parameters:
theta = 0; t = zeros (2,1); s = 1;

c_tps = zeros (xmax,dim+1);
d_tps = eye   (dim+1, dim+1);

w     = zeros (xmax+dim+1, dim);

% icp: perT_maxit = 1:
if strcmp (m_method(1:3),'icp')
  perT_maxit = 1;
end;

% ------------------------------------------------------------------
% Annealing procedure:
% ------------------------------------------------------------------

T       = T_init;
T_final = T_init / T_finalfac;
T_final =25 * T_final;
vx = x;
vy = y;

it_total = 1;
flag_stop = 0;
while (flag_stop ~= 1)

  for i=1:perT_maxit     % repeat at each termperature.

    % Given vx, y, Update m:
    if debug_flag; disp ('calc m ...'); end;
    m = cMIX_calc_m (vx, y, T, m_method, ...
   m_outliers_row, m_outliers_col,it_total,k_sigma);

    % Given m, update transformation:
    vy = m * y ./ ( (sum(m'))' * ones(1,dim));

    lamda1 = lamda1_init*length(x)*T;
    lamda2 = lamda2_init*length(x)*T;
    %lamda1 = lamda1_init;
    %lamda2 = lamda2_init;

    if debug_flag; disp ('calc c,d ...'); end;
    [c_tps, d_tps, w] = cMIX_calc_transformation (trans_type, ...
  lamda1, lamda2, sigma, x, vy, z);

    % w(1:length(z),:) = w(1:length(z),:)*0
    % d_tps = d_tps * 0 + eye(dim+1,dim+1);
    % c_tps = c_tps *0;

    if debug_flag; disp ('calc new x ...'); end;
    [vx] = cMIX_warp_pts (trans_type, x, z, c_tps, d_tps, w, sigma);
```

```
      end  % end of iteration/perT


    T = T * anneal_rate;

    % Determine if it's time to stop:
    % -------------------------------
    %  when T <= cluster variance, stop.

    if T < T_final; flag_stop = 1;

%   [d1,d2]=size(m);
     % match_matrix=zeros(d1,3);
      %for i=1:d1
       %  for j=1:d2
        %    if (m(i,j)~= 0)
         %      match_matrix(i,1)= i;
          %     match_matrix(i,2)= j;
           %    match_matrix(i,3)= m(i,j);
            %end;
        % end;
      %end;
     % match_matrix
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end;

    str = sprintf ('%s, T = %.4f:\t lamda1: %.4f lamda2: %.4f', ...
        m_method,T, lamda1, lamda2);
     disp (str);


    % Display:
    % --------
    it_total = it_total + 1;
    if (disp_flag == 1) | (flag_stop == 1) | ...
     (disp_flag ==2 & mod(it_total,10)==0) | ...
     (it_total == 1)
     figure(1);  clf;
     cMIX_plot_simple (2, x, y, z, vx, m, 1/xmax, T, ...
     trans_type, c_tps, d_tps, w, sigma, m_method);
      pause(0.5);
    end;

  end % end of annealing.



  % return outputs:
  % ---------------
  if strcmp (trans_type, 'tps')
    o1 = c_tps
    o2 = d_tps
    o3 = [];
  elseif strcmp (trans_type, 'rbf')
    o1 = w;
    o2 = [];
    o3 = [];
  elseif strcmp (trans_type, 'r+t')
    o1 = theta;
```

```
  o2 = tx;
  o3 = ty;
end
o3 = m;
%m(1:98,1:98)



%%%%%
% 1 % %%% cMIX_calc_m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Update m (correspondence).
%
% Usage:
% [m] = cMIX_calc_m (vx, y, T, 'icp');
% [m] = cMIX_calc_m (vx, y, T, 'mixture');
% [m] = cMIX_calc_m (vx, y, T, 'rpm');
%
% Notes: for "icp", set k_sigma = 0 -- no outlier.
%
% 01/31/00

function [m, m_outliers_row, m_outliers_col] = cMIX_calc_m ...
    (vx, y, T, m_method, m_outliers_row, m_outliers_col, it_total,
icp_sigma);

[xmax,dim] = size(vx);
[ymax,dim] = size(y);


% ---------------------------------------------------------------------
ICP ---
if strcmp (m_method, 'icp')

  k_sigma = icp_sigma;
  [m, dist_threshold] = cMIX_calc_m_ICP (vx, y, k_sigma);
  m = m + randn(xmax, ymax) * (1/xmax) * 0.001;
  [r,c]=size(y);
  diff=(y(:,1)-vx(:,1)).^2 + (y(:,2)-vx(:,2)).^2;

  error=sum(diff)/r;
  error=sqrt(error);
  %error=diff/abs(vx);
  %error=norm(diff,2);
  %rms=nerror/sqrt(nerror);
  fprintf('error = %f \n', error);



% ------------------------------------------------------ one way
mixture ---
elseif strcmp (m_method, 'mixture')

  % Given v=tranformed(x), update m:
  y_tmp = zeros (xmax, ymax);
  for it_dim=1:dim
    y_tmp = y_tmp + (vx(:,it_dim) * ones(1,ymax) - ones(xmax,1) *
y(:,it_dim)').^2;
```

```
  end;

  m_tmp = 1/sqrt(T) .* exp (-y_tmp/T);
  m_tmp = m_tmp + randn(xmax, ymax) * (1/xmax) * 0.001;

  m = m_tmp;

  % normalize accross the outliers as well:
  sy          = sum (m) + m_outliers_row;
  m           = m ./ (ones(xmax,1) * sy);

  % sx = sum(m')' + m_outliers_col;
  % m2 = m ./ (sx * ones(1,ymax));
  % m = (m+m2)/2;

  %%%%%My part for test only

  [r,c]=size(y);
  diff=(y(:,1)-vx(:,1)).^2 + (y(:,2)-vx(:,2)).^2;

  error=sum(diff)/r;
  error=sqrt(error);
  fprintf('error = %f \n', error);



% ------------------------------------------------------------ mixture -
RPM ---
elseif strcmp (m_method, 'mix-rpm')

  % Given v=tranformed(x), update m:
  y_tmp = zeros (xmax, ymax);
  for it_dim=1:dim
    y_tmp = y_tmp + (vx(:,it_dim) * ones(1,ymax) - ones(xmax,1) *
y(:,it_dim)').^2;
  end;

  m_tmp = 1/sqrt(T) .* exp (-y_tmp/T);
  m_tmp = m_tmp + randn(xmax, ymax) * (1/xmax) * 0.001;

  m = m_tmp;

  [m, junk1, junk2] = cMIX_normalize_m (m_tmp, m_outliers_col,
m_outliers_row);

  % normalize accross the outliers as well:
  %sy          = sum (m) + m_outliers_row;
  %m           = m ./ (ones(xmax,1) * sy);

  %sx = sum(m')' + m_outliers_col;
  %m2 = m ./ (sx * ones(1,ymax));
  %m = (m+m2)/2;


% ------------------------------------------------ RPM, double
normalization ---
elseif strcmp (m_method, 'rpm')
  % Given v=tranformed(x), update m:
  y_tmp = zeros (xmax, ymax);
  for it_dim=1:dim
```

```
    y_tmp = y_tmp + (vx(:,it_dim) * ones(1,ymax) - ones(xmax,1) *
y(:,it_dim)').^2;
  end;

  m_tmp = exp (-y_tmp/T);
  m_tmp = m_tmp + randn(xmax, ymax) * (1/xmax) * 0.001;

  % double normalization, but keep outlier entries constant.
  moutlier       = 1/xmax * 0.1;
  m_outliers_row = ones (1,ymax) * moutlier;
  m_outliers_col = ones (xmax,1) * moutlier;

  [m, junk1, junk2] = cMIX_normalize_m (m_tmp, m_outliers_col,
m_outliers_row);

  %%%%% RMS error

  [r,c]=size(y);
  diff=(y(:,1)-vx(:,1)).^2 + (y(:,2)-vx(:,2)).^2;
  error=sum(diff)/r;
  error=sqrt(error);
  fprintf('error = %f \n', error);

% ------------------------------------------- RPM, double
normalization ---
elseif strcmp (m_method, 'rpm-old')
  % Given v=tranformed(x), update m:
  y_tmp = zeros (xmax, ymax);
  for it_dim=1:dim
    y_tmp = y_tmp + (vx(:,it_dim) * ones(1,ymax) - ones(xmax,1) *
y(:,it_dim)').^2;
  end;

  m_tmp = exp (-y_tmp/T);
  m_tmp = m_tmp + randn(xmax, ymax) * (1/xmax) * 0.001;

  % double normalization, also update outlier entries.
  if (it_total == 1)
    moutlier       = 1/xmax * 0.1;
    m_outliers_row = ones (1,ymax) * moutlier;
    m_outliers_col = ones (xmax,1) * moutlier;
  end;
  [m, m_outliers_row, m_outliers_col] = cMIX_normalize_m (m_tmp,
m_outliers_col, m_outliers_row);

else
  disp ('# ERROR #: cMIX_calc_m -- wrong input!');
end
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------------------------------
% cplot.m
% ------------------------------------------------------------------------
% Plot points.
%
% Usage:
% [] = cplot (x);
% [] = cplot (x,y);
% [] = cplot (x,y,z);
%
% [] = cplot (x, marker_str, marker_size);
% [] = cplot (x, xmarker_str, xmarker_size, y, ymarker_str,
y_marker_size);
%

function [] = cplot (in1,in2,in3,in4,in5,in6);

% check input:
if (nargin == 1) % ------------------------------- (x) ---
  x = in1; xmarker_str = 'go'; xmarker_size = 6;
  y = [];  ymarker_str = 'r+'; ymarker_size = 6;
  z = [];  zmarker_str = 'bo'; zmarker_size = 6;

elseif (nargin == 2) % ------------------------- (x,y) ---
  x = in1; xmarker_str = 'go'; xmarker_size = 6;
  y = in2; ymarker_str = 'r+'; ymarker_size = 6;
  z = [];  zmarker_str = 'bo'; zmarker_size = 6;

elseif (nargin == 3) & (~isstr(in2)) % -------- (x,y,z) ---
  x = in1; xmarker_str = 'go'; xmarker_size = 6;
  y = in2; ymarker_str = 'r+'; ymarker_size = 6;
  z = in3; zmarker_str = 'bo'; zmarker_size = 6;

elseif (nargin == 3) & (isstr(in2)) % ---- (x, 'go', 3) ---
  x = in1; xmarker_str = in2; xmarker_size = in3;
  y = [];
  z = [];

elseif (nargin == 6) % -------- (x, 'go, 3, y, 'r+', 3) ---
  x = in1; xmarker_str = in2; xmarker_size = in3;
  y = in4; ymarker_str = in5; ymarker_size = in6;
  z = [];

else
  disp ('# ERROR #: cplot -- wrong input!');
  help cplot; return;
end;

% plot x:
[n, dim] = size(x);
if (n >= 1); cplot_1pointset (x, xmarker_str, xmarker_size); end;
hold on;

% plot y:
[n, dim] = size(y);
if (n >= 1); cplot_1pointset (y, ymarker_str, ymarker_size); end;

% plot z:
```

```
[n, dim] = size(z);
if (n >= 1); cplot_1pointset (z, zmarker_str, zmarker_size); end;
hold off;
```

```
%%%%%
% 1 % %%% cplot_1pointset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Plot one point set.
%
% Usage:
% [] = cplot_1pointset (x, xmarker_str, xmarker_size);
%
% 02/01/00

function [] = cplot_1pointset (x, xmarker_str, xmarker_size);

[n,dim] = size(x);
if (dim == 2)
  h = plot (x(:,1), x(:,2), xmarker_str, 'markersize', xmarker_size);
axis('equal');
  hold on;
elseif (dim == 3)
  h = plot3 (x(:,1), x(:,2), x(:,3), xmarker_str, 'markersize',
xmarker_size); axis('equal');
  axis('equal'); set (gca, 'box', 'on');
  hold on;
end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------------------
% cplot_2g.m
% ------------------------------------------------------------------
% Plot links between 2 corresponding point sets.
%
% Usage:
% [] = cplotg (x, y);                    % m = eye.
% [] = cplotg (x, y, m);                 % thr = 0.
% [] = cplotg (x, y, m, threshold);
% [] = cplotg (x, y, m, threshold, color_str);
% [] = cplotg (x, y, m, threshold, color_str, marker_str);
%

function [] = cplotg(x, y, m, threshold, color_str, marker_str);

% check input:
% ------------
[n, dim] = size (x);

if (nargin == 2)
  m         = eye (n,n);
  threshold = 0;
  gcolor_str = 'y:';

elseif (nargin == 3)
  threshold = 0;
  gcolor_str = 'y:';

elseif (nargin == 4)
  gcolor_str = 'y:';

elseif (nargin == 5)
  gcolor_str = color_str;

elseif (nargin == 6)
  ;
else
  disp ('# ERROR #: cplotg -- wrong input!');
  help cplotg; return;
end;

%keyboard

switch (dim)

  case 2 % ------------------------------------- 2D ---

  % normal plot:
  if (nargin < 6)
    % Reformat data:
    % --------------
    xy = [x; y];

    [siz1, temp] = size(x);
    [siz2, temp] = size(y);

    [indexi, indexj] = find ( m > threshold );
```

```
      index = indexi + (indexj-1) * siz1;

      msp = zeros(siz1,siz2);
      msp (index) = 1;

      madj = [zeros(siz1), msp;
    msp',zeros(siz2)];


      % Plot:
      % -----
      axis ('equal'); axis('off');
      gplot (madj, xy, gcolor_str);

    % otherwise, i want to change the color:
    else
      [siz1, temp] = size(x);
      [siz2, temp] = size(y);

      [indexi, indexj] = find ( m > threshold );
      n = length(indexi);

      % Plot:
      % -----
      for i=1:n
    hold on;
    tmp = [x(indexi(i),:); y(indexj(i),:)];
    plot (tmp(:,1), tmp(:,2), ...
        'color', color_str, 'linestyle', marker_str);
    axis ('equal');
      end;
    end;

  case 3 % ---------------------------------------- 3D ---
    [siz1, temp] = size(x);
    [siz2, temp] = size(y);

    [indexi, indexj] = find ( m > threshold );
    n = length(indexi);

    % Plot:
    % -----
    for i=1:n
      hold on;
      tmp = [x(indexi(i),:); y(indexj(i),:)];
      plot3 (tmp(:,1), tmp(:,2), tmp(:,3), ...
     'color', color_str, 'linestyle', marker_str);
    end;
  otherwise;
end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------
% ctps_gen.m
% ------------------------------------------------------------------------
% Purpose: Generate all parameters for TPS.
%
% Usage:
% [K]              = ctps_gen (x);
% [K]              = ctps_gen (x, y);
% [c,d]            = ctps_gen (x, y, lamda1);
% [c,d]            = ctps_gen (x, y, lamda1, lamda2);
% [q1,q2,R,K]      = ctps_gen (x);
% [q1,q2,R,K,c,d] = ctps_gen (x, y, lamda1);
%

function [o1,o2,o3,o4,o5,o6] = ctps_gen (x,y,lamda1,lamda2);

% check input.
if nargin <= 1 | nargin >= 5
  disp ('# ERROR #: ctps_gen -- wrong input !');
  help ctps_gen; return;
end;

o1 = [];
o2 = [];
o3 = [];
o4 = [];
o5 = [];
o6 = [];

% --- [K] = ctps_gen (x) ------------------------------------------
if nargin == 1 & nargout == 1
  [n, dim] = size (x); x = [ones(n,1), x];
  [K] = ctps_gen_K (x,x);

  o1 = K;

% --- [K] = ctps_gen (x,y) ----------------------------------------
elseif nargin == 2 & nargout == 1
  [n, dim] = size (x); x = [ones(n,1), x];
  [m, dim] = size (y); y = [ones(m,1), y];
  [K] = ctps_gen_K (x,y);

  o1 = K;

% --- [c,d] = ctps_gen (lamda, t, y) ------------------------------
elseif nargin == 3 & nargout == 2
  [n, dim] = size (x); x = [ones(n,1), x];
  [m, dim] = size (y); y = [ones(n,1), y];

  [K]       = ctps_gen_K (x,x);
  [q1,q2,R] = ctps_gen_qr      (x);

  [c,d]     = ctps_gen_cd      (lamda1,q1,q2,R,K,y);

  o1 = c;
  o2 = d;
```

```
% --- [c,d] = ctps_gen (x, y, lamda1, lamda2) ----------------------
elseif nargin == 4 & nargout == 2
  [n, dim] = size (x); x = [ones(n,1), x];
  [m, dim] = size (y); y = [ones(n,1), y];

  [K]       = ctps_gen_K (x, x);
  [q1,q2,R] = ctps_gen_qr     (x);

  [c,d]     = ctps_gen_cd_regularized (lamda1,lamda2,q1,q2,R,K,y);

  o1 = c;
  o2 = d;

% --- [q1,q2,R,K] = ctps_gen (x) -----------------------------------
elseif nargin == 1 & nargout == 4
  [n, dim] = size (x); x = [ones(n,1), x];

  [K]       = ctps_gen_K (x, x);
  [q1,q2,R] = ctps_gen_qr     (x);

  o1 = q1;
  o2 = q2;
  o3 = R;
  o4 = K;

% --- [q1,q2,R,K,c,d] = ctps_gen (x, y, lamda1) --------------------
elseif nargin == 3 & nargout == 6
  [n, dim] = size (x); x = [ones(n,1), x];
  [m, dim] = size (y); y = [ones(n,1), y];

  [K]       = ctps_gen_K (x, x);
  [q1,q2,R] = ctps_gen_qr     (x);

  [c,d]     = ctps_gen_cd (lamda1,q1,q2,R,K,y);

  o1 = q1;
  o2 = q2;
  o3 = R;
  o4 = K;
  o5 = c;
  o6 = d;
else
  disp ('# ERROR #: ctps_gen -- wrong input!');
  help ctps_gen;
end;



%%%%%
% 1 % %%%  ctps_gen_qr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Purpose: Genrate QR decomposition for pts (set x).
% Usage:   [q1,q2,R] = ctps_gen_qr (x);
% Notes:   x = [q1:q2] * r;
%          nxn [nxM,nxn-M] * nxM
% 01/21/00

function [q1,q2,R] = ctps_gen_qr (x);
```

```
[n,M] = size (x);

[q,r]    = qr(x);
q1       = q(:, 1:M);
q2       = q(:, M+1:n);
R        = r(1:M,1:M);



%%%%%
% 2 % %%% ctps_gen_K
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Purpose: Generate K (TPS kernel) matrix.
% Usage:   [K] = ctps_gen_K (x,z);
% Notes:   (x,z should be expanded before feed into here!).
%
% 01/21/00

function [K] = ctps_gen_K (x,z);

% Format:
[n, M] = size (x);
[m, M] = size (z);
dim    = M  - 1;

% calc. the K matrix.
% 2D: K = r^2 * log r
% 3D: K = -r
K= zeros (n,m);

for it_dim=1:dim
  tmp = x(:,it_dim+1) * ones(1,m) - ones(n,1) * z(:,it_dim+1)';
  tmp = tmp .* tmp;
  K = K + tmp;
end;

if dim == 2
  mask = K < 1e-10; % to avoid singularity.
  K = 0.5 * K .* log(K + mask) .* (K>1e-10);
else
  K = - sqrt(K);
end;



%%%%%
% 3 % %%% ctps_gen_cd
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Purpose: Calc. normal TPS c,d.
% Usage:   [c,d] = ctps_gen_cd (lamda1,q1,q2,R,K,y)
%
% 01/21/00

function [c,d] = ctps_gen_cd (lamda1,q1,q2,R,K,y);
```

```
[n,M] = size(y);

gamma = inv (q2'*K*q2 + lamda1*eye(n-M, n-M)) * q2' * y;
c     = q2 * gamma;
d     = inv(R) * q1' * (y-K*q2*gamma);
```

```
%%%%%
% 4 % %%% ctps_gen_cd_regularized
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Purpose: Calc. regularized TPS c,d.
%          (Regularize the affine transformation as well).
%
% Usage: [c,d] = ctps_gen_cd_regularized (lamda1,lamda2,q1,q2,R,K,y)
%
% 01/21/00

function [c, d] = ctps_gen_cd_regularized
(lamda1,lamda2,q1,q2,R,K,y);

[n,M] = size(y);
dim = M - 1;

gamma =  inv (q2'*K*q2 + lamda1*eye(n-M)) * q2' * y;
c     = q2*gamma;

% add regularization for "d" as well:
% d = inv(R) * q1' * (y-K*q2*gamma);
A = inv(R'*R + lamda2 * eye(length(R),length(R))) * ( R'*q1'*(y-
K*q2*gamma) - R'*R);
d = A + eye(dim+1,dim+1);
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------------------
% crbf_gen.m
% ---------------------------------------------------------------------
% Generate Radial-Basis-Function spline parameters.
%
% Usage:
% Usage:
% [phi]   = crbf_gen (x, z, sigma_kernel);
% [phi,w] = crbf_gen (x, y, z, lamda1, sigma_kernel);
% [phi,w] = crbf_gen (x, y, z, lamda1, lamda2, sigma_kernel);
%
% x -- pts to be warpped.
% y -- target pts.
% z -- basis pts.

function [o1,o2] = crbf_gen (x,y,z, lamda1, lamda2, sigma_kernel);


% check input:
% ------------

% --- [phi]   = gen_crbf (x, z, sigma_kernel) ----------------------
-----
if (nargin == 3);
  sigma_kernel = z;
  z            = y;

  [L,dim] = size(x);
  P       = length(z);

  [phi] = crbf_kernel (x, z, sigma_kernel);

  o1 = phi;

% --- [phi,w] = crbf_gen (x, y, z, lamda1, sigma_kernel) ------------
-----
elseif (nargin == 5);
  sigma_kernel = lamda2;
  lamda2       = 0;

  [L,dim] = size(x);
  P       = length(z);

  [phi] = crbf_kernel (x, z, sigma_kernel);

  G       = eye(L,L);                    % assume no outlier in "x".

  Iw                                   = zeros (P+dim+1, P+dim+1);
  Iw (1:P,1:P)                         = lamda1 * eye(P,P);
  Iaffine                              = zeros (P+dim+1, P+dim+1);
  Iaffine (P+1:P+dim+1, P+1:P+dim+1) = lamda2 * eye (dim+1,dim+1);

  w = inv(phi' * G * phi + (Iw + Iaffine)) * phi' * y;

  o1 = phi;
  o2 = w;
```

```
% --- [phi,w] = crbf_gen (x, y, z, lamda1, lamda2, sigma_kernel) ----
-----
elseif (nargin == 6);
  [L,dim] = size(x);
  P       = length(z);

  [phi] = crbf_kernel (x, z, sigma_kernel);

  G       = eye(L,L);                    % assume no outlier in "x".

  Iw                                   = zeros (P+dim+1, P+dim+1);
  Iw (1:P,1:P)                         = lamda1 * eye(P,P);
  Iaffine                              = zeros (P+dim+1, P+dim+1);
  Iaffine (P+1:P+dim+1, P+1:P+dim+1) = lamda2 * eye (dim+1,dim+1);

  w = inv(phi' * G * phi + (Iw + Iaffine)) * phi' * y;

  o1 = phi;
  o2 = w;


else
  disp ('# ERROR #: crbf_gen -- wrong input!');
  help crbf_gen;
end;
```

```
%%%%%
% 1 % %%% crbf_kernel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Calc. BRF kernel "phi".
%
% Usage:
% [phi] = cgtm_calc_kernel (x, z, sigma_kernel)
%
% 01/26/00

function [phi] = crbf_kernel (x, z, sigma_kernel)

[L,dim] = size(x);
P       = length(z);

phi = zeros(L,P);

for it_dim=1:dim
  tmp = x(:,it_dim) * ones(1,P) - ones(L,1) * z(:,it_dim)';
  tmp = tmp .* tmp;
  phi = phi + tmp;
end;
```

```
if dim == 2
  phi = exp(-phi/(sigma_kernel^2)); % gaussian basis fn.

else
  phi = - sqrt(phi);
end;

phi = [phi, x, ones(L,1)];    % x, ones: affine part.
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------------------
% ctps_plot_grid.m
% ----------------------------------------------------------
% Display TPS deformed grid.
%
% Usage:
% [] = ctps_plot_grid (x,y,c,d);
% [] = ctps_plot_grid (x,y,lamda);
% [] = ctps_plot_grid (x,y,c,d,  resolution,resolution_grid);
% [] = ctps_plot_grid (x,y,lamda,resolution,resolution_grid);
%
% for (x,y,c,d)
% x -- TPS basis points.
% y -- points to be warped.
%
% for (x,y,lamda)
% x -- TPS basis points and points to be warped.
% y -- target points

function [] = ctps_plot_grid_simple (x,y,c,d,...
    resolution,resolution_grid);

% t = [7,6; 10,9; 1 1; 1 15; 15 1; 15 15];
% y = [11,7; 7,10; 1 1; 1 15; 15 1; 15 15];
% [c,d] = ctps_gen (t, y, 1);

% check input.
% ------------
if (nargin == 3);     % input (x,y,lamda);
  lamda = c;
  [c,d] = ctps_gen (x,y,lamda);
  tmp = x;  y = x; x = tmp;

  resolution      = 4;
  resolution_grid = 3;

elseif (nargin == 4); % input (x,y,c,d);
  resolution      = 4;
  resolution_grid = 3;

elseif (nargin == 5); % input (x,y,lamda,resolution,resolution_grid);
  lamda           = c;
  resolution      = d;
  resolution_grid = resolution;
  [c,d]           = ctps_gen (x,y,lamda);
  tmp = x;  y = x; x = tmp;

elseif (nargin == 6);  % input (x,y,c,d,resolution,resolution_grid);
  % do nothing.

else
  disp ('# ERROR #: ctps_plot_grid -- wrong input!');
  help ctps_plot_grid; return;
end;



% generate grid points.
% ---------------------
```

```
[grid_pts, controls] = ctps_plot_grid_gen (x, resolution,
resolution_grid);


% Warp the grid:
% --------------
[grid_new] = ctps_warp_pts (grid_pts, x, c, d);


% Plot:
% -----
ori_color = ones(1,3) * 0.7;
new_color = 'b';

ctps_plot_gridbox (1, grid_pts, controls, ori_color, ':'); hold on;
ctps_plot_gridbox (1, grid_new, controls, 'b','-'); hold on;
%axis('equal'); axis ('off');
```

```
% Non-Rigid Point Matching (ICP) Demo:
% --------------------------------------------------------------------
% ctps_plot_grid_gen.m
% --------------------------------------------------------------------
% Generate grid points for displaying TPS deformation.
%
% Usage:
% [grid_pts, controls] = ctps_plot_grid_gen (x);
% [grid_pts, controls] = ctps_plot_grid_gen (x, resolution,
resolution_grid);
%
%         *** "controls" are for: ctps_plot_gridbox.
%

function [grid_pts, controls] = ctps_plot_grid_gen (x, resolution,
resolution_grid);

% check input:
% ------------
if (nargin == 1);       % input (x), set the other 2.
  resolution = 4;
  resolution_grid = 3;
elseif (nargin == 3);  % input (x, resolution, resolution_grid);
  ;
else
  disp ('# ERROR #: ctps_plot_grid_gen -- wrong input!');
  help ctps_plot_grid_gen;
end;



% set grid range:
% ---------------
xrange = [min(x(:,1)), max(x(:,1))];
yrange = [min(x(:,2)), max(x(:,2))];

% expand a little bit:
expand_ratio = 5;
xrange (1) = xrange (1) - (xrange(2)-xrange(1))/expand_ratio;
xrange (2) = xrange (2) + (xrange(2)-xrange(1))/expand_ratio;
yrange (1) = yrange (1) - (yrange(2)-yrange(1))/expand_ratio;
yrange (2) = yrange (2) + (yrange(2)-yrange(1))/expand_ratio;


% Generate the grid points:
% -------------------------
[grid_pts, rows, cols, points_row, points_col] = cgrid_generate ...
    (xrange(1), xrange(2), yrange(1), yrange(2), resolution,
resolution_grid);


controls    = zeros (4,1);
controls(1) = rows;
controls(2) = cols;
controls(3) = points_row;
controls(4) = points_col;
```

```
%%%%%
% 1 % %%% cgrid_generate.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Generate grid.
%
% Input: x/y       -- one of data sets.
%
% Output: grid_pts  -- grid points, same format as x/y.
%         rows,cols -- grid dimensionality.
%         points_row, points_cols -- points along each
%                                    row and col.
% ------------------------------------------------------------
% Last modified: 09/27/99

function [grid_pts, rows, cols, points_row, points_col] = ...
    cgrid_generate (xrange1, xrange2, yrange1, yrange2, ...
                    resolution, resolution_grid);

xrange = [xrange1, xrange2];
yrange = [yrange1, yrange2];

% a: grid square size.
a = min(xrange(2)-xrange(1), yrange(2)-yrange(1)) / resolution;
grid_step = a / resolution_grid;

rows = ceil((yrange(2)-yrange(1)) / a + 1);
cols = ceil((xrange(2)-xrange(1)) / a + 1);

yrange(2) = yrange(1) + (rows-1)*resolution_grid*grid_step;
xrange(2) = xrange(1) + (cols-1)*resolution_grid*grid_step;

%keyboard

grid_pts = [];
% points_row = floor( (xrange(2)-xrange(1))/grid_step );
points_row = (cols-1) * resolution_grid + 1; % two ending points.
for i=1:rows
  tmp_row = [[xrange(1):grid_step:xrange(2)]', ...
   ones(points_row,1) * (i-1) * a + yrange(1)];
  grid_pts = [grid_pts; tmp_row];
end;


% points_col = floor((yrange(2)-yrange(1))/grid_step );
points_col = (rows-1) * resolution_grid + 1; % two ending points.
for j=1:cols
  tmp_col = [ones(points_col,1) * (j-1) * a + xrange(1), ...
   [yrange(1):grid_step:yrange(2)]'];
  grid_pts = [grid_pts; tmp_col];
end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------------------------
```

```
% ctps_plot_gridbox.m
% ------------------------------------------------------------------
% Plot grid box for TPS demonstration.
%
% Usage:
% [] = cplot_gridbox (grid_method, grid, controls);
%
% grid_method: 0 -- pts
%              1 -- pts + linking lines.
% controls: (rows, cols, points_row, points_col).
%

function [] = ctps_plot_gridbox (grid_method, grid, controls, ...
    marker_color, marker_type);

rows       = controls(1);
cols       = controls(2);
points_row = controls(3);
points_col = controls(4);



switch (grid_method)

  case 0
    plot (grid(:,1), grid(:,2), '.', 'color',
marker_color,'markersize', 5);

  case 1
    for i=1:rows
      tmp = grid ( (i-1)*points_row+1:i*points_row,:);
      plot (tmp(:,1), tmp(:,2), 'color', marker_color, ...
     'linestyle', marker_type);%, ...
     %'erasemode', 'background');
      hold on;
    end;

    start_index = rows * points_row;
    for j=1:cols
      tmp = grid ( (j-1)*points_col + start_index + 1 : j*points_col
+ ...
     start_index, :);
      plot (tmp(:,1), tmp(:,2), 'color', marker_color, ...
     'linestyle', marker_type);%, ...
%    'erasemode', 'background');
      hold on;
    end;
  otherwise;
    disp ('ERROR: cplot_gridbox -- wrong input parameters');
end;
```

```
% Non-Rigid Point Matching (ICP) Demo :
% ------------------------------------------------------------------
% ctps_warp_pts.m
% ------------------------------------------------------------------
% Purpose: Given TPS [t,c,d], warp pts x --> x1.
%          (z is the basis point set).
%          (x1 is in normal coordinate, not expanded !).
%
% Usage:
```

```
% [pts1] = ctps_warp_pts (pts, x, c, d);
% [pts1] = ctps_warp_pts (pts, x, y, lamda);
% [pts1] = ctps_warp_pts (pts, x, y);        lamda default = 1;
%

function [pts1]= ctps_warp_pts (pts, x, c, d);

% check input.
if nargin <= 1 | nargin >= 5
  disp ('# ERROR #: ctps_warp_pts -- wrong input !');
  help ctps_warp_pts; return;
end;


[dim1, dim2] = size(d); % d      -- affine: dim > 1
                        % lamda -- par:    dim = 1.

% --- [pts1] = ctps_warp_pts (pts, x, c, d) ------------------------
-
if dim1 > 1

   K = ctps_gen (pts,x);

  % Warp pts --> pts1:
  [n,dim] = size(pts); pts = [ones(n,1), pts];
  pts1 = pts*d + K*c;
  pts1 = pts1 (:,2:dim+1);
end;


% [pts1] = ctps_warp_pts (pts, x, y, lamda) -------------------------
--
if dim1 == 1

  x = x;
  y = c;     % y     is taking c's position as input now.

  if (nargin == 3)
    lamda = 1;
  else
    lamda = d; % lamda is taking d's position as input now.
  end;

  K     = ctps_gen (pts,x);
  [c,d] = ctps_gen (x,y,lamda);

  % Warp pts --> pts1:
  [n,dim] = size(pts); pts = [ones(n,1), pts];
  pts1 = pts*d + K*c;
  pts1 = pts1 (:,2:dim+1);
end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------------------
% 3 % %%% cMIX_warp_pts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
function [vx] = cMIX_warp_pts (trans_type, x, z, c_tps, d_tps, w,
sigma_kernel);
```

```
switch (trans_type)
  case 'tps'
    vx = ctps_warp_pts (x, z, c_tps, d_tps);
  case 'rbf'
    vx = crbf_warp_pts (x, z, w, sigma_kernel);
  % case 'gtm_tps'
  %  vx = cgtm_warp_pts ('tps_style', x, z, w, 0);
  otherwise;
    disp ('# ERROR #: cMIX_warp_pts -- wrong input!');
end
```

```
% Non-Rigid Point Matching (ICP) Demo:
% --------------------------------------------------
% 4 % %%% cMIX_plot_simple
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Plot the cMIX progress. (no Cx, just T -- as isotropic covariance).
%
% Usage:
% [] = cMIX_plot_simple (method, x, y, z, vx, m, m_threshold, T, ...
%    transformation_type, c_tps, d_tps, w, sigma_kernel);
%

function [] = cMIX_plot_simple (method, x, y, z, vx, m, m_threshold,
T, ...
    transformation_type, c_tps, d_tps, w, sigma_kernel, m_method);


% configure:
xmarker = 'go'; xsize = 6;
ymarker = 'r+'; ysize = 3;
zmarker = 'go'; zsize = 6;
inter_marker = 'g+';


% -----------------------------------------------------------------
% Generate 3 windows.
% -----------------------------------------------------------------
[siz1,dim] = size (x);
[siz2,dim] = size (y);
c = c_tps;
d = d_tps;


% --- 2D data -----------------------------------------------------
-
if (dim == 2)

switch (method)
  case 2

   % h_sub1 = subplot ('position', [0.05 0.6 0.2 0.3]);
   % cplot (x, xmarker, xsize); hold on;
   % cplot (y, ymarker, ysize);
   % axis ('equal'); axis ('off'); title ('Original V and X');


%    h_sub2 = subplot ('position', [0.05 0.1 0.2 0.3]);
%    cplot (vx, xmarker, xsize); hold on;
%    cplot (y,  ymarker, ysize);
%    axis ('equal'); axis ('off'); title ('Transformed V + X');

    h_sub3 = subplot ('position', [0.3 0.1 0.4 0.7]);
    cplotg (vx, y, m, m_threshold); hold on;
    if ~(strcmp(m_method, 'icp'))
      cMIX_plot_mixture_simple (vx,T,'b-'); hold on;
    end;
    cplot (vx, xmarker, xsize); hold on;
    cplot (y,  ymarker, ysize); hold on;
    axis ('equal'); axis ('off'); title ('Transformed V + X');
```

```
  %  h_sub4 = subplot ('position', [0.75 0.6 0.2 0.3]);
  %  switch (transformation_type)
  %    case 'tps'
%  ctps_plot_grid (x, x, c, d);
%  axis('equal'); axis('off');title ('TPS Warping');
%      case 'rbf'
%  crbf_plot_grid (x,z,w, sigma_kernel);
%  axis('equal'); axis('off');title ('RBF Warping');
   % case 'gtm_tps'
   % cgtm_plot_grid_simple ('tps_style', x,y,z,w, 0);
   %  axis('equal'); axis('off');title ('GTM Warping');
  %  end;
%   vx2 = [ones(siz1,1), x] * d; vx2 = vx2(:,2:dim+1);
 %  cplot (y,   ymarker,      ysize); hold on;
 %  cplot (vx, xmarker, xsize); hold on;
 %    cplot (vx2, inter_marker, xsize);
 %   axis('equal'); axis('off'); %title ('TPS Warping');

  %  h_sub5 = subplot ('position', [0.75 0.1 0.2 0.3]);
  %  vy  = m * y ./  ( (sum(m'))' * ones(1,dim));
  %  cplot (y,  'r.',          ysize); hold on;
  %  cplot (vy, ymarker,       xsize);
    % cplot (x,  xmarker,       xsize);
    % cplot (vx, inter_marker, xsize);
    % cplotg(x, vx);
 %   axis('equal'); axis('off'); title ('Estimated Shape Y=MX');

    % [siz1,temp] = size (x);
    % cplot_2g (x, m*y, 0,0,0,1, eye(siz1), m_threshold);
    % title ('m*y w/ links');

  case 0

    cplotg (vx, y, m, m_threshold); hold on;
    cplot  (vx, xmarker, xsize); hold on;
    cplot  (y,  ymarker, ysize); hold on;
    cMIX_plot_mixture_simple (vx, T); title ('Transformed V + X');

  case 1

    h_sub1 = subplot ('position', [0.05 0.6 0.25 0.3]);
    cplot (x, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize);
    axis ('equal'); axis ('off'); title ('Original V and X');

    h_sub2 = subplot ('position', [0.05 0.1 0.25 0.3]);
    cplot (vx, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize);
    axis ('equal'); axis ('off'); title ('Transformed V + X');

    h_sub3 = subplot ('position', [0.35 0.1 0.55 0.7]);
    cplotg (vx, y, m, m_threshold); hold on;
    cMIX_plot_mixture_simple (vx, T);
    axis ('equal'); axis ('off'); title ('Transformed V + X');


  otherwise;
end;
```

```
% --- 3D data ------------------------------------------------------
-
else

switch (method)

  case 1

    set (gcf, 'color', [0 0 0]);
    hold off;

    h_sub1 = subplot ('position', [0.05 0.6 0.25 0.3]); axis
('equal'); axis ('off');
    cplot (x, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize); title ('Original V_x and Y');
    set(gca, 'box', 'on');

    h_sub2 = subplot ('position', [0.05 0.1 0.25 0.3]); axis ('off');
    cplot (vx, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize); title ('Transformed V_x + Y');
    set(gca, 'box', 'on');

    h_sub3 = subplot ('position', [0.35 0.1 0.55 0.7]);
    % cMIX_plot_mixture_simple (vx, T); title ('Transformed V_x +
Y');
    % keyboard
    cplotg ('black', vx, y, m, m_threshold); hold on;
    cplot (vx, xmarker, xsize); hold on;
    cplot (y, ymarker, xsize); title ('Transformed V_x + Y'); hold
on;
    axis('on'); set(gca, 'box', 'on'); rotate3d on;

    % view (6, 88);
    disp ('yahhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh');

  case 2
    set (gcf, 'color', [0 0 0]);
    hold off;

    h_sub1 = subplot ('position', [0.05 0.6 0.2 0.3]); axis
('equal');
    cplot (x, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize); title ('Original V_x and Y');
    set(gca, 'box', 'on');

    h_sub2 = subplot ('position', [0.05 0.1 0.2 0.3]);
    cplot (vx, xmarker, xsize); hold on;
    cplot (y, ymarker, ysize); title ('Transformed V_x + Y');
    set(gca, 'box', 'on');

    h_sub3 = subplot ('position', [0.3 0.1 0.4 0.7]);
    cMIX_plot_mixture_simple (vx, T);
    cplotg ('black', vx, y, m, m_threshold); hold on;
    cplot (vx, xmarker, xsize); hold on;
    cplot (y, ymarker, xsize); title ('Transformed V_x + Y'); hold
on;
    axis('on'); set(gca, 'box', 'on'); title ('Transformed V_x + Y');

    h_sub4 = subplot ('position', [0.75 0.6 0.2 0.3]);
    % if mod(it_total,3) == 0     % this is expensive.
```

```
    switch (transformation_type)
      case 'tps'

   cplot (y, ymarker, ysize); hold on;
   if sum(sum(c)) ~= 0
     % ctps_plot_grid_simple('', vx1, vy, c,d); axis('equal');
     % axis('off');title ('TPS Warping');
     vx2 = [ones(siz1,1), x] * d; vx2 = vx2(:,2:dim+1);
     ctps_plot_grid_simple ('', x(:,1:2), y(:,1:2),
c(:,1:3),d(1:3,1:3));

     %cplot (vx, 'g+', xsize); hold on;
     cplot (vx, xmarker, ysize); hold on;
     axis('equal'); axis('off');title ('TPS Warping');
   end

      case 'gtm_tps'
   cgtm_plot_grid_simple ('tps_style', x,y,z,w, 0);
   axis('equal'); axis('off');title ('GTM Warping');

      case 'gtm_gaussian'
   cgtm_plot_grid_simple ('gaussian_style', x,y,z,w, sigma_kernel);
   axis('equal'); axis('off');title ('GTM Warping');
      otherwise; disp ('no');
    end;

    h_sub5 = subplot ('position', [0.75 0.1 0.2 0.3]);
    vy       = m * y ./  ( (sum(m'))' * ones(1,dim));
    % [vz]     = cMIX_warp_pts (transformation_type, z, z, c_tps,
d_tps, w, sigma_kernel);
    [P,jnk] = size(z);

    cplot (y,  'r.', ysize); hold on;
    cplot (z,  xmarker, xsize); hold on;
    %cplot (vz, 'g+', xsize); hold on;
    %cplot (vy, ymarker, xsize); hold on;
    %cplot_2g_simple ('', z, vz, eye(P,P), 0); title ('V_z + V_y');

end;

end;
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ----------------------------------------
% cMIX_plot_mixture_simple.m
% ------------------------------------------------------------------------
% Plot the clusters as bunch of ecllipses.
%
% Usage: [] = cMIX_plot_mixture_simple (vx, T);
% ------------------------------------------------------------------------

function [] = cMIX_plot_mixture_simple (vx, T, color_str);

% check input:
% ------------
if (nargin == 2)
  color_str = 'b-'; % default.
end;

[c,dim] = size (vx);

a = sqrt(T);
b = sqrt(T);

% Generate the ecllips:
% ---------------------
step_theta = 20;
tmp_theta  = [1:step_theta:360+step_theta]'/360*2*pi;
tmp_pts    = [a.*cos(tmp_theta), b.*sin(tmp_theta)];
n          = length(tmp_theta);

for i=1:c
  if (dim == 2)
    % Draw the ecllips:
    % -----------------
    plot (tmp_pts(:,1) + vx(i,1), tmp_pts(:,2)+vx(i,2), color_str);
hold on;
  else
    % disp ('plot3');
    plot3 (tmp_pts(:,1)+vx(i,1), tmp_pts(:,2)+vx(i,2),
zeros(n,1)+vx(i,3), color_str); hold on;
    %plot3 (zeros(n,1)+vx(i,1), tmp_pts(:,1)+vx(i,2),
tmp_pts(:,2)+vx(i,3), color_str); hold on;
    %plot3 (tmp_pts(:,1)+vx(i,1), zeros(n,1)+vx(i,2),
tmp_pts(:,2)+vx(i,3), color_str); hold on;

    %[X,Y,Z] = sphere(8);

    %X1 = X*T + vx(i,1);
    %Y1 = Y*T + vx(i,2);
    %Z1 = Z*T + vx(i,3); hold on;
    %mesh (X1, Y1, Z1); caxis([19 19.000000001]);hidden off;
  end;
end;
```

```
% Rigid Point Matching (ICP) Demo:
% ------------------------------------------------
% cMIX_normalize_m.m
% ------------------------------------------------------------------
% Double normalization of m (with outlier col/row).
%
% Input:  m, m_outlier_col, m_outlier_row -- m entries.
% Output: m, m_outlier_col, m_outlier_row -- normalized.
% ------------------------------------------------------------------

function [m, m_outlier_col, m_outlier_row] = cMIX_normalize_m (m, ...
    m_outlier_col, m_outlier_row);

% Parameters:
norm_threshold = 0.05;
norm_maxit     = 10;

[xmax, ymax] = size(m);

norm_it = 0;
while (1 > 0)

  % --- Row normalization ----------------------------------------
-
  sx = sum(m')' + m_outlier_col;
  m  = m ./ (sx * ones(1,ymax));
  m_outlier_col = m_outlier_col ./sx;

  % --- Column normalization -------------------------------------
-
  sy = sum(m) + m_outlier_row;
  m  = m ./ (ones(xmax,1)*sy);
  m_outlier_row = m_outlier_row ./sy;

  % time to quit?
  err = ((sx-1)'*(sx-1) + (sy-1)*(sy-1)')/(length(sx)+length(sy));
  if err < (norm_threshold .* norm_threshold); break, end
  % run out of time:
  norm_it = norm_it + 1;
  if norm_it >= norm_maxit; break; end

end

% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------
% 2 % %%% cMIX_calc_transformation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
function [c_tps, d_tps, w] = cMIX_calc_transformation
(transformation_type, ...
    lamda1, lamda2, sigma_kernel, x, vy, z);

c_tps = [];
d_tps = [];
w     = [];

switch (transformation_type)
  case 'tps'
```

```
    [c_tps,d_tps]  = ctps_gen (x, vy, lamda1, lamda2);
    % [c_tps, d_tps] = ctps_generate_cd_regularized (lamda1, lamda2,
x, vy);
  case 'rbf'
    [phi, w] = crbf_gen (x, vy, z, lamda1, lamda2, sigma_kernel);
  %case 'gtm_tps'
  %  [phi, w] = cgtm_calc_w ('tps_style', x, vy, z, lamda1, lamda2,
0);
  otherwise;
    disp ('# ERROR #: cMIX_calc_transformation -- wrong input!');
end
```

```
% Non-Rigid Point Matching (ICP) Demo:
% ------------------------------------------------
% ------------------------------------------------------------
% cMIX_calc_m_ICP.m:
% ------------------------------------------------------------
% Calc. two-way  ICP m.
%
% Usage:
% [m, dist_threshold] = cMIX_calc_m_ICP (vx,y);
% [m, dist_threshold] = cMIX_calc_m_ICP (vx,y,k_sigma);
% [m, dist_threshold] = cMIX_calc_m_ICP (vx,y,k_sigma,which_way);
%
% Notes: which_way -- specify one way ICP m.
%                0 -- both way (default).
%                1 -- x to y.
%                2 -- y to x.

function [m, dist_threshold] = cMIX_calc_m_ICP
(vx,y,k_sigma,which_way);

% check input:
if (nargin == 2)
  k_sigma = 0;   % default, no outlier rejection.
  which_way = 0;

elseif (nargin == 3)
  which_way = 0;

elseif (nargin == 4)

else
  disp ('# ERROR #: cMIX_calc_m_ICP -- wrong input !');
  help cMIX_calc_m_ICP; return;
end;


if k_sigma == 0;
  dist_threshold_flag = 0;
  dist_threshold      = 1e10;
else
  dist_threshold_flag = 1;
  dist_threshold      = 0;
end; % no outlier.

[siz1, dim]  = size(vx); xmax = siz1;
[siz2, temp] = size(y); ymax = siz2;

% Find nearest neighbour for each pt in x:
% ----------------------------------------
[M1, dist_x] = cICP_findneighbours (vx, y);
[M2, dist_y] = cICP_findneighbours (y, vx);

if dist_threshold_flag ~= 0
  dist    = [dist_x; dist_y];
  n       = length (dist);
  mean_x  = sum (dist) / n;
  sx      = std(dist); % sqrt ( 1 / (n-1) * sum ( (dist - mean_x) .*
(dist - mean_x)));
```

```
  dist_threshold = mean_x + k_sigma * sx;
end;


% this update thing of threshold doesn't work !!!
% ------------------------------------------------
% xdist_sum = sum (xdistances);
% dist_threshold = xdist_sum / (siz1/frac) * 3;
m1 = zeros(xmax,ymax); m2 = m1;
for i=1:xmax
  if dist_x(i) > dist_threshold; M1 (i) = -1; else; m1(i, M1(i)) = 1;
end;
end;
for j=1:ymax
  if dist_y(j) > dist_threshold; M2 (j) = -1; else; m2(M2(j), j) = 1;
end;
end;

if (which_way == 0)
  m = (m1 +m2)/2;
elseif (which_way == 1)
  m = m1;
elseif (which_way == 2)
  m = m2;
end;



%%%%%
% 1 % %%% cICP_fnneighbours
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
% Find nearest neighbour in template t for each point in x. 3d/2d
%
% Usage: [M, distances] = cICP_findneighbours (x, t);
% Notes:
%         M -- M(1) = 10, y(10) is nearest from x(1);
%         distance -- distance (x(i)-t(j))^2
%
% 02/01/00

function [M, distances] = cICP_fnneighbours (x, t);

[m, dim] = size(x);
[n, dim] = size(t);
M = zeros (m,1);

% |x-t| matrices:
% ---------------
xttmp = zeros (n, m);
for i=1:dim
  xtmp = ones(n,1) * x(:,i)';
  ttmp = t(:,i)  * ones(1,m);
  xttmp = xttmp + (xtmp - ttmp) .* (xtmp - ttmp);
end;

% M + min_dist list:
% ------------------
[min_dist, min_index] = min(xttmp);
distances = (sqrt(min_dist))';
```

```
M         = min_index';
% Non-Rogod Point Matching (ICP) Demo:
% ----------------------------------------------------

% cICP_findneighbours.m
% ---------------------------------------------------------------------
% Find nearest neighbour in template t for each point in x. 3d/2d
%
% Usage: [M, distances] = cICP_findneighbours (x, t);
% Notes:
%         M -- M(1) = 10, y(10) is nearest from x(1);
%         distance -- distance (x(i)-t(j))^2
% ---------------------------------------------------------------------

function [M, distances] = cICP_findneighbours (x, t);

[m, dim] = size(x);
[n, dim] = size(t);
M = zeros (m,1);


% |x-t| matrices:
% ---------------
xttmp = zeros (n, m);
for i=1:dim
  xtmp = ones(n,1) * x(:,i)';
  ttmp = t(:,i)  * ones(1,m);
  xttmp = xttmp + (xtmp - ttmp) .* (xtmp - ttmp);
end;



% M + min_dist list:
% ------------------
[min_dist, min_index] = min(xttmp);
distances = (sqrt(min_dist))';
M         = min_index';


% theta = transformation (1);
% t(1)  = transformation (2);
% t(2)  = transformation (3);
% s     = transformation (4);

% cs = cos(theta); sn = sin(theta); R = [cs, -sn; sn, cs];
% ry = y*R'*s;
% tr_y = ry;
% for k=1:2, tr_y(:,k) = tr_y(:,k) + t(k);, end

% for i=1:xmax
%
%   dist_tmp = 0;
%   dist_min = 0;
%   index    = -1;
%
%   dist_min = ((tr_y(1,:) - x(i,:)) * (tr_y(1,:) - x(i,:))');
%   index    = 1;
%
%   for j=2:ymax
%
%     dist_tmp = ((tr_y(j,:) - x(i,:)) * (tr_y(j,:) - x(i,:))');
```

```
%    if dist_tmp < dist_min
%       dist_min = dist_tmp;
%       index = j;
%     end;
%   end;
%
%   M(i) = index;
%   distances(i) = sqrt(dist_min);
% end;

% Non-Rogod Point Matching (ICP) Demo:
% ------------------------------------------------
% cResult.m
% ---------------------------------------------------------------------
% Draws a graph between RMS error and # of iterations
% with each algorithm
% ---------------------------------------------------------------------
err1=[0.141748
   0.139493
   0.128503
   0.126911
   0.120476];
err11=[0.135105
   0.130635
   0.126646
   0.124283
   0.118614];
err12=[0.126322
   0.081379
   0.072285
   0.070807
   0.072643];

err2=[0.086228
   0.086076
   0.0859944
   0.08509
   0.084467];

err21=[0.066240
   0.065467
   0.065034
   0.064833
   0.062526];

   err22=[0.152867
      0.134787
      0.109486
      0.094657
      0.067378];

err31=[0.190278
   0.190288
   0.189762
   0.189561
   0.190242];
err32=[0.075294
   0.069818
   0.069202
   0.068947
   0.067332];
```

```
    time11=[38
    40
    46
    52
    60];

time12=[39
    41
    47
    53
    61];

time13= [90
    100
    110
    125
    140];
time21=[35
    38
    43
    46
    55];

time22= [36
    39
    44
    47
    56];

time23=[65
    75
    90
    105
    120];


iter1= [31
    37
    42
    45
    55];

iter12=[155
    185
    210
    225
    275];

er1=[0.153069
0.151312
0.149068
0.140519
0.138637
0.133971
0.127891
0.125256
0.123791
0.122858
];

er2=[0.140544
```

```
0.138880
0.135669
0.132285
0.129080
0.126033
0.122944
0.120331
0.119079
0.118672

];

er3=[ 0.178279
0.164624
0.143055
0.093813
0.076645
0.072770
0.071426
0.071482
0.072257
0.072602
];
er21=[
0.122342
0.099882
0.095666
0.089309
0.086253
0.086128
0.085989
0.085908
0.085842
0.085492
0.084719
0.084471

];

er22=[

0.067810
0.056730
0.056763
0.056377
0.055964
0.055385
0.054845
0.054419
0.054111
0.053402
0.053216
0.053070
];

er23=[
0.199179
0.203814
0.206080
0.202850
0.164970
```

```
0.139995
0.126533
0.104823
0.092452
0.083868
0.077455
0.070130
];

er31=[0.316173
0.222866
0.196807
0.187654
0.189114
0.190221
0.190566
0.189749
0.189753
0.190637
0.190739
0.190404
0.190238
];

er32=[0.095008
0.082544
0.077977
0.077150
0.076168
0.075458
0.070439
0.069702
0.068793
0.068291
0.067793
0.067439
0.066996
];

er33=[0.393020
0.351857
0.252412
0.171334
0.115613
0.086319
0.072572
0.070350
0.068256
0.066224
0.061299
0.059170
0.057871
];

er41=[
0.134906
0.115658
0.110010
0.100553
0.088624
0.086921
```

```
0.087083
0.086233
0.086169
0.086857
0.086694
0.086318
0.086036
];

er42=[0.065276
0.059264
0.059114
0.058931
0.058541
0.058169
0.057662
0.056919
0.056939
0.056445
0.055602
0.055276
0.054920
];

er43=[
0.255769
0.262750
0.266250
0.262324
0.213520
0.145293
0.095455
0.082713
0.077772
0.073500
0.069766
0.060945
0.052823
];
iter=[
 5
10
15
20
25
30
35
40
43
46
49
51
53
];
figure(1);
%title('Different results relatied to figure 4');

hold on
plot (iter,er41, '- r*', 'LineWidth',2);
%text(0.126, 47,'\leftarrow ICP with TPS only', 'FontSize', 12);
plot (iter,er42, '- b*', 'LineWidth',2);
```

```
%text(0.132, 35,'ICP with TPS & CG \rightarrow', 'FontSize', 12,
'HorizontalAlignment','right');
%figure(2)

plot (iter,er43, '- g*', 'LineWidth',2);
%title('RPM results related to figure 1');
hold off

ylabel('RMS error');
xlabel('Number of iterations');
%set(gca,'XTick',5:10:55)
%set(gca,'YTick',-0.1:0.05:0.3)
%set(gca,'YTickLabel',{'0','0.05','0.1','0.15','0.2','0.25','0.3'})
```

%text(0.125, 50,'\leftarrow ICP with TPS only', 'FontSize', 12);

References

[1] P. K. Banerjee and A.W. Toga. Image alignment by integrated rotational and translational transformation matrix. *Physics in medicine and biology*, 39:1969–1988,1994.

[2] G. Barequet and M. Sharir, Partial Surface and Volume Matching in Three Dimensions, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(9), Sep 1997

[3] S. Belongie, J. Malik and J. Puzicha, Matching Shapes, *International Conference on Computer Vision ICCV*, 2001.

[4] R. Benjemaa, and F. Schmitt, Fast Global Registration of 3D Sampled Surface Using a Multi-Z-Buffer Technique, *Proc. 3DIM*, 1997.

[5] P.J. Besl and R.C. Jain, Three-Dimensional Object Recognition. *Computing Surveys*, Vol. 17, pp. 75-145, March 1985.

[6] P.J. Besl and N.D. McKay, A Method for Registration of 3-D Shapes, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239-256, Feb. 1992.

[7] R. Bhatia. *Matrix Analysis*, volume 169 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1996. page 38.

[8] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Patt. Anal. Mach. Intell.*, 11(6):567–585, June 1989.

[9] L. G. Brown, A survey of image registration techniques. *Computing Surveys*, Vol. 24, pp. 325–376, December 1992.

[10] K. Brunnstrom and A.J. Stoddart, Genetic Algorithms for Free-Form Surface Matching, *13th Int. Conf. on Pattern Recognition, pp D689-673, Vienna, Austria,* 1996.

[11] M. Celenk, Three-dimensional object recognition using cross-sections, *27th Southeastern Symposium on System Theory (SSST'95)*, March 12 - 14, 1995

[12] C.S. Chua and R. Jarvis, 3D free-form surface registration and object recognition, *Int'l J. of Computer Vision*, Vol. 17, pp. 77-99, 1996.

[13] H. Chui, Non-Rigid Point Matching: Algorithms, Extensions and Applications. Ph.D. Thesis, Yale University, 2001

[14] C. Dorai, A. Jain, COSMOS - a representation scheme for 3D free-form objects, *IEEE Transaction Pattern on Pattern Analysis and Machine Intelligence,* 19(10): pp. 1115-1130, 1997.

[15] J.S. Duncan, and N. Ayache, Medical Image Analysis: Progress over Two Decades and the Challenges Ahead, *IEEE Trans. Pattern Analysis and Machine Intelligence* , Vol. 22, No. 1, Jan 2000.

[16] J. Feldmar and N. Ayache. Rigid, af£ne and locally af£ne registration of free-form surfaces. *Intl. J. Computer Vision,* 18(2):99–119, May 1996.

[17] A.W. Fitzgibbon, Robust Registration of 2D and 3Dpoint Sets. Department of Engineering Science, University of Oxford

[18] S. Gold, A. Rangarajan, C. P. Lu, S. Pappu, and E. Mjolsness. New algorithms for 2-D and 3-D point matching: pose estimation and correspondence. *Pattern Recognition*, 31(8):1019–1031, 1998.

[19] S. Gold and A. Rangarajan, A Graduated Assignment Algorithm for Graph Matching, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377-388, April 1996.

[20] S. Granger, X. Pennec, and A. Roche, Rigid Point-Surface Registration using Oriented Points and an EM variant of ICP for Computer Guided Oral Implantology, *Research report RR-4169, INRIA,* 2001.

[21] E. Guest, M. Fidrich, S. Kelly, E. Berry, Robust Surface Matching for Registration, *3DIM,* 1999.

[22] M. Hebert, K. Ikeuchi and H. Delingette, A spherical representation for recognition of freeform surfaces, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7): 681- 689, July 1995.

[23] L. S. Hibbard and R. A. Hawkins. Objective image alignment for three dimensional reconstruction of digital autoradiograms. *J. Neurosci. Methods*, 26:55–75, 1988.

[24] B.K.P. Horn, Extended Gaussian Image, *Proc. IEEE*, Vol. 72, pp. 1671-1686, 1984.

[25] H.Hugli, C. Schutz, Geometric Matching of 3D Objects: Assessing the Range of Successful Initial Configurations. *0-8186-7943-3/97 IEEE*

[26] A. Johnson, Spin-Images: a representation for 3-D surface matching, Ph.D. Thesis, CMU-RITR-97-47, Robotics Institute, Carnegie Mellon University,1997.

[27] T. Joshi, J. Ponce, B. Vijayakumar and D.J. Kriegman, HOT curves for modeling and recognition of smooth curved 3D objects, *Proc. IEEE conf. Computer Vision and pattern recognition*, Seattle, Wash., pp.876-880, June, 1994.

Intelligence, 15:2, pp. 119-132, 1994.