Tree Search and SingularValue Decomposition: A Comparison of Two Strategies For Point-Pattern Matching

by

Philip Ifrah

B. Eng.

A thesis submitted to the Faculty of Graduate Studies and
Research in partial fulfillment of the requirements
for the degree of Master of Engineering

Department of Electrical Engineering
McGill University
Montréal, Canada
August, 1996

Abstract

Two approaches for solving point-pattern matching problems are compared; namely, a graph-matching algorithm [1] and an SVD-based procedure [2]. In both cases, the features that are used in the matching process are point coordinates in Euclidean n-space, $\mathbb{E}^n$. The patterns being matched are assumed to be related by a combination of two transformations: (1) a permutation of the feature points which establishes the correspondence between the feature points of the different patterns and (2) a global geometric transformation based on rigid motions which aligns the patterns once the point correspondences are known. The problem of finding the first transformation, known as the *point correspondence problem*, is the most demanding part of the matching process in terms of computational requirements; accordingly, the focus is placed on the algorithms' ability to establish point correspondences. Computer simulations are used to evaluate the performance of the algorithms' respective search strategies in terms of both the accuracy of the final solution and the speed with which the solution is obtained. In all of the experiments, the performance of the graph matching algorithm is clearly superior to that of the SVD-based method in terms of both speed and accuracy; however, it is shown that the computational requirements of the tree search procedure used by the graph matching algorithm are strongly dependent on factors such as the magnitude of the noises that are contained in the patterns and on the mutual distances between the feature points. The major weakness of the SVD-based algorithm is its inconsistency in converging to the expected solution, especially when extra or occluded points are present in one or more of the patterns to be matched.

Sommaire

Deux façons pour résoudre de divers problèmes d'association de formes sont comparées; à savoir, un algorithme d'association de graphes [1] et une procedure basée sur la décomposition en valeurs singulières (DVS) [2]. Dans les deux cas, les charactéristiques utilisées dans le procédé d'association sont des points dans l'éspace euclidéen de dimension n, $\mathbb{E}^n$. Les formes en question sont presumées etre liées par une combinaison de deux transformations: (1) une permutation des points charactéristiques qui établit la mise en correspondence des points charactéristiques des diverses formes, et (2) une transformation géometrique globale basée sur l'hypothèse des mouvements rigides qui est supposée aligner les formes lorsque la mise en correspondence est connue. Le problème pour trouver la première transformation, connue comme le *problème de mise en correspondence des points*, est la plus astreignante partie du procédé d'association en terme des exigeances de calcul; par conséquent, la concentration est surtout placée sur la compétence des algorithmes d'établir la mise en correspondence des points. Des simulations sont éffectuées pour evaluer la performance des strategies de recherche de chaque algorithme, en termes de la précision de la solution finale ainsi que la rapidité avec laquelle la solution est obtenue. Dans toutes les experiences, la performance de l'algorithme d'association de graphes est nettement supérieure à celle de la méthode DVS en terme de vitesse et de précision; cependant, il est démontré que les exigeances de calcul de la procedure de traversée d'arbre qui est utilisée par l'algorithme d'association de graphes sont fortement dependants de plusieurs facteurs tels que l'intensité des bruits et des distances mutuels entre les points charactéristiques. La majeure faiblesse de la procedure DVS est son inconsistence à converger vers la solution attendue, particulièrement dans le cas où le nombre de points dans les deux formes n'est pas ègale.

Acknowledgments

First of all, I would like to thank my thesis supervisor, Dr. S.D. Morgera, for his careful guidance and support throughout the course of my research. His enthusiasm and encouragement vis-à-vis my work was truly inspiring and has helped me to learn a great deal about the research process.

I would like to thank my friends in the INSL lab who were always there to lend a helping hand: Sam Torrente, Helgi Sigurdsson, Dan Gravelle, Pietro Fionda, and Jihad Hallik, to name only a few. Without them, my experience would surely have been greatly diminished.

Finally, I would like to express my deepest gratitude to my family whose support was of paramount importance in completing this thesis. Their advice, guidance, and support throughout my studies has helped me to achieve my goals with relative ease and with great success.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pattern matching problems that involve two or more data sets which are related by a geometric transformation have applications in a broad range of scientific areas. In computer vision, pattern matching plays a key role in applications such as object recognition, target tracking, autonomous vehicle navigation, robot guidance, and the dynamic monitoring of production processes. The inherent usefulness of pattern matching problems has also led to significant applications in areas such as photogrammetry, image registration, aerial reconnaissance, image coding, and others. In many cases, the application in which the matching problem arises has a strong influence on the development, or use, of a particular algorithm to find the solution. This situation has led to the development of enumerable methodologies for solving various forms of the basic pattern matching problem. In this study, two recently proposed algorithms, one based on the singular-value decomposition (SVD) of a matrix [2], and the other based on a recursive tree search procedure [1], are compared.

1.1 Overview of Point-Pattern Matching

When the data sets to be matched consist of point coordinates in n-dimensional Euclidean space $\mathbb{E}^n$, where $n > 0$, the problem is called *point-pattern matching*, or simply *point matching*. The two algorithms compared in this study are point-matching techniques. In general, point-matching is composed of two distinct subproblems, which are commonly called the *motion estimation problem* and the *point correspondence problem*. These subproblems are often solved separately from one another, using different techniques; however, a single algorithm is sometimes used which encompasses both problems. An overview of the two previous point-matching subproblems is given in the sequel.

1.1.1 The Motion Estimation Problem

In this subproblem, a geometric transformation is sought which aligns the data sets as closely as possible. If the point coordinates in the data sets to be matched are obtained from 2- or 3-dimensional objects, then the geometric transformation aligns the data sets such that corresponding objects in the data sets have the same orientation and position in a given frame of reference; furthermore, if the objects are rigid, or undeformable, then the geometric transformation describes a rotation through one or more angles and a translation. In the previous case, the goal of the motion estimation problem is to recover the so-called *motion parameters* (*i.e.*, rotation angle(s) and translation vector) from the estimated geometric transformation. The assumption of rigid objects applies in many cases, and is used in a large majority of the matching algorithms in the literature. Both the SVD-based algorithm and the tree search procedure which are compared in this thesis assume rigid point configurations.

1.1.2 The Point Correspondence Problem

In this subproblem, a transformation is sought which establishes the correspondences between the points in the two data sets to be matched. Many point matching problems involve data sets for which point-to-point correspondences between the data sets have not been established; moreover, the feature points in one of the data sets may not appear in the other set due to occlusion from other objects, self-occlusion as points rotate out of view, shadows, *etc.* This situation can be represented by an appropriate permutation of the points in one of the data sets.

If the data sets to be matched are called the *template* set and the *sensed* set, then the goal of the point correspondence problem can be stated as follows: for each point in the template set, the corresponding point in the sensed set must be found, if it exists. The mapping of points in the template set onto the sensed set which is obtained as the solution to this problem is the desired transformation. In general, the computational complexity of the point correspondence problem is significantly greater than that of the motion estimation problem; accordingly, the emphasis in this study is placed on the problem of finding point correspondences.

1.2 Outline of the Thesis

The thesis is organized as follows:

In Chapter 2, a review of the pertinent literature on pattern matching is presented. This chapter provides both an overview of various applications in which matching problems arise and a survey of the different techniques which are used to solve them. The overview of applications section presents several practical uses of matching techniques, while introducing some of the commonly used terminology. The survey section presents a historical perspective on pattern matching and provides references to the

numerous approaches which have been used to solve the point matching problem.

Chapter 3 presents the fundamental concepts which are used in most pattern matching schemes. Specifically, a number of "basic elements" are described which are commonly found in the literature on point-matching techniques. The latter elements are useful for describing matching problems in general terms, and provide a framework for comparing the two matching techniques which are the focus of attention in this study.

The SVD-based algorithm is described in detail, in Chapter 4. This algorithm is based on a function optimization approach for solving both the motion estimation problem and the point correspondence problem. The SVD-based procedure is essentially a specific implementation of the theoretical framework developed by Brockett [3]. In Chapter 4, an overview of the concepts behind the approach is given, and all of the relevant aspects of the algorithm are described in detail.

The tree search procedure is described in detail in Chapter 5. This algorithm, by Cheng and Don [1], is a graph-matching approach for solving the point correspondence problem. Some basic concepts related to graph theory and tree search methods are provided as background material before describing the graph-based point matching algorithm.

All comparisons between the two matching algorithms are made in Chapter 6. For the most part, this chapter describes the computer simulations that were used to compare the two algorithms. Both the dynamic behaviour of the algorithms and the relative speed and accuracy with which an optimal match is obtained using these techniques are examined. A summary and discussion of the computer simulations is given in Chapter 7.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents a review of the pertinent literature on pattern matching. In Section 2.2, an overview of applications in which matching problems arise is presented. The overview gives a broad perspective on matching problems and their practical uses, and also provides a context in which specific techniques can be discussed. Section 2.3 gives a survey of existing methodologies for solving point pattern matching problems. In the survey, a broad range of techniques is presented, to provide a global view of the different approaches which are used to solve various point pattern matching problems.

2.2 Overview of Applications

A broad perspective on matching problems and associated techniques can be acquired by considering a number of different applications where pattern matching has been used. In this section, some of the most prevalent applications in areas such as computer vision, image registration, and a few others will be discussed.

2.2.1 Vision-Based Applications

In computer vision, pattern matching is fundamental to many applications such as object recognition, scene analysis, motion prediction, and trajectory planning. Historically, the computer vision and robotics communities have been among the most productive in developing matching algorithms and also in publishing literature on the subject. The important role which pattern matching plays in vision-based systems, combined with steady improvements in both sensor and computing technologies, have spurred a great amount of activity related to matching techniques over the past few decades. Moreover, fairly recent developments in range sensing devices have made matching techniques using 3-dimensional data points popular in computer vision applications.

Matching strategies play a key role in object recognition, where the goal of the vision system is to locate and identify objects in a particular scene. Typically, a vision system is equipped with a camera and other sensing devices which it uses to gather data from its environment. From this data, it must derive an interpretation of objects in a scene before proceeding to carry out a given task. Common tasks include the assembly or inspection of manufactured parts, the analysis of microscopic images, and the navigation of robots or autonomous vehicles. A general paradigm in model-based object recognition is illustrated in Figure 2.1. At the data collection stage, a camera or some other sensing device is used to obtain raw data which usually needs to be processed before yielding any useful information. Once the data has been collected, it is passed through a number of low-level processing stages to extract an appropriate set of so-called *features* which are later used by the matching algorithm. Low-level tasks such as segmentation, where individual objects in an image are separated from one another, and the process of extracting the same features on an object for each image used in the matching process, are important problems in their own right; however, such issues are beyond the scope of this study and will not be covered here. Once the low-

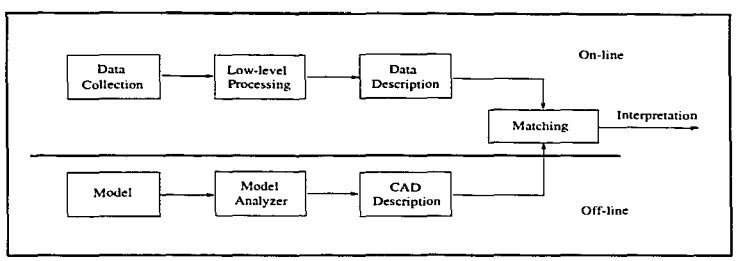


Figure 2.1: A general paradigm in object recognition. Adapted from [3].

level processing is complete, the data is used to establish a rough object representation which is eventually compared with object models that have been previously stored in memory.

Object models are typically constructed either using actual objects or an appropriate computer-aided design (CAD) system. When actual objects are used, data points on the object are collected from several viewpoints and are then integrated in a coherent fashion to form the model. In the case of CAD systems, a set of predefined primitives are used by a computer programmer to interactively describe a particular object. When the vision system is operated, the object models serve as a basis against which object descriptions derived from the data are compared. By using an appropriate matching strategy, an object sensed by the vision system may be identified as one of the known object models, and its orientation relative to the model can be determined as well. A recent and comprehensive survey of techniques used in model-based object recognition is found in [4].

Pattern matching also plays an important role in computer vision problems where the motion and structure of moving objects is to be inferred from a sequence of images. Many applications exist in which relative motions between objects and sensing devices are captured by time varying imagery. Common applications include

target tracking, dynamic monitoring of production processes, cloud and weather systems tracking and robot guidance. Typically, the problems of recovering both the structure and motion of objects from a time-ordered sequence of images can be solved independently [5]. In structure estimation, the main objective is to estimate the positions of one or more objects in space from the observation of a number of feature points in two or more distinct images [1]. When the objects are static, their positions are typically derived from a pair of images taken from different viewpoints. The problem of recovering scene geometry just described is referred to as *stereoscopic depth measurement* when the relative displacement of the two viewpoints is known *a priori*. Given the displacement between two views of an object, a simple method based on triangulation can be used to recover depth information from a pair of 2-D images [6]. Often, however, the displacement between the two viewpoints is not known and the method of triangulation cannot be used. Instead, the 3-D locations of objects in space and the displacement between views must be inferred from two or more 2-D images.

Once the three-dimensional object positions are known, the so-called motion parameters can be obtained which describe the relative motion between objects and sensors in the time varying images. When image sequences are used to derive the motion parameters, the results can either be extrapolated to predict future motions or interpolated to recover intermediate motions [7]. Such predictions are key elements in planning safe trajectories for autonomous vehicle navigation or in determining grasp sites for robot manipulators. In both structure recovery and motion estimation, pattern matching plays an important role. Details regarding the use of matching techniques in both of the previous problems are deferred to Section 2.3.

[1] Alternatively, a number of laser-based techniques may be used to directly obtain a set of 3D coordinates, or so-called *range data*, from the environment [4].

2.2.2 Image Processing Applications

Another widely used application of pattern matching is image registration [8]. A problem frequently encountered in image analysis is the need to compare images which have been acquired at different times, by different sensors or from different viewpoints. The goal of image registration in such cases is to align the images such that meaningful disparities between them are highlighted. A similar problem which commonly arises in image processing involves the search for a template or reference structure within an image. The previous problems are actually quite similar to the ones described earlier in the context of computer vision systems; in fact, matching techniques developed for image registration are often used in vision systems and vice-versa. Common uses of image registration techniques include character recognition, diagnostic medical imaging, and remotely sensed data processing for both civilian and military applications.

The use of image registration techniques in medical image analysis illustrates an important application, where data acquired from different sensors is integrated to provide complementary information. For each of the various modalities used in medical imaging, including X-Ray Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET), and others, a particular type of sensor technology is used to collect data from patients. In each case, the range of imaging applications in which the sensors are suitable is limited. For instance, CT and MRI are particularly well suited for displaying anatomical structures, but provide little functional information [9]. On the other hand, PET scans are mostly useful for delineating functional and metabolic activity, but delineate anatomy poorly [9]; therefore, images obtained from the different modalities provide complementary information. CT and MRI also provide complementary morphologic information, since CT is especially well suited for depicting bony structures while MRI is better for visualizing soft tissue pathology and nervous tissue [10]. By combining the data obtained from different modalities, the shortcomings of various imaging techniques, taken individually, can be

overcome to yield images with more complete diagnostic information.

As an example, consider a case where a patient's liver is to be examined from a set of medical images. An image of the patient's liver using MRI would be appropriate to depict anatomical structures and could be used to plan a medical procedure since it resembles what a surgeon would see during an operation [8]. However, functional information is poorly illustrated by the MRI image and must be obtained through some other imaging modality. A SPECT (Single Photon Emission Computed Tomography) image could be used for this purpose, after administering a suitable radio-nuclide compound to the same anatomical region depicted in the MRI image [8]. The SPECT image would highlight some of the functional behaviour of the liver and could be used to distinguish between cancers and other benign lesions. By combining the complementary information available in the MRI and SPECT images, both improved diagnosis and better surgical planning are possible. But, since the two images are typically taken at different times, using different resolutions, and from different viewpoints they cannot simply be overlaid [8]. By using a matching technique, the transformation mapping image points in one image directly onto image points in the other image can be found, and can then used to effectively combine the information provided by the MRI and SPECT modalities.

2.2.3 Other Applications

Matching problems also arise in other areas such as astronomy, psychology, and telecommunications. In astronomy, a problem frequently encountered is the matching of star lists against catalog information. Based on two-dimensional photometric data, the centroids (*i.e.* center of mass locations) of star positions can be derived to establish so-called star lists [11]. Both new and previously acquired star lists may need to be compared with cataloged information either for identification or to observe possible changes. Since the information used to derive separate star lists is obtained at dif-

ferent times, and from different viewpoints, cataloged star lists are not likely to be the same as newly acquired ones. Typically, star lists differ in cardinality (*i.e.* , the number of points in the lists), ordering of points, and a transformation which includes translation, rotation and scaling [11]. The star matching problem, therefore, consists of finding point correspondences between two star lists and the rigid transformation which best relates the two point sets. In psychology, researchers sometimes need to compare data obtained via non-metric multidimensional scaling techniques with similar data compiled using more traditional factor-analytic techniques [12]. Since data obtained using the previous techniques can be related through an appropriate choice of rotation, translation, and scaling, a matching scheme can be used to find the rigid transformation which allows meaningful comparisons to be made [12]. In telecommunications, matching techniques have been used for inter-frame image coding [13], [14] [15]. By estimating the trajectories of moving objects in a sequence of images, a significant gain in data compression can be achieved [14].

2.3 Survey of Approaches

2.3.1 Early Approaches

Early approaches for solving point-matching problems can be traced back to Thompson [16], in the area of photogrammetry. The problem studied in [16] is a classic photogrammetric task, known as the problem of absolute orientation: given measurements of the coordinates of several points in different coordinate systems, the objective is to find the transformation which relates the two systems. In [16], Thompson finds a least-squares solution for the absolute orientation problem for the case in which exactly three points are measured. Techniques developed later by Oswal [17] and then Horn [18] are somewhat more general and have the advantage of being able to handle more than three points. Schönemann and Carroll [12] treat a similar problem, while

attempting to compare point configurations obtained through the multidimensional scaling of psychological data. Like Horn [18], Schönemann and Carroll develop closed-form expressions which can be solved noniteratively. In particular, the latter authors use an Eckart-Young decomposition [19] of an arbitrary symmetric matrix to find a solution. The SVD-based method developed by Arun *et al* [20], and later refined by Umeyama [21], is closely related to the previous approach.

2.3.2 Methods Based on Projective Relations

A substantial number of point matching algorithms have been developed, in the context of computer image analysis, for the purpose of deriving both the motion and structure of moving objects from a series of images. Early studies in motion estimation include methods for analyzing the two-dimensional movement of clouds from a series of satellite images [22] [23], techniques for reducing the bandwidth of television signals through motion-compensated video coding [24] [25], and others. Since then, numerous algorithms have been devised with the specific intent of extracting not only motion estimates, but also object structure from a series of images.

In many cases, the motion and structure of objects is described in terms of image positions, recorded from different viewpoints of a given scene. Typically, approaches to solving the pattern matching problem based on the use of image positions heavily depend on a geometric description of the image formation process used in obtaining the observed images. Based on the imaging model used, a number of projective relations involving both the three-dimensional coordinates of the points and the parameters of motion are exploited to derive a set of equations. Most often, the so-called central projection model is used to formulate the problem, although other possibilities include spherical, and parallel projection models. As a result, large sets of complicated non-linear equations are derived, which usually must be solved using iterative numerical algorithms (*e.g.* , [26]). Moreover, the solutions obtained are often sensitive to noise

and to the choice of initial guesses for the unknowns [26], [27]. Smoothing, or overdetermination of the estimation equations, by using a much larger number of match points than equations, has been suggested to reduce the effects of noise [26]. Alternatively, a larger number of image frames may be used [28], [29].

Matching strategies based on projective relations alone have generally been considered too computationally complex for many applications; consequently, a wide variety of simplifying assumptions have been used to render the problem tractable. Some of these simplifications include the use of parallel projections to model the imaging process [30], restricted motion (*e.g.* , small angle rotation [31], and pure translation or pure rotation [32]), restricted geometric configuration of objects (*e.g.* planar patches [33], curved surfaces [34], points on a convex hull [35]) and known object geometry [28]. Unfortunately, matching algorithms derived under the previous restrictions are often only approximately applicable to real images; therefore, alternatives to solving the basic nonlinear equations have been sought. Notably, some researchers have shown that when a sufficient number of point correspondences are available (usually 7 or more), the matching problem can be uniquely solved with a set of *linear* equations [36], [34]; however, solutions obtained with the linear algorithms appear to be highly sensitive to noise [27].

2.3.3 Methods Based on Optical Flow

Another way to describe the motion and structure of objects in images is in terms of *optical flow*. In many computer vision applications, the motion of objects is recorded in terms of "optical velocities", or the projection of the three-dimensional velocities of points moving in space onto a two-dimensional image plane. The resulting field of two-dimensional velocities on the image plane, also called the "retinal surface", is referred to as *optical flow*. Matching problems formulated in terms of optical flow are generally based on one of two broad classes of techniques used to compute this

quantity. The first class of methods compute optical flow by tracking characteristic brightness patterns from one frame to another in a time-ordered sequence of images, and are collectively referred to as *feature-based* techniques [6]. The second class of methods, known as *gradient-based* techniques, derive optical flow through an equation which relates optical velocities to spatial and temporal changes in an image, *i.e.*,

$$\frac{\partial f}{\partial x}u + \frac{\partial f}{\partial y}v + \frac{\partial f}{\partial t} = 0,$$

where f is the image function, t is time, and u and v are the x and y components of optical velocity [6]. Again, a set of nonlinear equations is generally derived by exploiting projective relations involving point positions, and by decomposing motion into a rotation and a translation [37], [38], [6]. Hence, aside from the problems associated with obtaining accurate velocity measurements, the difficulties involved in solving systems of nonlinear equations still remain.

2.3.4 Methods Based on Unit Quaternions

When matching is performed over three-dimensional space, a different approach to the matching problem, based on unit quaternions, is possible. Essentially, a 3-D rotation can be represented by a quaternion of unit norm, $\bar{\mathbf{q}}$, where

$$\bar{\mathbf{q}} = \begin{Bmatrix} \mathbf{Q} \\ q \end{Bmatrix} = \begin{Bmatrix} \mathbf{X}\sin(\theta/2) \\ \cos(\theta/2) \end{Bmatrix},$$

$\mathbf{X}$ is a (3×1) column vector representing the axis of rotation, and θ is the angle of rotation about $\mathbf{X}$ [39]. If the rotation matrix Θ is written in terms of the latter (4×1) vector, a quadratic expression for the matching error, $\epsilon(\Theta)$, can be obtained [40]. Shuster [40] appears to have been the first to show that the optimal quaternion which minimizes $\epsilon(\Theta)$ is the eigenvector associated with the largest eigenvalue of a symmetric (4×4) matrix. Similar results have been obtained by Faugeras and Hebert [41] and Horn [42], although the latter two references extend Shuster's results somewhat to resolve issues that arise in 3-D scene analysis.

2.3.5 Methods For Finding Point Correspondences

All of the techniques previously mentioned tacitly assume that the point sets to be matched are *ordered*; specifically, they assume that the so-called point correspondence problem has been solved. Several approaches for solving the point correspondence problem alone have been devised (*e.g.*, [1], [43], [44], [45], [46], [47]). Many of these approaches employ a tree search procedure to execute an "exhaustive" search of all possible solutions to the problem. Actually, the extent of the search is usually limited by applying various constraints which rule out unlikely paths in the search tree. Without such constraints, the computational requirements of an exhaustive search could quickly become unwieldy. For instance, if the data sets contain p points and q points, respectively, then the total number of potential pairings between the points in the two data sets is [47]:

$$p(p-1)(p-2)\ldots(p-q+1) \ , \tag{2.1}$$

when $p \geq q$. If the number of points in the data sets is roughly equal (*i.e.*, $p \approx q$), then the total number of potential solutions to the matching problem is $\mathcal{O}(p!)$; accordingly, the computational complexity of the tree search can quickly become unmanageable as the number of feature points to be matched increases. The efficiency of the tree search can, however, be vastly improved by identifying and eliminating the paths in the tree that do not lead to an optimal solution. This is done either by incorporating simple calculations, such as "forward checking" or "looking ahead" [44] in the search procedure or by guiding the search according to an application-specific similarity metric. Often, so called geometric constraints serve as the basis for constructing the similarity metric; although, different criteria are sometimes used (*e.g.*, [45]).

The use of geometric constraints to reduce the complexity of the tree search is common in the literature. Grimson and Lozano-Perez [43] have devised a technique for identifying and locating objects which can be modeled as polyhedra, by generating an exploring a so-called *interpretation tree*. The number of potential paths to be

explored in the search tree is reduced by using local constraints on distances between object faces, angles between face normals, and angles of vectors between sensed points. The approach proposed by Chen and Huang [47] is based on the assumption that the point patterns are subjected only to rigid, or undeformable, motions; in this case, local distance and angular constraints are used to reduce the size of the search tree. Cheng and Don [1] exploit similar relationships between the feature points that are based on the rigid motion assumption. Their method uses the point sets to construct connected graphs whose vertices are matched by a recursive tree traversal algorithm. Other graph-based approaches include: [48], [49], [50], [51]; notably, Gmür and Bunke [48] use a tree search algorithm whose efficiency depends on a combination of geometric constraints and neighbourhood relations between adjacent nodes in the graphs that are being matched.

Several methods have been devised to solve for both point correspondences and rigid motion parameters in a single unified approach. In Part I of Lin et al. [52], a joint Fourier transform and correlation method is proposed to determine the 3-D motion parameters for point sets without correspondences. Unfortunately, the cost functions used in the algorithm, in addition to the parameters involved in their evaluation, appear to have been selected in an *ad hoc* manner; moreover, the important effects of noise and occluded points are not taken into consideration. The so-called iterative closest point (ICP) algorithm proposed by Besl and McKay [53] handles both noisy data and occluded points, provided that every point in the sensed data set can be paired with a corresponding point in the template, or "model", set. This method solves the point correspondence problem by computing the closest point on a geometric entity to a given point in the sensed data set. The specific geometric entity that is used in the computation depends on the internal representation of the model data and must be known *a priori*. Once the closest point procedure is completed, the motion parameters are obtained using a quaternion-based algorithm, such as the one described by Horn [18]. The entire procedure (*i.e.*, closest point computation and registration) is repeated

until a mean-square distance metric is minimized. The rate at which the ICP algorithm converges to a local minimum is dictated by the choice of a set of initial rotations and translations which are selected according to the "shape complexity" of the model data.

A theoretical treatment of the matching problem for both ordered and unordered point sets has been presented by Brockett [3] and is based on the theory of Lie groups and Lie algebras. The theoretical framework laid down in [3] essentially transforms the point-matching problem from a difficult combinatorial optimization problem to a significantly simpler "calculus-type" one; consequently, a number of matrix equations are derived, for various types of matching problems, which can be optimized in a minimax sense. Morgera and Lie Chin Cheong [54, 2] have adopted the theory in [3] to develop an iterative algorithm for solving the equations that were derived by Brockett. The latter algorithm is implemented using a combination of the steepest ascent/descent method and singular value decomposition (SVD) to find the point correspondences and the motion parameters, respectively; although, either of the two previous methods could be used alone to solve the general unordered point-matching problem. The use of the SVD for solving point-pattern matching problems has many precedents in the literature. Schönemann and Carroll [12], Horn [18], Arun *et al.* [20], and Umeyama [21] have all devised techniques which are based on the SVD for solving the motion estimation problem; however, none of the previous approaches attempt to solve the point correspondence problem.

In this study, the algorithm proposed by Morgera and Lie Chin Cheong [2] is implemented, using the SVD to find both the point correspondences and the motion parameters. This implementation is used to make comparisons, in terms of speed and accuracy, with the graph matching approach proposed by Cheng and Don [1]. All comparisons are made with regard to the point correspondence problem; although, the SVD algorithm is used to find the motion parameters at times, for illustrative purposes.

Chapter 3

Elements of Pattern Matching

3.1 Introduction

Given the diversity of both existing matching problems and the techniques that are available for solving these problems, it is useful to identify some common attributes that can be used for purposes of general discussion and comparative analysis. In this chapter, some basic elements are described which are common to most matching schemes. The latter elements are useful for describing matching problems in general terms, and provide a framework within which various matching techniques can be compared.

Methodologies for solving point-matching problems can be characterized through different combinations of choices for the following components: a feature space, a geometric transformation, a search strategy, and a similarity metric. The *feature space* defines the type of information in each data set that will be used for matching. This information typically describes the salient features of the data sets that can be used in the matching process. Given the feature space, a class of mathematical transformations is selected that can presumably be used to align the patterns. This class of transfor-

mations defines all the possible relationships that can exist between the data sets to be matched. A *search strategy* is then devised, which determines how the features which have been extracted from the data are to be tested in the search for an optimal transformation. At this point, the definition of an optimal match is determined. The degree of success in matching that has been achieved is measured by the *similarity metric*. Typically, matching proceeds according to the search strategy until a transformation is found which optimizes the similarity metric in some sense. In virtually all matching schemes, both the design and success of the technique are influenced by the choices made for each of the four previous components. Accordingly, these components may be viewed as basic elements of pattern matching. In the sequel, each element is considered in detail.

3.2 The Elements

3.2.1 Feature Spaces

Most matching techniques avoid analyzing large amounts of data by using only a characteristic subset of the data in the matching process. This subset is typically obtained through a preprocessing step which both identifies and extracts a set of so-called *features* or *tokens* from the raw data. Commonly used features include: points, edges, contours, surfaces, and image intensities; statistical features such as centroids or moment invariants; and, higher-level structural and syntactic descriptions [8]. The group of features obtained from the preprocessing step defines the feature space of a particular matching procedure. If the same type of feature is extracted from each data set involved in the matching process, only elements that are contained in the feature space need to be matched. In this way, the amount of data to be analyzed in the matching process is significantly reduced.

As an example illustrating the selection of a feature space, consider the problem of aligning two radiographic images of a patient at different times, as shown in Figure 3.1. The objective, in this case, is to find a transformation which can correct for differences between the two images, so that comparisons can be made for diagnostic purposes. Instead of using all of the data available in both images, a preprocessing step can be used to remove extraneous information and to extract a set of useful features. One approach might be to use a standard edge detection scheme as a preprocessing step, in which case image intensities corresponding to both edges and regions of highest contrast would define the feature space. Another possibility would be to select a set of 2-dimensional positions in each image which correspond to the same set of recognizable points in the region of interest. In this case, the features used in the matching process would be 2-D point coordinates.

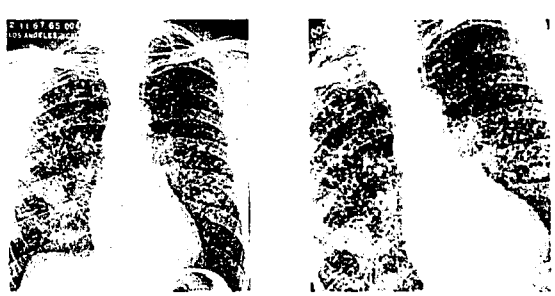


Figure 3.1: A pair of radiographic images from the same patient

A large number of matching algorithms use so-called point features as tokens for matching, since:

- point features can be obtained accurately and with relative ease,
- mathematical manipulations of points are both direct and simple,
- point-based techniques can be used over a broad range of applications.

Scanning devices that measure the 3-D coordinates of points in space have become both accurate and readily available, due to recent improvements in laser technology [1]. Consequently, point-based matching techniques have become increasingly attractive in a number of applications, including computer vision, remote sensing, and others. When the data to be matched involves two-dimensional imagery, point features can be relatively easy to identify and extract, either automatically or through human interaction. The automatic location of feature points is preferable when large amounts of data are to be matched. In such cases, the features are frequently dominant points along curves such as corners, line intersections, inflection points, points of maximum curvature, and points along discontinuities [8]. When a small set of points is sufficient for the matching application, landmarks can be manually selected which are known to be stationary and can be easily pin-pointed in both data sets. Another way to obtain feature points for matching, in image-related applications, is to place markers in the scene which can be used as reference points. For instance, in medical imaging, identifiable structures, called fiducial markers, are placed in known positions in the patients to act as reference points, or stereotactic frames are affixed to a patient's head to provide an identifiable three-dimensional coordinate reference frame [8].

An advantage of matching techniques which use points as tokens, instead of using more complex features, is their ability to handle patterns of arbitrary shape. Features having greater structural complexity, such as surfaces for example, often impose constraints which prohibit the analysis of arbitrarily shaped patterns. There are also several cases where matching is performed on data which does not describe physical objects (*e.g.* , multidimensional scaling of data in psychology [12]); other applications exist in which point-like data is readily available (*e.g.* , star-matching, in astronomy [11]). Consequently, point-based techniques can be useful over a wide variety of applications. On the other hand, matching on the basis of individual point comparisons, rather than using more complex features, can require a greater amount of computational effort, since generally more comparisons have to be made.

3.2.2 Geometric Transformations

Having chosen a feature space for matching, a mathematical transformation must be selected which is presumably capable of bringing the data set to be matched into alignment. In most cases, the transformation can be expressed as a matrix which describes the computations that are necessary for mapping point-coordinates in one data set onto point-coordinates in the other data set. When the mapping is geometric in nature (*i.e.* , angles, lengths, etc. are involved), it is called a *geometric transformation.*

A few important assumptions must be made when selecting an appropriate transformation for a particular matching problem. One of these assumptions involves the dimensionality of the data points that will be manipulated by the transformation. Most methods assume that the data to be matched lie in two- or three-dimensional space; however, matching problems can arise in any dimension. For instance, a 4-D problem arises when matching a time series of 3-D images [55]. Matching can also be performed on data sets of different dimensionality. For example, data lying in the n-dimensional Euclidean space, $\mathbb{E}^n$, may be matched with data in the m-dimensional Euclidean space, $\mathbb{E}^m$, where $m \neq n$ [2].

Another important consideration is the domain over which transformations in the search space are applicable to the data. Essentially, matching transformations can either be *global* or *local.* A global transformation has parameters which influence the transformation of a data set as a whole. In contrast, the parameters of a local transformation affect only certain *parts* of the data set; hence, the mapping effected by a local transformation may be considered as the union of several smaller maps which are applicable to different parts of the data. The difference between local and global transformations is illustrated in Figure 3.2. Local transformations can account for many types of variations between data sets in a computationally efficient manner vis-à-vis global ones. For instance, in image processing applications, local transformations may be necessary to describe position-dependent variations such as object deformations,

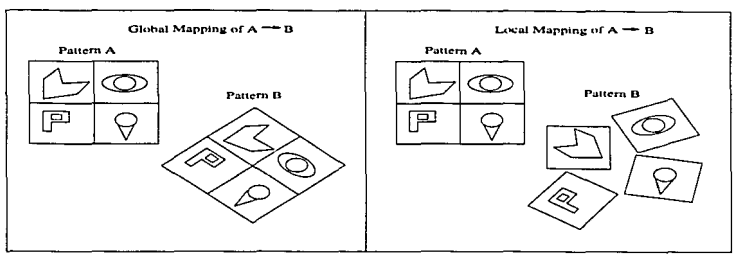


Figure 3.2: An example of 2D global and local transformations

perspective distortions of complex 3-D scenes, or nonlinear distortions due to sensors [8]. Global transformations are generally ineffective in such cases, since they assume that only one variation exists over the entire image and, therefore, preclude the possibility of independent local changes. Nevertheless, the use of a global transformation is applicable in many cases; hence, the increase in complexity involved in finding local mappings is often unjustified. Moreover, it is always possible to apply "global" techniques to local areas in the data, and then describe the overall mapping in a piecewise fashion. The point-matching approaches that are compared in this study search for global transformations.

The selection of a particular class of transformations which best suits the data to be matched is yet another important consideration. The most commonly used transformation classes in pattern matching are: rigid, affine, perspective, and curved. *Rigid transformations* preserve a number of mathematical relationships, such as the distances between points and the angles between lines, when mapping points in one data set onto points in another data set. Consequently, this class of transformations is useful for describing the motion of undeformable, or rigid, objects. *Affine transformations* are characterized by the property that straight lines are mapped onto straight lines, while preserving parallelism; therefore, this class of transformations can account for distortions such as scaling, which may or may not be uniform, and shearing. *Per-*

spective transformations appear mostly in image-related applications and can account for distortions that arise from the projection of objects at varying distances to the sensor onto the image plane [8]. *Curved transformations* map straight lines onto curves, and can account for any type of object distortion. Some important properties of the previous transformation classes will be discussed in the sequel.

Rigid transformations can be decomposed into a rotation through one or more angles, and a translation [37]. Accordingly, the mapping of an m-dimensional coordinate vector $\mathbf{p}_1$ onto an n-dimensional vector $\mathbf{p}_2$ by a rigid transformation can be expressed by the equation:

$$\mathbf{p}_2 = \Theta\mathbf{p}_1 + \mathbf{t}, \tag{3.1}$$

where Θ is an $(n \times m)$-dimensional rotation matrix, and $\mathbf{t}$ is an n-dimensional translation vector. In most matching schemes, $m = n$; nevertheless, the previous equation is expressed in general terms to include the possibility of matching data sets from different dimensions. Implicit in (3.1) is the constraint that the rotation matrix Θ must be orthogonal, when $m = n$. This mathematical property ensures that the lengths of all lines in the data set and the angles between them are preserved by the mapping. The $(n \times n)$-dimensional matrix Θ is orthogonal if

$$\Theta^T\Theta = \Theta\Theta^T = \mathbf{I}_n, \tag{3.2}$$

where $\mathbf{I}_n$ is the n-dimensional identity matrix. When $m \neq n$, the $(n \times m)$-dimensional matrix $\Theta\binom{n}{m}$ must satisfy the equation:

$$\Theta^T\binom{m}{n}\Theta\binom{n}{m} = \mathbf{I}_m, \tag{3.3}$$

where $\Theta^T\binom{m}{n}$ is $(m \times n)$-dimensional, and $\mathbf{I}_m$ is the m-dimensional identity matrix [2]. Since, in the previous case, $\Theta\binom{n}{m}$ is not a square matrix, only its columns are orthonormal to each other and, therefore, the product $\Theta^T\binom{n}{m}\Theta\binom{m}{n} \neq \mathbf{I}_n$; hence, the matrix which satisfies (3.3) is not an orthogonal transformation but is known instead as a *partial isometry matrix* [56]. In both cases, the transformation Θ is a proper rotation

matrix if and only if its determinant is $+1$; a transformation Θ having a determinant of -1 is called an improper rotation, or a *reflection* [39].

Orthogonal rotation matrices that satisfy (3.1) can be expressed in terms of the geometric quantities which define a rigid-body motion. For instance, when $m = n = 2$, the rotation of a two-dimensional rigid body is expressed in terms of a rotation angle, φ, as

$$\Theta_{2D} \doteq \begin{pmatrix} \cos\varphi & \pm\sin\varphi \\ \sin\varphi & \mp\cos\varphi \end{pmatrix}. \tag{3.4}$$

A two-dimensional rigid body rotation is illustrated in Figure 3.3. When $m = n = 3$,

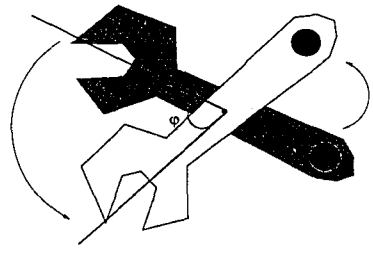


Figure 3.3: Two-dimensional rigid body rotation

many different representations are possible for the rotation matrix, including Euler angles, the Gibbs vector, Cayley-Klein parameters, Pauli-spin matrices, axis-and-angle systems, and Hamilton's quaternions [39]. In particular, Euler angles and axis-and-angle systems are often used in the literature on pattern matching. When Euler angles are used, the three-dimensional rotation of a rigid body is described as the result of rotating the body through three angles: θ_x, θ_y, and θ_z, about the x, y, and z axes, respectively. Equivalently, a rigid body that has undergone any sequence of rotations can be described by a single rotation of that body through an angle θ about an axis $\mathbf{n}$; hence, the axis-and-angle representation. If the previous representation is used, then

the 3D rotation matrix, Θ_{3D} is given by,

$$\Theta_{3D} \doteq \begin{pmatrix} n_x^2(1-c)+c & n_x n_y(1-c)-n_z s & n_x n_z(1-c)+n_y s \\ n_x n_y(1-c)+n_z s & n_y^2(1-c)+c & n_y n_z(1-c)-n_x s \\ n_x n_z(1-c)-n_y s & n_y n_z(1-c)+n_x s & n_z^2(1-c)+c \end{pmatrix}, \tag{3.5}$$

where $s = \sin\theta$, $c = \cos\theta$, and (n_x, n_y, n_z) is the directional cosine of the rotation axis, $\hat{\mathbf{n}}$, such that $n_x^2+n_y^2+n_z^2 = 1$. The axis-and-angle description of 3D rigid body rotation is illustrated in Figure 3.4.

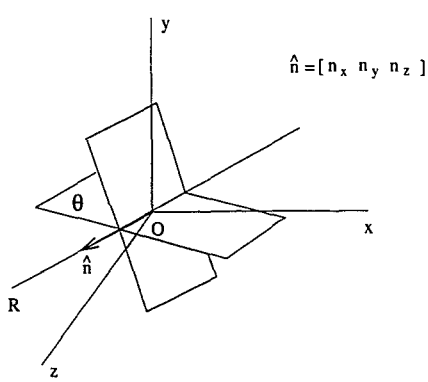


Figure 3.4: Axis-and-angle description of three-dimensional rigid body rotation

Affine transformations can be decomposed into a linear transformation and a translation. Accordingly, n-dimensional affine transformations can be expressed in a form similar to (3.1),

$$\mathbf{p}_2 = \mathbf{A}\mathbf{p}_1 + \mathbf{t}, \tag{3.6}$$

where $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{t}$ are the same as in (3.1) and $\mathbf{A}$ is a real-valued matrix. The main difference between affine and rigid transformations is that the matrix $\mathbf{A}$ of (3.6) is not necessarily orthogonal and, therefore, angles and lengths are no longer preserved by the mapping; however, parallel lines still remain parallel [8]. Affine transformations

are useful when the relationships between the points involve changes in aspect ratio, or other scale-related distortions.

Perspective transformations are typically used to account for the distortion which occurs when a 3D scene is projected onto a 2D image plane. This class of transformations maps straight lines in one data set onto straight lines in the other data set, but does not necessarily preserve any parallel relationships that may exist between the lines [57]. Projective transformations can be mathematically expressed as a linear transformation in a higher dimensional space. For instance, a 2D mapping of the point (x, y) onto the point (x', y') can be expressed as [57]:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} u/w \\ v/w \end{pmatrix}, \text{where } \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.7}$$

and w represents the extra so-called *homogeneous* coordinate.

Curved transformations can map a straight line onto a curve, and are generally expressed by a real-valued function in n variables, for n-D mappings. A well-known class of curved transformation functions are the transformations of polynomial type. For instance, a polynomial function in 2D can be expressed as [57]:

$$\begin{aligned} x' &= a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + \ldots \\ y' &= b_{00} + b_{10}x + b_{01}y + b_{20}x^2 + b_{11}xy + b_{02}y^2 + \ldots, \end{aligned} \tag{3.8}$$

where (x, y) and (x', y') are the 2D points before and after the transformation, respectively. Curved transformations can account for any type of (object) distortion and, therefore, represent the most general type of transformation described in this section; however, the nonlinear nature of the equations which describe the mapping of points between the data sets makes them difficult to solve.

3.2.3 Search Strategies

The search strategy of a matching technique dictates how the search for an optimal transformation is conducted. In many cases, the choice of a particular search strategy is largely influenced by the characteristics of the feature space. For example, the generalized Hough transform was developed with the specific intent of matching shapes from contours [8]. Also, data sets which have been stored as trees or relational graphs are efficiently matched by search strategies which are specifically developed to handle such data structures [44]. Other important considerations in selecting a search strategy are: how the strategy deals with missing information, whether the strategy can be implemented in parallel, what assumptions are made by the strategy, and what the typical storage and computational costs are.

Search strategies can be roughly classified into two categories: direct or search-oriented [57]. Direct methods compute the transformation parameters in a straight-forward manner, and are usually based on lists of corresponding points [57]. A shortcoming of direct methods is that simplifications of a matching problem are often needed to permit the use of straight-forward calculations in finding the required transformation [57]. Such simplifications usually either limit the usefulness of the technique or significantly reduce the accuracy of its result.

Search-oriented techniques typically start from one or more initial guesses, and use a similarity metric to guide the search for an optimal transformation. Consequently, the success of a search-based method is largely affected by the characteristics of the similarity metric which it employs. A key objective in the design of a search-based method is to construct a similarity metric, or rating function, that rates candidates in a reasonable way. Typically, the rating function needs to be evaluated very often; therefore, a comprise between the accuracy of the rating function and the computational speed of the algorithm is sometimes necessary [57]. The general behaviour of the rating function is another important consideration in search-based methods. If the

rating function has only one extremum, then simple and relatively fast strategies such as the method of steepest descent can be used to find the optimal transformation. In many cases, however, the rating function is not well behaved, in the sense that many local extrema exist, which usually calls for a compromise between the speed and the accuracy of the search strategy [57]. Also, search strategies that employ ill-behaved rating functions often lead to sub-optimal results [57].

3.2.4 Similarity Metrics

The similarity metric measures how closely two patterns are matched. Based on this metric, potential matches can be compared to determine which transformation from the search space minimizes the error in matching. Perfect matches (*i.e.* , solutions with zero error) cannot be obtained in real applications, since noises due to optics, scanners, or other factors, such as quantization and feature extraction, invariably corrupt the data. An important source of errors in range images is attributed to inaccuracies in the x, y and z coordinate measurements that are obtained using a variety of methods [58, 59]. These inaccuracies are typically modeled as additive Gaussian stochastic processes and are often assumed to exist largely in the z (*i.e.,* depth) coordinate [4]; however, the noise level inherent in each coordinate measurement is not equal, in practice, and depends on both the characteristics and the physical limitations of the sensors [58, 59]. Other errors, which may be introduced by the matching strategy itself, must also be taken into account. Consequently, matching techniques strive to obtain an *optimal* transformation, which yields the best match vis-à-vis the similarity metric.

A large number of pattern matching techniques use a particular instance of the so-called L_p-norm as the similarity metric, *viz.* , :

$$\left(\int_S |f(\mathbf{x}) - g(\mathbf{x})|^p d\mathbf{x}\right)^{1/p}, \tag{3.9}$$

where $\mathbf{x}$ is a vector in n-dimensional Euclidean space, $\mathbb{E}^n$, $f(\mathbf{x})$ and $g(\mathbf{x})$ are real-

valued functionals defined over the region $\mathcal{S} \subset \mathbb{E}^n$, and $1 \leq p < \infty$. A similar criterion exists for discrete-time applications, known as the ℓ_p-norm, where the integral in the expression above is replaced by an appropriate number of summations, and finite limits are imposed on $\mathcal{S}$. If the functionals $f(\mathbf{x})$ and $g(\mathbf{x})$ are interpreted as the reference pattern and the approximation to the reference pattern, respectively, then (3.9) can be re-written as:

$$\left(\int_{\mathcal{S}} |e(\mathbf{x})|^p d\mathbf{x}\right)^{1/p}, \tag{3.10}$$

where $e(\mathbf{x})$ is a real-valued functional defined over the region $\mathcal{S} \subset \mathbb{E}^n$, and which represents the error of the matching process. From the previous equation, it is seen that the L_p-norm decreases with increasing similarity between the two patterns being matched; moreover, the smallest value of the metric is attained when $f(\mathbf{x}) = g(\mathbf{x})$ or, equivalently, when the error $e(\mathbf{x}) = 0$.

Two commonly used instances of the L_p-norm in pattern matching are the L_1-norm and the L_2-norm. When the value of p in (3.9) is $p = 1$, the L_1-norm is given by:

$$\int_{\mathcal{S}} |f(\mathbf{x}) - g(\mathbf{x})|\, d\mathbf{x}, \tag{3.11}$$

which is also known as the sum of absolute differences. This metric has been shown to work well for the registration of images when the transformation involves only small rotations, translations, or scale changes [60]. Perhaps the most commonly used metric in pattern matching is the sum of squares of differences, also known as the *least squares* criterion:

$$\int_{\mathcal{S}} |f(\mathbf{x}) - g(\mathbf{x})|^2\, d\mathbf{x}, \tag{3.12}$$

which is the L_2-norm squared. A disadvantage of methods that are based on least-squares estimation is that they perform poorly in the presence of outliers (*e.g.* points severely corrupted by noise) or point correspondence mismatches [61]. In such instances, methods based on the so-called *least median of squares estimator* (LMedS) are better suited to solve the motion estimation problem [61, 62].

A similarity measure which is closely related to the L_2-norm is the cross-correlation metric. Cross-correlation is mostly used in image processing applications such as template matching or pattern recognition, where the location and orientation of a template or pattern in an image is to be determined. Many image registration techniques use cross-correlation both as a similarity metric and as a search strategy [63], [52], [64]. The goal of these techniques is to maximize the so-called *cross correlation function*, which can be expressed in two-dimensional real space as:

$$C(u,v) = \int_x \int_y f(x,y)g(x-u, y-v)dxdy, \tag{3.13}$$

where f is the template, g is the image, and the dimensions of the template are much smaller than those of the image. If the template and the image are exactly equal at a translation $(u,v) = (i,j)$, then the match is indicated by a peak in the cross-correlation at $C(i,j)$ [8]. Conventional cross correlation techniques generally have the desirable properties of being simple, flexible and amenable to digital, optical, and electro-optical implementation [54]. Moreover, the latter techniques are robust in the presence of random, or *white*, noise, which is usually an adequate way to represent the noise sources responsible for corrupting the data to be matched. Since the cross-correlation metric needs to computed for each allowable transformation and for each position in an image, however, the computational costs quickly become unmanageable as the number of possible transformations grows. Hence, the usefulness of conventional cross correlation methods is generally limited to low-dimensional matching problems ($n \leq 2$) which are further restricted to small rigid or affine transformations [8].

Other similarity measures exist, many of which are largely dependent on the properties of the data or on the specific nature of the search strategy. The so-called "fitness function" that is used in the genetic algorithm by Ansari *et al.* [45] is a good example of this. Also, in structural or syntactic techniques, where patterns are stored as hierarchical structures including trees and graphs, specialized metrics are devised, based on the properties of the data structures or on various relations of interest. A

few examples include: the distance between candidate graphs [48] and the change of entropy between "random" graphs [51]. Tree-search methods sometimes use the error accumulated in searching a specific path of the tree as the similarity metric (*e.g.* [1], [47]). The so-called *k-th order error* measure, which is used in the tree-search procedure proposed by Cheng and Don [1], is an example of the previous similarity metric. A detailed description of this error measure is given in Chapter 5.

3.3 Summary

In this chapter, four basic elements which can be identified in virtually all pattern matching techniques were presented; namely, the feature space, the geometric transformation, the search strategy, and the similarity metric. By selecting an appropriate feature space, the data which is used in the matching process can be significantly reduced. Commonly used feature spaces include: points, edges, surfaces, image intensities, and others. The choice of a geometric transformation determines in what way the data sets being matched can be brought into alignment. The transformation can either be global or local, depending on whether a single transformation is sufficient for aligning the data sets, or whether several transformations need to be applied to various local regions in the data sets, respectively. The transformation must also be selected from among the various existing classes of transformations, which include: rigid transformations, affine transformations, projective transformations, and curved transformations. Search strategies for obtaining the assumed transformation were discussed in a general manner. Two types of search strategies were discussed; namely, direct methods, and search-oriented methods. In the latter case, the similarity metric is intimately involved in the searching process. Some of the similarity metrics that were discussed include: the sum of absolute differences, the method of least squares, and cross-correlation. The previous metrics are commonly used in pattern matching. A few other similarity metrics, which are less frequently encountered in the literature, were also briefly discussed.

Chapter 4

An SVD-Based Algorithm

4.1 Introduction

The task of finding point correspondences, in matching problems for which the data sets are unordered, is known to be combinatorial in nature; consequently, the number of computations which are required to find an optimal solution to the point correspondence problem, can quickly become unmanageable. Brockett [3] has demonstrated, however, that this difficult combinatorial optimization problem can be cast into a differentiable, or "calculus-type" setting, which greatly reduces its inherent complexity. This reduction in complexity is achieved by formulating the matching problem using the theory of Lie groups and Lie algebras, making it possible to recast various types of point matching problems as function optimization problems. Morgera and Lie Chin Cheong [54] have taken Brockett's work one step further, by developing an algorithm for solving these function optimization problems. The algorithm is iterative, and uses the SVD as a primary tool for finding the optimal solution. In this chapter, the iterative point matching algorithm proposed by Morgera and Lie Chin Cheong [54, 2] is reviewed.

In Section 4.2, a mathematical description of various point matching problems is given. For convenience, the mathematical formulations in this section use the same notation as in [54, 2]; notwithstanding, the matching problems which are presented in this chapter are not specific to any application or technique. In Section 4.3, the background behind Morgera and Lie Chin Cheong's approach is briefly presented; specifically, the functionals that are involved in the optimization process are discussed, for a particular type of point matching problem. The algorithm for solving the previous problem is described in Section 4.4. Extensions of the theory, to include other point matching problems, are discussed in Section 4.5.

4.2 Problem Formulation

In this section, a formal mathematical description of various point-matching problems is addressed. Let the two point sets $\{\mathbf{x}_i : i = 1, 2, \ldots, p\} \in \mathbb{E}^n$ and $\{\mathbf{y}_i : i = 1, 2, \ldots, q\} \in \mathbb{E}^m$ represent the patterns to be matched, where $\mathbb{E}^n$ and $\mathbb{E}^m$ denote n- and m-dimensional Euclidean spaces, respectively. Using this general notation, a number of different point-matching problems can be defined, according to whether or not the point sets have the same cardinality (*i.e.* , equal number of points) and whether the point sets are from spaces having the same or different dimensionality. The cardinality of the point sets is denoted by the pair (p, q); whereas, the dimensionality of the data is denoted by the pair (m, n). In realistic applications, the number of points in the data sets are not equal (*i.e.* , $p \neq q$). This difference in cardinality between the point sets is often the result of occlusion. In the previous case, certain points may disappear, or others may arise because objects are either partially or completely hidden by the presence of other objects or shadows, or because certain features are rotated out of view [65]. When $p = q$, it is still possible to have points in both data sets which have no correspondences; hence, matching problems in which $p = q$ do not necessarily imply that a one-to-one correspondence exists between the points in the two patterns. In

most cases, the data sets contain points which have been measured in spaces having the same dimensionality (*i.e.* , $m = n$); however, some applications exist where point sets from different dimensions need to be matched.

Point-matching problems can also be defined as ordered or unordered, depending on whether or not the correspondences between the feature points in the two data sets are known. If the data sets are arranged such that $\mathbf{x}_1$ corresponds to $\mathbf{y}_1$, $\mathbf{x}_2$ corresponds to $\mathbf{y}_2$, and so on, then the point sets are said to be ordered. In general, the data is not arranged in this way and, therefore, the ordering of the points in the data sets should be considered arbitrary. An exception to this is when the features are extracted explicitly by the user, in which case the correspondences are known *a priori*; thus, whether or not ordered data can be assumed depends on the application. In the sequel, several of the previous point-matching problems will be formulated in mathematical terms.

4.2.1 Matching of Point Sets With the Same Cardinality

Case A: Ordered Data

Suppose the two patterns take the form of *ordered* n-tuples, $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$, and $\{\mathbf{y}_i : i = 1, 2, \ldots, p\}$, which will be called the the *sensed* data set and the *template* data set, respectively, from now on. Further, assume that both patterns lie in the n-dimensional Euclidean space, $\mathbb{E}^n$. If the geometric transformation is assumed to belong to the class of rigid transformations, then the points in the sensed data set and the template data set are related by a *noisy rigid motion* model, which is given by [54]:

$$\mathbf{x}_i = \phi(\mathbf{y}_i) + \mathbf{b} + \mathbf{n}_i \,, \tag{4.1}$$

where $\phi(\mathbf{y}_i)$ denotes the rotated version of the vector $\mathbf{y}_i$, $\mathbf{b}$ is an n-dimensional column vector representing a translation in $\mathbb{E}^n$, and the $\mathbf{n}_i$ are stochastic zero mean noise vectors which are mutually statistically independent. The operator ϕ is used to denote

rotations, in the context of Lie groups; specifically, $\phi \in \Phi$, a representation of the orthogonal Lie group [54, 2]. By the same token, we assume that ϕ acts linearly on a vector $\mathbf{x} \in \mathbb{E}^n$ such that $\phi(\mathbf{x}) \rightarrow \Theta^{\mathbf{T}}\mathbf{x}$, where Θ is an n-dimensional rotation matrix. The operator ϕ must also preserve lengths, such that

$$\|\phi(\mathbf{x}_i)\|^2 = \|\mathbf{x}_i\|^2 \,, \tag{4.2}$$

where the notation $\|\cdot\|$ denotes the Euclidean norm, and $\|\mathbf{x}_i\|^2 = \mathbf{x}_i^T\mathbf{x}_i$ is the square of the length of the vector $\mathbf{x}_i$[1]. Note that (4.2) is equivalent to the constraint: $\Theta\Theta^{\mathbf{T}} = \mathbf{I}_n$, which was discussed in Section 3.2.2.

If the least squares criterion is used as the similarity metric, then the objective of this matching problem is to minimize the function

$$\epsilon(\Theta, \mathbf{b}) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_i) - \mathbf{y}_i - \mathbf{b}\|^2 \,, \tag{4.3}$$

with respect to $(\Theta, \mathbf{b})$. The previous equation can be simplified, by virtue of the so-called *centroid coincidence theorem* [52], which states: if the least-squares solution to (4.3) is $(\Theta^*, \mathbf{b}^*)$, where Θ^* and $\mathbf{b}^*$ are the optimal rotation matrix and the optimal translation vector, respectively, then the two point sets $\{\mathbf{x}_i\}$ and $\{\mathbf{x}_i' \doteq \Theta^*\mathbf{y}_i + \mathbf{b}^*\}$ have the same centroid; that is, if the centroids of the patterns $\{\mathbf{x}_i\}$, $\{\mathbf{x}'_i\}$, and $\{\mathbf{y}_i\}$ are given by:

$$\bar{\mathbf{x}} = \frac{1}{p}\sum_{i=1}^{p}\mathbf{x}_i \,, \tag{4.4}$$

$$\bar{\mathbf{x}}' = \frac{1}{p}\sum_{i=1}^{p}\mathbf{x}_i' = \Theta^*\bar{\mathbf{y}} + \mathbf{b}^* \,, \tag{4.5}$$

$$\bar{\mathbf{y}} = \frac{1}{p}\sum_{i=1}^{p}\mathbf{y}_i \tag{4.6}$$

respectively, then $\bar{\mathbf{x}} = \bar{\mathbf{x}}'$. The previous theorem is useful in demonstrating that the translation vector $\mathbf{b}$ can be eliminated from (4.3) if, for each of the patterns to be

[1]The Euclidean norm of a vector $\mathbf{x} \in \mathbf{E}^n$ is defined as $\|x\| = \left(\sum_{k=1}^{n} \xi_k^2\right)^{1/2}$, where ξ_k are the elements of the vector $\mathbf{x}$ [39].

matched, the corresponding centroid is removed before applying the matching technique; specifically, let

$$\begin{aligned} \mathbf{x}_i'' &= \mathbf{x}_i - \bar{\mathbf{x}} \quad \text{and} \\ \mathbf{y}_i'' &= \mathbf{y}_i - \bar{\mathbf{y}} \,. \end{aligned} \tag{4.7}$$

Then,

$$\begin{aligned} \mathbf{x}_i - (\Theta \mathbf{y}_i + \mathbf{b}) &= \mathbf{x}_i'' + \bar{\mathbf{x}} - (\Theta \mathbf{y}_i'' + \Theta \bar{\mathbf{y}} + \mathbf{b}) \\ &= \mathbf{x}_i'' - \Theta \mathbf{y}_i'' \,, \end{aligned} \tag{4.8}$$

since $\bar{\mathbf{x}} = \bar{\mathbf{x}}' = \Theta \bar{\mathbf{y}} + \mathbf{b}$. Hence, (4.3) can be rewritten as,

$$\epsilon(\Theta) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_i) - \mathbf{y}_i\|^2 = \sum_{i=1}^{p} \|\Theta^{\mathbf{T}} \mathbf{x}_i - \mathbf{y}_i\|^2 \,, \tag{4.9}$$

where $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ represent the sensed data set and the template data set, respectively, with their corresponding centroids removed. Given the optimal rotation matrix Θ^* which minimizes (4.9), the optimal translation, $\mathbf{b}^*$, is given by:

$$\mathbf{b}^* = \bar{\mathbf{x}} - \Theta^* \bar{\mathbf{y}} \,, \tag{4.10}$$

where $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are the centroids of the patterns $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$, respectively.

Case B: Unordered Data

Now, suppose that $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$, and $\{\mathbf{y}_i : i = 1, 2, \ldots, p\}$, are *unordered*. In this case, the search space must be extended to include not only the set of all possible rigid transformations, but also the set of all permutations of the integers $\mathcal{Z}_p \doteq \{1, 2, \ldots, p\}$. To include permutations in the noisy rigid motion model, we first define the $(n \times p)$-dimensional matrices $\mathbf{X} = [\mathbf{x_1}\ \mathbf{x_2}\ \cdots\ \mathbf{x_p}]$ and $\mathbf{Y} = [\mathbf{y_1}\ \mathbf{y_2}\ \cdots\ \mathbf{y_p}]$. The shuffling, or reordering, of data points can then be represented by a p-dimensional permutation matrix $\mathbf{\Pi}$ which acts on the point set $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$ according to

$\{\mathbf{x}_{\pi(i)} : i = 1, 2, \ldots, p\} \rightarrow \mathbf{X\Pi}$. The matrix $\mathbf{\Pi}$ reorders the columns of $\mathbf{X}$ according to the correspondence which exists between the feature points of the two patterns. For the unordered problem, the noisy rigid motion model can be written as:

$$\mathbf{x}_{\pi(i)} = \phi(\mathbf{y}_i) + \mathbf{b} + \mathbf{n}_i \ , \tag{4.11}$$

where $\mathbf{x}_{\pi(i)}$ denotes the point in the sensed data set which corresponds to the ith point in the template data set. A simple example illustrating the relationships between the point sets $\{\mathbf{x}_i\}$, $\{\mathbf{x}_{\pi(i)}\}$, and $\{\mathbf{y}_i\}$ is given in Figure 4.1.

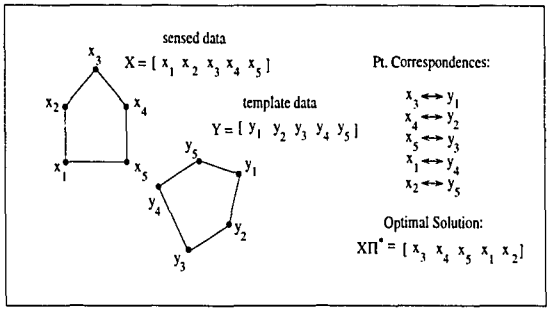


Figure 4.1: Example illustrating the point-correspondence problem

Since the shuffling operation does not affect the computation of the centroids, translations do not need to be considered in the optimization procedure, as in Case A; therefore, only Θ and $\mathbf{\Pi}$ need to be solved for. If the centroids of $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ are removed, as before, then the objective of the noisy point-pattern matching problem with permutations is to *jointly* determine the matrices Θ and $\mathbf{\Pi}$ such that the function

$$\epsilon(\Theta, \mathbf{\Pi}) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_{\pi(i)}) - \mathbf{y}_i\|^2 = \|\Theta^T \mathbf{X\Pi} - Y\|_F^2 \tag{4.12}$$

is minimized, where $\|\cdot\|_F$ denotes the Frobenius (Hilbert-Schmidt) norm[2]. As in case

[2]The Frobenius norm of a matrix $\mathbf{A}$ is defined as $\|A\|_F = \left(\sum_i \sum_j a_{ij}^2\right)^{1/2}$, where a_{ij} are the elements of the matrix $\mathbf{A}$ [39].

A, the optimal translation, $\mathbf{b}^*$ can be determined using the optimal rotation estimate Θ^* and the centroids of the patterns, according to (4.10).

4.2.2 Matching of Point Sets With Different Cardinality

In this problem, the two patterns to be matched take the form $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$ and $\{\mathbf{y}_i : i = 1, 2, \ldots, q\}$, where $p > q$. Specifically, we consider the case where all of the points in the template set have a correspondence with q out of the p points in the sensed data set, leaving $p - q$ points unmatched. This type of matching problem can arise, for example, when some of the points in the sensed data set are either missing or occluded in the template data set. Equivalently, the template data set may have more points than the sensed data set, in which case all of the sensed points have a correspondence in the template. An example of the former problem is illustrated in Figure 4.2.

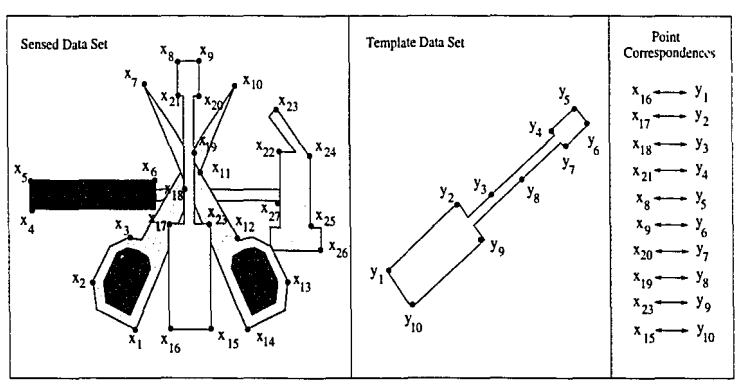


Figure 4.2: A point-matching example in which the data sets have different cardinalities

This matching problem can be formulated as in the previous subsection, for the general unordered case, with the following exception: instead of searching for the p-dimensional permutation matrix $\mathbf{\Pi}$ which is a permutation of the integers $\mathcal{Z}_p \doteq \{1, 2, \ldots, p\}$, we now look for the $(p \times q)$-dimensional matrix $\mathbf{\Pi}(_q^p)$ which is a permutation of q integers out of a possible p integers. To find the optimal transformation $\mathbf{\Theta}^*$, in this case, we must determine which of the q points in the sensed data set provide the best match between the patterns. It should be noted that, unlike the square $(p \times p)$-dimensional matrix $\mathbf{\Pi}$, where $\mathbf{\Pi}^T\mathbf{\Pi} = \mathbf{\Pi}\mathbf{\Pi}^T = \mathbf{I}_p$, the $(p \times q)$-dimensional matrix $\mathbf{\Pi}(_q^p)$ consists of $(p - q)$ row vectors which are zero; consequently, the product $\mathbf{\Pi}^T(_p^q)\mathbf{\Pi}(_q^p) = \mathbf{I}_q$, where $\mathbf{I}_q$ is the q-dimensional identity matrix, while the product $\mathbf{\Pi}(_q^p)\mathbf{\Pi}^T(_p^q)$ is a diagonal matrix with q ones and $(p-q)$ zeros as its diagonal elements.

Define the $(n \times p)$- and $(n \times q)$-dimensional matrices

$$\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_p]\,, \tag{4.13}$$

$$\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \ldots\ \mathbf{y}_q]\,, \tag{4.14}$$

where $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ are the sensed points and the template points, respectively, with their corresponding centroid removed [3]. Then, the function to be minimized with respect to $(\mathbf{\Theta}, \mathbf{\Pi}(_q^p))$ is:

$$\epsilon(\mathbf{\Theta}, \mathbf{\Pi}(_q^p)) = \|\mathbf{\Theta}^T\mathbf{X}\mathbf{\Pi}(_q^p) - \mathbf{Y}\|_F^2\,. \tag{4.15}$$

Once again, the optimal translation $\mathbf{b}^*$ can be determined using the optimal rotation estimate $\mathbf{\Theta}^*$ and the centroids of the patterns, according to (4.10).

4.2.3 Matching of Point Sets from Different Dimensions

In the previous subsections, the point patterns to be matched were always assumed to lie in the same Euclidean space $\mathbb{E}^n$. Now, suppose that the patterns $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$

[3]Note that the centroid of the template set now becomes $\bar{\mathbf{y}} = \frac{1}{q}\sum_{i=1}^{q}\mathbf{y}_i$

and $\{\mathbf{y}_i : i = 1, 2, \ldots, p\}$ consist of vectors in $\mathbb{E}^n$ and $\mathbb{E}^m$, respectively, where $n > m$. This problem is essentially the same as the one in the previous subsection, except that the dimensions of the matrix Θ are now changed from $(n \times n)$ to $(n \times m)$. To maintain the rigidity constraint, the rotation matrix $\Theta(^n_m)$ must satisfy the relation:

$$\Theta^{\mathbf{T}}(^m_n)\Theta(^n_m) = \mathbf{I}_m \,, \tag{4.16}$$

where $\mathbf{I}_m$ is the m-dimensional identity matrix. Under this new constraint, the functional $\epsilon(\Theta(^n_m), \mathbf{\Pi})$ to be minimized with respect to $(\Theta(^n_m), \mathbf{\Pi})$, for the general case of unordered data, is given by

$$\epsilon(\Theta(^n_m), \mathbf{\Pi}) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_{\pi(i)}) - \mathbf{y}_i\|^2 = \|\Theta^T(^m_n)\mathbf{X}\mathbf{\Pi} - \mathbf{Y}\|_F^2 \,, \tag{4.17}$$

where $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_p]$ is an $(n \times p)$-dimensional matrix, and $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \ldots\ \mathbf{y}_p]$ is an $(m \times p)$-dimensional matrix. When the data sets have a different number of points (*i.e.* $p \neq q$), (4.17) becomes:

$$\epsilon(\Theta(^n_m), \mathbf{\Pi}(^p_q)) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_{\pi(i)}) - \mathbf{y}_i\|^2 = \|\Theta^{\mathbf{T}}(^m_n)\mathbf{X}\mathbf{\Pi}(^p_q) - \mathbf{Y}\|_F^2 \,, \tag{4.18}$$

where $\mathbf{\Pi}(^p_q)$ is the $(p \times q)$-dimensional permutation matrix discussed in Section 4.2.2, and $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \ldots\ \mathbf{y}_q]$ is an $(m \times q)$-dimensional matrix.

4.3 Point-Matching As a Function Optimization Problem

In this section, the theory behind Morgera and Lie Chin Cheong's algorithm [2, 54] is briefly presented. The objective is to describe the basic approach, and to introduce the functionals which are involved in the optimization process, for a particular type of matching problem; namely, the point matching problem in which the data sets have the same cardinality (*c.f.* Section 4.2.1). The theory of Lie groups and Lie algebras

which, indeed, is an integral part of the approach, is beyond the scope of this discussion and will not be considered here. Interested readers are referred to [66] for a general treatment of the subject; also, the application of Lie groups and Lie algebras in point matching problems is discussed in detail in [2, 3, 54].

Consider, once again, the problem of matching two data sets having the same cardinality, and whose points are unordered. In this case, the objective is to minimize the least squares error function,

$$\epsilon(\Theta, \Pi) = \sum_{i=1}^{p} \|\phi(\mathbf{x}_{\pi(i)}) - \mathbf{y}_i\|^2 = \|\Theta^T \mathbf{X}\Pi - \mathbf{Y}\|_F^2, \tag{4.19}$$

with respect to (Θ, Π). One way to solve this problem, while reducing its complexity, is to assume that the optimization problems in Θ and Π are separable, and to solve for one of the transformation matrices, while keeping the other fixed. In this way, it is possible to derive functions of one variable only, which can be solved in an iterative manner; hence, the least squares problem defined by (4.19) is recast as a *function optimization* problem. The previous approach was first presented by Brockett [3], and was later adopted by Morgera and Lie Chin Cheong [2], who also provided a proof for the separability of the matching problem.

Suppose that the solution for Θ is sought first, while keeping Π constant[4]. This can be done by searching for the symmetric functions of the sensed data points that are independent of the permutation Π and which will map under ϕ to the same symmetric functions, when evaluated on the template points. For example, the centroid of the sensed data set, $\bar{\mathbf{x}} = \frac{1}{p}\sum_{i=1}^{p} \mathbf{x}_i$, maps under ϕ to the centroid of the template data set, $\bar{\mathbf{y}} = \frac{1}{p}\sum_{i=1}^{p} \mathbf{y}_i$. Similarly, the second moment of the sensed data set, $\mathbf{Q} = \frac{1}{p}\sum_{i=1}^{p} \mathbf{x}_i\mathbf{x}_i^T$, maps under ϕ onto the corresponding second moment for the template data set, $\mathbf{N} = \frac{1}{p}\sum_{i=1}^{p} \mathbf{y}_i\mathbf{y}_i^T$. In the method proposed by Morgera and Lie Chin Cheong, the previous symmetric functions are used in the matching procedure for finding Θ^*, the optimal

[4]The case where Π is found first, while Θ is kept fixed, is not discussed here, for the sake of brevity. Interested readers are referred to [54] or [2] for details.

rotation matrix. That is, the symmetric (second moment) functions are matched, such that the functional

$$\sigma = \|\phi(\mathbf{Q}) - \mathbf{N}\|_F^2 \tag{4.20}$$

is minimized.

Now, if $\phi \in \Phi$, a representation of the orthogonal Lie group, and if, further, ϕ acts linearly on vectors in $\mathbb{E}^n$, then the orthogonal group acts on the $(n \times n)$ symmetric matrix $\mathbf{Q}$ via the tensor product $\Theta \otimes \Theta$; that is [54]:

$$\phi(\mathbf{Q}) = \Theta^T \mathbf{Q} \Theta. \tag{4.21}$$

Using the previous fact, it can be shown that minimizing σ, in (4.20), is equivalent to maximizing the function

$$g(\Theta) = \mathrm{tr}(\Theta^T \mathbf{Q} \Theta \mathbf{N}), \tag{4.22}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of an n-dimensional matrix [5], and with Θ restricted to belong to the $(n \times n)$ orthogonal group of matrices $\mathcal{O}(n)$ [54]. Once the optimal rotation matrix Θ^* is found, the objective is to find the permutation matrix Π that will minimize $\epsilon(\Theta^*, \Pi)$, in (4.19). This can be achieved by maximizing the function

$$f(\Pi) = \mathrm{tr}(\mathbf{Z} \Pi \mathbf{Y}^T) \tag{4.23}$$

as shown in [54], where $\mathbf{Z} = \Theta^{*\mathbf{T}} \mathbf{X}$. Further, if the p-dimensional diagonal matrices $\mathbf{D}_{\mathbf{z}_i}$ and $\mathbf{D}_{\mathbf{y}_i}$ are formed from the ith rows of $\mathbf{Z}$ and $\mathbf{Y}$, respectively, then (4.23) can be rewritten as [54]:

$$f(\Pi) = \sum_{i=1}^{n} \mathrm{tr}(\Pi^T \mathbf{D}_{\mathbf{z}_i} \Pi \mathbf{D}_{\mathbf{y}_i}). \tag{4.24}$$

Since the permutation matrices are a subset of the orthogonal matrices [54], (4.24) can be seen as a special case of the functional $g(\Theta)$ (*c.f.* (4.22)).

To minimize the least squares error function $\epsilon(\Theta, \Pi)$, in (4.19), Morgera and Lie Chin Cheong have proposed the following three-step procedure [2]:

[5] The trace of an n-dimensional matrix $\mathbf{A} = [\xi_1, \xi_2, \ldots, \xi_n]$, where ξ_i, $i = 1, 2, \ldots, n$ are n-dimensional column vectors, is given by: $\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} \xi_i(\mathbf{i})$.

1. Find the orthogonal matrix $\widetilde{\Theta}^{\star} \in \mathcal{O}(n)$ which maximizes the function $g(\Theta) = \mathrm{tr}(\Theta^T \mathbf{Q} \Theta \mathbf{N})$, where $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$ and $\mathbf{N} = \mathbf{Y}\mathbf{Y}^T$.

2. Find the permutation matrix $\mathbf{\Pi}^{\star}$ which maximizes the function $f(\mathbf{\Pi}) = \mathrm{tr}(\tilde{\mathbf{Z}}^{\star} \mathbf{\Pi} \mathbf{Y}^T)$, where $\tilde{\mathbf{Z}}^{\star} = \widetilde{\Theta}^{\star T} \mathbf{X}$.

3. Find the exact transformation matrix $\Theta^{\star}$ by computing $\mathbf{M}^{-T}(\mathbf{M}^T\mathbf{M})^{1/2}$, where $\mathbf{M} = \mathbf{X}\mathbf{\Pi}^{\star}\mathbf{Y}^T$.

Note that, in Step 2, the function $f(\mathbf{\Pi})$ is maximized over the set of $(p \times p)$-dimensional permutation matrices $\mathbf{\Pi}$, which are actually a subset of the $(p \times p)$-dimensional orthogonal matrices $\Psi \in \mathcal{O}(p)$; consequently, the initial value of Θ that is obtained in Step 1 might not correspond exactly to the optimum transformation matrix for the solution of the permutation matrix $\mathbf{\Pi}^{\star}$. By considering the transformation matrices $\Theta \in \mathcal{O}(n)$ as a continuous set, or space, and the permutation matrices $\{\mathbf{\Pi}\}$ as a discrete set, there exists a *range* of rotation estimates $\Theta^{\star}$ that can yield the same optimal permutation matrix $\mathbf{\Pi}^{\star}$, when $f(\mathbf{\Pi})$ is maximized in Step 2 of the algorithm [2]. But, once $\mathbf{\Pi}^{\star}$ is determined, it can be shown that minimizing $\epsilon(\Theta, \mathbf{\Pi}^{\star})$ is equivalent to maximizing a function having the form $f(\Theta) = \mathrm{tr}(\mathbf{M}\Theta^T)$ [54]. The previous optimization problem turns out to have a unique solution $\Theta^{\star}$ which can be obtained from the closed-form expression [54]

$$\Theta^{\star} = \mathbf{M}^{-T}(\mathbf{M}^{\mathbf{T}}\mathbf{M})^{1/2}, \tag{4.25}$$

where $\mathbf{M} = \mathbf{X}\mathbf{\Pi}^{\star}\mathbf{Y}^T$, and the square root matrix $(\mathbf{M}^{\mathbf{T}}\mathbf{M})^{1/2}$ is taken to be positive definite; hence, $\Theta^{\star}$, in Step 3, is referred to as the *exact* transformation matrix. It should be noted that, when the data sets are ordered (*i.e.* $\mathbf{\Pi} = \mathbf{I}$), the optimum solution to the point-matching problem can be obtained via the closed-form expression of (4.25), where $\mathbf{M} = \mathbf{X}\mathbf{Y}^T$ [2].

4.4 The Algorithm

Given the functionals that were presented in Section 4.3, a number of well-known iterative optimization algorithms such as: the method of steepest descent/ascent, recursive least squares estimation, and others, can be used to devise an iterative matching procedure. In [54, 2], Morgera and Lie Chin Cheong demonstrate the utility of two optimization procedures, for implementing the three step procedure that was outlined in Section 4.3; namely, the method of steepest descent/ascent, and an SVD-based procedure which will be discussed shortly. The previous authors have also shown that a hybrid technique, composed of both the SVD-based algorithm and the steepest descent/ascent method could be used to carry out their three step matching procedure [2].

For the functionals involved in the three step matching procedure, the singular value decomposition (SVD) of a symmetric matrix having the form $\mathbf{M}^T\mathbf{M}$ can be used as the primary tool in a recursive optimization procedure. Recall, from Section 4.3, that the functional $f(\Theta) = tr(\mathbf{M}\Theta^T)$ which arises when the patterns are ordered (*i.e.*, $\Pi = \mathbf{I}$) reaches its optimum point when

$$\Theta = \mathbf{M}^{-T}(\mathbf{M}^T\mathbf{M})^{1/2} = \mathbf{M}(\mathbf{M}^T\mathbf{M})^{-1/2} \tag{4.26}$$

where $\mathbf{M} = \mathbf{X}\mathbf{Y}^T$ and $(\mathbf{M}^T\mathbf{M})^{1/2}$ is the real, symmetric positive definite square root matrix of $(\mathbf{M}^T\mathbf{M})$. The matrix $(\mathbf{M}^T\mathbf{M})^{-1/2}$ in (4.26) is readily computed by employing the SVD of the matrix $\mathbf{P} = \mathbf{M}^T\mathbf{M}$, which is given by

$$\mathbf{P} = \mathbf{U}\Sigma\mathbf{V}^T, \tag{4.27}$$

where Σ is a diagonal matrix and $\mathbf{U}$ and $\mathbf{V}$ are orthogonal. In (4.27), the diagonal elements of Σ are the singular values of $\mathbf{P}$, and the columns of $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of $\mathbf{P}$, respectively [67]. Using the decomposition of (4.27), it

is straightforward to show that[6]

$$(\mathbf{M}^T\mathbf{M})^{-1/2} = \mathbf{P}^{-1/2} = \mathbf{V}\boldsymbol{\Sigma}^{-1/2}\mathbf{V}^T. \tag{4.28}$$

The function $f(\boldsymbol{\Theta}) = \text{tr}(\mathbf{M}\boldsymbol{\Theta}^T)$ can have several local extrema. If the $(n \times n)$-dimensional matrix $\mathbf{M}$ is nonsingular, and the product $\mathbf{M}^T\mathbf{M}$ has n distinct positive eigenvalues $\{\lambda_i : i = 1, 2, \ldots, n\}$, it can be shown that there exist 2^n distinct real symmetric square root matrices $(\mathbf{M}^T\mathbf{M}^{1/2})$ [54]. From (4.26), it can be seen that 2^n equivalued maxima are, therefore, possible for $f(\boldsymbol{\Theta})$. However, only one maximum point exists for which the $(n \times n)$-dimensional real symmetric square root matrix $(\mathbf{M}^T\mathbf{M})^{1/2}$ is positive definite (*i.e.* , all of its eigenvalues are positive) and only one minimum point exists for which $(\mathbf{M}^T\mathbf{M})^{1/2}$ is negative definite (*i.e.* , all of its eigenvalues are negative) [54]. The remaining $2^n - 2$ stationary points are *saddle points* of $f(\boldsymbol{\Theta})$.

In the case of $g(\boldsymbol{\Theta}) = \text{tr}(\boldsymbol{\Theta}^T\mathbf{Q}\boldsymbol{\Theta}\mathbf{N})$, which needs to be maximized in Step 1 of the three step procedure, a closed form expression, such as the one in (4.26), cannot be derived to compute the optimal orthogonal matrix $\boldsymbol{\Theta}^*$. However, it can be shown that, at the extreme point for $g(\boldsymbol{\Theta})$, we have [54]:

$$\boldsymbol{\Theta} = (\mathbf{Q}\boldsymbol{\Theta}\mathbf{N})(\mathbf{N}\boldsymbol{\Theta}^T\mathbf{Q}^2\boldsymbol{\Theta}\mathbf{N})^{-1/2}, \tag{4.29}$$

where $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$, $\mathbf{N} = \mathbf{Y}\mathbf{Y}^T$, and the matrix $(\mathbf{Q}\boldsymbol{\Theta}\mathbf{N})$ is assumed to be nonsingular. If we define the matrix $\mathbf{R}(\boldsymbol{\Theta}) = \mathbf{Q}\boldsymbol{\Theta}\mathbf{N}$, then (4.29) can be rewritten as

$$\boldsymbol{\Theta} = \mathbf{R}(\boldsymbol{\Theta})[\mathbf{R}^T(\boldsymbol{\Theta})\mathbf{R}(\boldsymbol{\Theta})]^{-1/2}, \tag{4.30}$$

which is similar, in form, to (4.26). From (4.30), it is seen that the optimum orthogonal matrix $\boldsymbol{\Theta}^*$ which maximizes $g(\boldsymbol{\Theta})$ can be obtained by recursively computing $\boldsymbol{\Theta}$, according to (4.29), until it converges to the optimum point. Step 1 of the three step procedure in Section 4.3 can, therefore, be implemented by employing the following recursive procedure:

[6]Note that $\mathbf{U} = \mathbf{V}$ when $\mathbf{P}$ is both square and symmetric; this is the case when $\mathbf{P} = \mathbf{M}^T\mathbf{M}$

SVD-based Procedure for Maximizing $g(\Theta)$

1. Choose an initial point, $\Theta(0)$
2. For $k = 0, 1, 2, \ldots$, compute:

$$\mathbf{A}(k) = \mathbf{Q}\Theta(k)\mathbf{N} \tag{4.31}$$

$$\mathbf{B}(k) = [\mathbf{A}^T(k)\mathbf{A}(k)]^{-1/2} \tag{4.32}$$

$$\Theta(k+1) = \mathbf{A}(k)\mathbf{B}(k). \tag{4.33}$$

At each step of the recursive procedure, the SVD is employed to compute the inverse positive definite square root matrix of $\mathbf{A}^T(k)\mathbf{A}(k)$ in (4.32), as previously described.

As in most iterative algorithms, the choice of the initial condition $\Theta(0)$ has a strong influence on the convergence of the algorithm. In general, $\Theta(0)$ can be any orthogonal matrix; but, as discussed in [54], $\Theta(0)$ can be chosen such that the recursive algorithm converges to different local maxima. For example, when Θ is a (2×2)-dimensional orthogonal matrix, if $\Theta(0) = \mathbf{I}_2$ is chosen, where $\mathbf{I}_2$ is the two-dimensional identity matrix, then the algorithm should converge to the optimum orthogonal matrix Θ^* having a determinant of $+1$ (*i.e.* , the optimum *rotation* matrix). In fact, setting $\Theta(0)$ to any 2D matrix whose determinant is positive will yield the same result [54]. If, on the other hand, $\Theta(0) = \mathbf{J}$ is chosen, where

$$\mathbf{J} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad \mathbf{J} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{4.34}$$

then the algorithm should converge to the optimum *reflection* matrix, having a determinant of -1. The same result is obtained if $\Theta(0)$ is set to any other orthogonal matrix whose determinant is negative [54].

The permutation matrix Π which maximizes the function $f(\Pi)$ can be found using a similar recursive procedure, by simply replacing the matrices $\mathbf{Q}$ and $\mathbf{N}$ of (4.29)

by the appropriate diagonal matrices $\mathbf{D}_{\mathbf{z}_i}$ and $\mathbf{D}_{\mathbf{y}_i}$, respectively. Step 2 of the matching procedure can, therefore, be implemented using the following algorithm:

SVD-based Procedure for Maximizing $f(\mathbf{\Pi})$:

1. Choose an initial point, $\mathbf{\Pi}(0)$.

2. For $k = 0, 1, 2, \ldots$, compute:

$$\mathbf{A}(k) = \sum_{i=1}^{n} \mathbf{D}_{\mathbf{z}_i} \mathbf{\Pi}(k) \mathbf{D}_{\mathbf{y}_i} \tag{4.35}$$

$$\mathbf{B}(k) = [\mathbf{A}^T(k)\mathbf{A}(k)]^{-1/2} \tag{4.36}$$

$$\mathbf{\Theta}(k+1) = \mathbf{A}(k)\mathbf{B}(k). \tag{4.37}$$

The previous algorithm converges to the orthogonal matrix $\mathbf{\Pi}^* \mathbf{D}_{\mathbf{I}_p}$, where $\mathbf{\Pi}^*$ is a permutation matrix and $\mathbf{D}_{\mathbf{I}_p}$ is the diagonal square root of the p-dimensional identity matrix, $\mathbf{I}_p$ [54]; that is, $\mathbf{D}_{\mathbf{I}_p} = \text{diag}(\alpha_1, \alpha_2, \ldots, \alpha_p)$, where $\alpha_i = \pm 1$, $i = i, 2, \ldots, p$. Since there are 2^p distinct diagonal square roots of $\mathbf{I}_p$, and $p!$ permutations are possible, the number of stationary points for $f(\mathbf{\Pi})$ is $2^p p!$, of which 2^p are equivalued local minima, and 2^p are equivalued local maxima [54]. Following the same line of reasoning, since the optimum point of $g(\mathbf{\Theta})$ can be shown to involve a permutation of $\mathbf{D}_{\mathbf{I}_n}$, the diagonal square root of the n-dimensional identity matrix $\mathbf{I}_n$, the total number of stationary points for $g(\mathbf{\Theta})$ is $2^n n!$, of which 2^n are equivalued local minima, and 2^n are equivalued local maxima [54]. To avoid the stationary points of $f(\mathbf{\Pi})$, the initial condition $\mathbf{\Pi}(0)$ should not be set equal to the identity matrix $\mathbf{I}_p$, or any other matrix of the form $\mathbf{\Pi}\mathbf{D}_{\mathbf{I}_p}$. Aside from the previous restriction, any $(p \times p)$-dimensional orthogonal matrix can be used as the initial condition.

Having completed Steps 1 and 2 of the matching procedure, the exact transformation matrix $\mathbf{\Theta}^*$ can be computed, according to (4.26). It should be noted that the

matrix product $\mathbf{M}^T\mathbf{M}$ of (4.26) is not required to be nonsingular, since the SVD procedure for computing the inverse symmetric square root matrix can still be carried out in pseudoinverse form. Specifically, the pseudoinverse or the Moore-Penrose generalized inverse of a matrix $\mathbf{A}$ is defined as [68]

$$\mathbf{A}^{\#} = \mathbf{U}\begin{bmatrix} \mathbf{D}^{-1} & 0 \\ 0 & 0 \end{bmatrix}\mathbf{V}^T, \tag{4.38}$$

where $\mathbf{D}^{-1} = \text{diag}(\lambda_1^{-1}, \lambda_2^{-1}, \ldots, \lambda_w^{-1})$, such that $(\lambda_1, \lambda_2, \ldots, \lambda_w)$ are the eigenvalues of the matrix $\mathbf{A}$, and w is the rank of $\mathbf{A}$. The matrices $\mathbf{U}$ and $\mathbf{V}$ are two orthogonal matrices such that

$$\mathbf{V}^T\mathbf{A}\mathbf{U} = \begin{bmatrix} \mathbf{D} & 0 \\ 0 & 0 \end{bmatrix}, \tag{4.39}$$

where $\mathbf{D}$ is a $(w \times w)$-dimensional diagonal matrix, and $\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_w)$, which contains the non-zero eigenvalues of $\mathbf{A}$. The pseudoinverse form of the SVD, in (4.39), can also be used in the recursive algorithms for maximizing the functions $g(\Theta)$ and $f(\Pi)$ which were previously described, in the event that the matrix products $\mathbf{Q}\Theta(k)\mathbf{N}$ or $\sum_{i=1}^{n} \mathbf{D}_{\mathbf{z}_i}\Pi(k)\mathbf{D}_{\mathbf{y}_i}$, respectively, are nonsingular. It is to be noted, however, that when the pseudoinverse is required, the recursive algorithms previously described will not converge to orthogonal matrices, in general; moreover, the points of convergence for the algorithms are highly dependent on the initial guesses, $\Theta(0)$ and $\Pi(0)$ [54].

4.5 Extensions of the Theory

As discussed in [54, 2], the theory behind the previous approach for point matching can be extended to include other matching problems. Specifically, Morgera and Lie Chin Cheong have shown that their matching procedure could be extended to include patterns from Euclidean spaces of different dimensions (*i.e.* , $m \neq n$), and also to handle patterns where the number of feature points are not the same (*i.e.* , $p \neq q$). This subsection deals with the latter case only, for the sake of brevity, and also since it is

more commonly encountered in practice than the former problem. Interested readers are referred to [2, 54] for details regarding the matching of patterns from different dimensions, under the previous framework.

Consider, once again, the matching problem in which the patterns have different cardinalities *i.e.*, the patterns to be matched have the form: $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$, and $\{\mathbf{y}_i : i = 1, 2, \ldots, q\}$, where $p > q$. In this case, the function to be minimized with respect to $(\Theta, \mathbf{\Pi}(_q^p))$ is

$$\epsilon(\Theta, \mathbf{\Pi}(_q^p)) = \|\Theta^T \mathbf{X} \mathbf{\Pi}(_q^p) - \mathbf{Y}\|_F^2 , \tag{4.40}$$

where $\mathbf{\Pi}(_q^p)$ is a $(p \times q)$-dimensional permutation matrix. It can be shown that the minimization of (4.40) is equivalent to the minimization of the function

$$h(\Theta, \mathbf{\Pi}(_q^p)) = tr[\mathbf{\Pi}^T(_p^q)\mathbf{D}_Q\mathbf{\Pi}(_q^p)] - 2tr[\Theta^T\mathbf{X}\mathbf{\Pi}(_q^p)\mathbf{Y}^T], \tag{4.41}$$

in the case $p > q$[7], where $\mathbf{D}_Q = \text{diag}(\mathbf{X}^T\mathbf{X})$ is a p-dimensional matrix whose elements are the diagonal elements of the symmetric matrix $\mathbf{X}^T\mathbf{X}$ [54].

As discussed in [54], the previous minimization problem is *not* separable, but can still be implemented using a procedure similar to the one described in Section 4.3:

1. Determine the orthogonal matrix $\widetilde{\Theta}^* \in \mathcal{O}(n)$ which maximizes the function $g(\Theta) = \text{tr}(\Theta^T\mathbf{Q}\Theta\mathbf{N})$, where $\mathbf{Q} = \frac{1}{p}\mathbf{X}\mathbf{X}^T$ and $\mathbf{N} = \frac{1}{p}\mathbf{Y}\mathbf{Y}^T$.

2. Determine the permutation matrix $\mathbf{\Pi}^*(_q^p)$ which minimizes the function $h(\mathbf{\Pi}(_q^p)) = \text{tr}[\mathbf{\Pi}^T(_p^q)\mathbf{D}_Q\mathbf{\Pi}(_q^p)] - 2\text{tr}[\widetilde{\mathbf{Z}}^*\mathbf{\Pi}(_q^p)\mathbf{Y}^T]$, where $\widetilde{\mathbf{Z}}^* = \widetilde{\Theta}^{*T}\mathbf{X}$.

3. Determine the exact transformation matrix $\Theta^* \in \mathcal{O}(n)$ which maximizes the function $f(\Theta) = \text{tr}(\mathbf{M}\Theta^T)$, where the matrix $\mathbf{M} = \mathbf{X}\mathbf{\Pi}(_q^p)\mathbf{Y}^T$.

In Step 1, an approximation to Θ^* is obtained by mapping the normalized second moment of the sensed pattern onto the corresponding normalized moment of

[7] A similar function is minimized in the case $p < q$ [54]

the template pattern. The approximate value $\widetilde{\Theta}^*$ is then used in Step 2 to minimize the function $h(\Theta, \Pi(_q^p))$ and, hence, obtain the optimum permutation matrix $\Pi^*(_q^p)$. Given the previous permutation estimate, the exact rotation estimate Θ^* is found by substituting $\Pi^*(_q^p)$ back into (4.40) and minimizing the function $\epsilon(\Theta, \Pi^*(_q^p))$ with respect to Θ. Since this matching problem is not separable, the previous minimization is only approximately equivalent to the maximization of the function $f(\Theta)$. In some cases, and especially when $\widetilde{\Theta}^*$ and Θ^* are not close, $\Pi^*(_q^p)$ may not be the same for both of the two previous orthogonal matrices. When this happens, Steps 2 and 3 must be recursively repeated until the optimum transformation matrix Θ^* and the optimum permutation matrix $\Pi^*(_q^p)$ are found.

Since the function $g(\Theta)$ to be optimized in Step 1 is the same as in the case for which $p = q$, the recursive SVD-based algorithm for minimizing $g(\Theta)$ that was presented in Section 4.4 can also be used here. Step 2 is handled by noticing that the second term of (4.41) is essentially the function $f(\Pi(_q^p))$; hence the expression for $h(\Pi(_q^p))$ can be rewritten as:

$$h(\Pi(_q^p)) = tr[\Pi^T(_p^q)\mathbf{D}_Q\Pi(_q^p)] - 2\sum_{i=1}^{n} tr[\Pi^T(_p^q)\mathbf{D}_{\widetilde{Z}_i^*}\Pi(_q^p)\mathbf{D}_{y_i}], \tag{4.42}$$

where, in this case, $\mathbf{D}_{\widetilde{Z}_i^*}$ and $\mathbf{D}_{y_i}$ are $(p \times p)$- and $(q \times q)$-dimensional diagonal matrices, respectively. The function $h(\Pi(_q^p))$ can, therefore, be minimized using the SVD-based procedure for minimizing $f(\Pi)$ that was described in Section 4.4, with the following changes. First, the expression for $\mathbf{A}(k)$ in (4.35) is replaced by

$$\mathbf{A}_k(_q^p) = \mathbf{D}_Q\Pi_k(_q^p) - 2\sum_{i=1}^{n} \mathbf{D}_{\widetilde{Z}_i^*}\Pi_k(_q^p)\mathbf{D}_{y_i}, \tag{4.43}$$

and, second, the *negative* inverse symmetric square root matrix $[\mathbf{A}_k^T(_p^q)\mathbf{A}_k(_q^p)]^{-1/2}$ must be chosen, for $h(\Pi(_q^p))$ to converge to its minimum point. As for Step 3, the solution is directly computed from (4.26), where (now) $\mathbf{M} = \mathbf{X}\Pi^*(_q^p)\mathbf{Y}^T$, and the square root is the inverse symmetric positive definite square root matrix.

4.6 Summary

Various types of point-matching problems can be defined, as shown in this chapter. In specific cases, the theory of Lie groups and Lie algebras can be used to derive an equivalent function optimization problem; accordingly, the optimal solution can be found by minimizing (or maximizing) an appropriate functional, in the least-squares sense. For the general unordered point matching problem, the solution is obtained by jointly optimizing two functionals: one in terms of Θ, an orthogonal rotation matrix, and the other in terms of Π, a permutation matrix. This can be accomplished by following a three-step procedure. In the first step, an initial estimate of Θ (or Π) is obtained; this initial estimate is used in the second step, to obtain the optimal estimate for Π (or Θ); the third step is used to refine the initial estimate obtained in step 1.

Many different optimization algorithms exist, which can be used to implement the previous three step procedure. In [2, 54], the method of steepest ascent/descent is used to solve step 1 of the the three-step procedure, while step 2 is solved using an SVD-based algorithm and vice-versa. The two previous iterative algorithms can also be used alone, to solve the first two steps of the matching procedure. In this chapter, the focus is placed on the SVD-based iterative algorithm. The convergence properties of the previous algorithm are such that the choice of the initial condition has a strong influence on the convergence of the algorithm. Many local extrema can exist, which minimize the error in matching, and the final solution to which the algorithm converges is somewhat dependent on the choice of the initial guess; however, this guess does not need to be close to the true solution for the algorithm to converge.

Chapter 5

A Graph Matching Algorithm

5.1 Introduction

In this chapter, a second approach for solving the point correspondence problem is reviewed. This approach is fundamentally different from the one that was discussed in Chapter 4. The matching technique proposed by Cheng and Don [1], which is discussed in this chapter, is a graph-theoretic approach for solving the point correspondence problem. The method is based on concepts taken from graph theory, and uses both the point features and additional structural information, such as the interrelationships between features, for matching purposes. To find the optimal solution to the correspondence problem, a tree-based search strategy is employed, which is guided by a specially designed similarity metric.

The context behind this graph-matching approach is established in Section 5.2.1. Some basic concepts from graph theory are presented, and these ideas are used to formulate the point correspondence problems that were described in Chapter 4 (*c.f.* Section 4.2) as graph-matching problems. A number of definitions related to trees, which are derived from the previous graph theoretic concepts, are given in Section 5.2.2.

These definitions are then used to describe two widely known tree-based search techniques in Section 5.2.3. Having provided the relevant background, the graph matching approach proposed by Cheng and Don [1] is reviewed in Section 5.3.

5.2 Graph Matching

Graph-based matching techniques are commonly used in applications for which relevant information is available for matching purposes in addition to the selected features. This information can often be employed in the search strategy of a matching technique and can reduce the computational requirements for finding the optimal solution to the correspondence problem. In a graph-based approach, each of the patterns to be matched is considered as a graph, composed of both the features any other information which is used in the matching process. The solution to the correspondence problem is found by comparing the graphs, which is commonly achieved by employing a tree-based search procedure. The fundamental concepts related to graphs, trees, and tree-based procedures are discussed in this section.

5.2.1 Graphs

Some relevant definitions associated with graphs can be summarized as follows [69, 70]:

- A *graph* G is generally defined as a pair $G = (V, E)$, where $V = \{V_1, V_2, \ldots, V_n\}$ is a set of points, called *vertices*, and $E = \{E_1, E_2, \ldots, E_m\}$, $m \neq n$, is a set of lines, called *edges*, which join together certain pairs of vertices.
- Two vertices are called *adjacent* is there is an edge joining them.
- A *path* of length s is a sequence of vertices $(V_0, V_1, V_2, \ldots, V_s)$ such that V_k is adjacent to V_{k+1}, for $0 \leq k < s$.

- A graph is *connected* if there is a path between any two vertices of the graph.
- A *cycle* is a path of length three or more from a vertex to itself.

The previous definitions are illustrated in Figure 5.1, which shows a connected graph with six vertices and nine edges. In the figure, the vertex B is adjacent to vertices

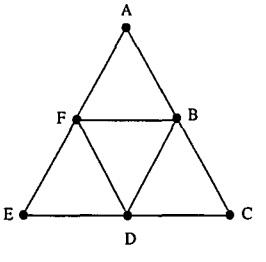


Figure 5.1: A graph

A, C, D, and F, but not to vertex E; there are two paths of length two from B to D, namely (B, C, D) and (B, F, D). Also, several cycles can be identified, including (A, F, B, A).

In general, graphs can be used to model a wide variety of situations. For example, the graph in Figure 5.1 could be used to represent a simple telecommunications network, in which case the vertices of the graph indicate telephones, switching centers, computers, or other communications devices, and the edges of the graph denote physical links, such as telephone lines. The same graph of Figure 5.1 could be used to represent other situations in which sets of elements are connected together by physical links, like electrical networks, road maps, oil pipelines, and subway systems, for example. In the context of pattern matching, the graph in Figure 5.1 may be used in the so-called structural description of an object; this type of description is commonly used in object recognition, to store complex models of objects in a database. In this case,

the vertices in the figure could represent the features which have been extracted for matching purposes, and the edges could represent the interrelationships between the features; for instance, the edge joining vertices A and B, in Figure 5.1, might indicate the location of feature A relative to feature B (*e.g.* , A is above B).

When a point-matching problem is formulated using graphs, the objective is to find either a *graph isomorphism* or a *subgraph isomorphism*. The two previous graph matching problems can be described through the following example. Let T and S be two graphs, representing the template data set and the sensed data set, respectively. In a graph matching procedure, T and S are compared vertex-by-vertex to determine any correspondences which may exist between the two graphs. If a one-to-one correspondence can be established between the vertices of T and S such that pairs of adjacent vertices are preserved, then the two graphs are said to be *isomorphic* [70]. An example of two graphs which are isomorphic is shown in Figure 5.2. Note

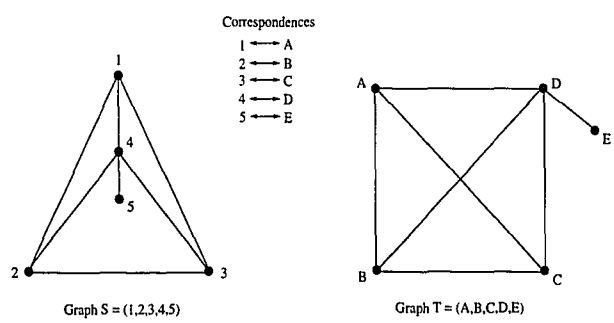


Figure 5.2: A Graph Isomorphism

that the exact positions of the vertices do not matter when drawing a graph; only the relationships between the vertices are important. For this reason, two graphs do not have to look the same to be isomorphic, as shown in Figure 5.2. The problem of finding a graph isomorphism is analogous to the point matching problem in which both

point sets have the same cardinality; moreover, no extra or missing points are allowed in either of the two point sets being matched. In other words, a one-to-one mapping must exist between every point feature in the data sets that are being matched. When one of the patterns involved in the matching process has some elements which are missing with respect to the other pattern to be matched, it may still be possible to find a subgraph isomorphism, in which one-to-one correspondences exist between only a subset of the vertices in S and T. The problem of finding a subgraph isomorphism is analogous to the point matching problem in which the data sets being matched have different cardinalities (*c.f.* Section 4.2.2). An example of a subgraph isomorphism is shown in Figure 5.3.

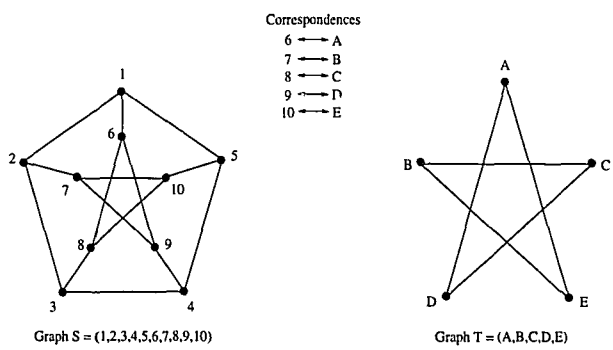


Figure 5.3: A Subgraph Isomorphism

5.2.2 Trees

One way to search for an isomorphism between two graphs is through a brute-force enumeration of all possible solutions to the matching problem. This exhaustive strategy is actually used to solve a number of difficult problems in graph theory, including graph

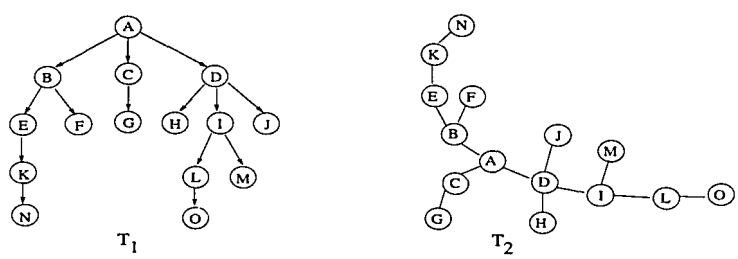


Figure 5.4: A Few Examples of Trees

isomorphism. In most cases, the enumeration process is carried out using a special type of graph, called a *tree*. Specifically, a tree is a graph with a designated vertex called a *root* such that a unique path exists from the root to any other vertex in the tree [70]. From the previous definition, it can easily be deduced that trees are connected graphs that have no cycles [70]. Two examples of trees are illustrated in Figure 5.4.

Referring to Figure 5.4, a few basic concepts regarding trees can be discussed. Note that the trees T_1 and T_2 in Figure 5.4 are almost identical. The only difference between the two trees, apart from the way they are drawn, is that T_1 is a *directed graph*, while T_2 is an undirected one. Until now, only undirected graphs have been discussed. The major difference between the two types of graphs is that the edges in a directed graph are ordered pairs of vertices of the form (V_i, V_j), $i \neq j$, to indicate that the two endpoints V_i and V_j are joined by a *directed edge*, or an *arc*, from V_i to V_j. Directed edges are drawn with arrows, as shown in the tree T_1. If a tree is a directed graph, it is sometimes called a *rooted tree*, to emphasize the fact that it has a unique root. On the other hand, a tree which is an undirected graph can have any vertex as the root; this can easily be seen in the tree T_2. In general, a tree can be viewed as a collection of smaller *subtrees* which are somehow connected together. From the previous perspective, it follows that every vertex of a tree is the root of some subtree. For example, the root A, in T_1, has three subtrees: $\{B, E, F, K, N\}$, in which vertex B

is the root; $\{C, G\}$, in which vertex C is the root; and $\{D, H, I, J, L, M, O\}$, in which vertex D is the root. The same could be said of the tree T_2, if the vertex A is designated as the root of the entire tree.

The standard terminology which is used to describe trees consists largely of genealogical words which are taken from the terminology of family trees [69]. In general, the vertices of a tree are discussed with respect to some type of parent-child relationship. For instance, each vertex of a tree is the *father* of the roots of its subtrees, and the latter vertices are called *brothers* or *siblings*. Similarly, the terms *ancestor* and *descendant* are used to describe a relationship that may span several levels of the tree. Referring to Figure 5.4, vertex B is the father of vertices E and F, and is also an ancestor of K and N; at the same time, B is the child of A and is also the brother of C and D. When a vertex of a tree has no descendants, it is called a *leaf* or a *terminal node*; hence, in Figure 5.4, the vertices $\{F, G, H, J, N, O\}$ in trees T_1 and T_2 are leaves.

5.2.3 Tree-Based Search Procedures

Having introduced some of the relevant terminology and a few basic concepts regarding trees, the relationship between trees and the "brute force" enumeration strategy that was mentioned earlier can now be discussed. In searching for an isomorphism between two graphs, the task of pairing a vertex in the template graph with some vertex in the sensed graph is equivalent to the general problem of selecting one out of a (finite) number of possible choices, according to some selection criteria. Rooted trees provide a natural framework for solving this type of problem. One of the properties of a rooted tree which makes it particularly useful for this type of application is its structure. Inherent in the structure of a rooted tree is a way to organize the possible choices to be made such that a systematic search though all the paths contained in the tree yields a complete enumeration of all possible solutions. In the sequel, two tree-based approaches for enumerating the potential solutions of a problem are briefly described.

One approach for exploring the different paths of a tree is to use a method called *depth-first search*, or *backtracking*. The objective of this method is to trace the deepest possible path in the tree to some leaf, starting from the root. If this path is not the solution, or if all possible solutions are needed, the search is continued by backtracking up the same path to the parent of the leaf, which represents the previous choice, and by then following a different edge in building a path to a new leaf. After all the edges leaving this parent vertex have been tried, the search backtracks up one level higher, and the procedure continues in the same manner. Eventually, the paths from the root to all the leaves of the tree are explored by this method; hence, a complete enumeration of all possible alternative sequences is achieved. If, however, only one solution is required, the method can be terminated as soon as a path is found which is a solution. An example which illustrates the progression of the depth-first search procedure is shown in Figure 5.5.

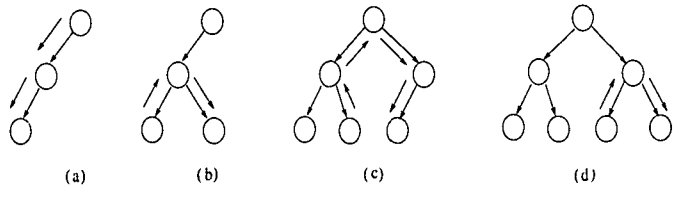


Figure 5.5: Exploring A Tree Using Depth-First Search: The search progresses from (a)-(d); arrows indicate the direction of the traversal.

Another way to explore a tree is to use the so-called *breadth-first search* method. In this procedure, the search begins by determining all the edges leaving the root. Then, all the edges leaving each of the root's children are determined, and so on; hence, the search is conducted by fanning out uniformly through the tree, starting from the root. An example which illustrates the progression of the breadth-first search procedure is shown in Figure 5.6. For problems in which only one solution is required, it may be advantageous to search for the deepest path in the tree, rather than building a large

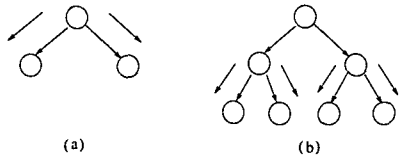


Figure 5.6: Exploring A Tree Using Breadth-First Search: The search progresses from (a)-(b); arrows indicate the direction of the traversal.

number of partial paths, only one of which is potentially useful. On the other hand, for problems in which the shortest path is required, or if long dead-end paths can be encountered while the solution paths tend to be relatively short, the breadth-first search may be preferable over backtracking [70].

Clearly, the previous enumeration procedures are meant to be thorough, rather than efficient, in finding a solution to a problem. If the solution to a problem involves a sequence of choices, then a brute-force investigation of every possible combination of alternatives quickly becomes unwieldy as the number of choices increases. For instance, in the point-matching problem, if the graphs representing the patterns to be matched have p and q vertices, respectively, where $p \geq q$, then the total number of potential pairings between the vertices of the two graphs is [47]:

$$p(p-1)(p-2)\ldots(p-q+1) \ . \tag{5.1}$$

When $q \approx p$, (5.1) shows that the total number of potential solutions to the matching problem is of the order $\mathcal{O}(p!)$. Consequently, as the number of features to be matched increases, the brute-force enumeration approach quickly becomes unmanageable. For this reason, a major challenge in devising tree search strategies, which tend to be exhaustive, lies in eliminating as many of the search paths in the tree as possible without actually traversing them. Specifically, if a vertex can be shown not to lead to a desirable solution, then that vertex, and the entire subtree below it, can be ignored, or *pruned*, in the search process. Pruning can drastically reduce the number of tests involved in the enumeration process and effectively determines the speed with which

a tree search method can find the optimal solution to a problem. When tree search procedures are applied to point matching problems, pruning is typically achieved by applying geometric constraints such as the distance between two points, or the angle between two lines, when potential pairings are tested. These geometric constraints are typically derived from the assumption that is made about the geometric transformation which is presumed to be capable of aligning the data sets (*c.f.* Section 3.2.2). A few examples of techniques which use geometric constraints to prune the search tree are found in [43, 44, 47, 1, 48]; notably, Grimson and Lozano-Perez [43] show that simple geometric constraints can be extremely powerful in reducing the number of potential search paths.

5.3 Graph Matching Algorithm

The concepts related to graphs, trees, and tree-searching, which were discussed in the previous section, are an integral part of the graph matching algorithm proposed by Cheng and Don [1]. In this section, the algorithm is reviewed. The basic strategy, which applies to the case of unordered data sets with the same cardinality, is described in Section 5.3.1. Extensions to include the possibility of missing or extraneous points in one or both of the patterns to be matched are discussed in Section 5.3.2. A data splitting strategy, which can be used to speed up the matching process, is presented in Section 5.3.3.

5.3.1 Basic Strategy

Recall the problem of matching two unordered point sets with the same cardinality, which was discussed in Section 4.2.2. This problem can be viewed as a graph matching problem where $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$ and $\{\mathbf{y}_i : i = 1, 2, \ldots, p\}$ represent the two point

sets to be matched. Further, define a graph $T = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_p)$, whose vertices are the position vectors in the template set $\{\mathbf{y}_i\}$. The objective of the graph matching problem is to construct a graph $\hat{T} = (\mathbf{x}_{\pi(1)}, \mathbf{x}_{\pi(2)}, \ldots, \mathbf{x}_{\pi(p)})$, from the points in the sensed data set $\{\mathbf{x}_i\}$, such that a graph isomorphism between T and $\hat{T}$ is found. The vertices of $\hat{T}$ are written as $\mathbf{x}_{\pi(i)}$ to designate sensed data points which have been reordered such that the ith vertex of $\hat{T}$ corresponds to the ith vertex of T; hence, the sequence of integers which describes the reordering process is the solution to the point correspondence problem. Since no structural relationships between the feature points are assumed to be known, the edges of the previous graphs are not considered in the matching process. Consequently, the graphs used in this algorithm are described solely by their respective vertices. Also, the feature points are not assumed to have any attributes; however, any such *a priori* information can be embedded into the graph and used to guide the matching process[1] [1].

To find the optimal graph $\hat{T}$, Cheng and Don's algorithm uses the backtracking tree search procedure which was described earlier. In this case, the search tree is organized such that each vertex represents a pairing of one sensed point with one template point; therefore, paths of length p in the search tree can be viewed as potential solutions to the problem. While traversing the paths of the tree, the vertices are examined one at a time, in depth-first order, and any vertex which is inconsistent with its ancestors is discarded; moreover, the entire subtree which is rooted at the inconsistent vertex is rejected by the matching algorithm. To verify that a vertex is consistent with its ancestors, two types of tests are made. The first test determines whether or not the structures which are formed by the two partially matched point sets are congruent. The congruence of matched structures is established using a geometric constraint, which exploits the principle of invariance of distance measures under rigid body motion. Specifically, the latter principle states that the distance between any two

[1]This is not attempted as part of the computer simulations, in Chapter 6, since the use of *a priori* information is considered to be beyond the scope of this thesis.

points which have been extracted from the same rigid object remains fixed, regardless of the position or orientation of the object [47]. A similar geometric constraint, based on the angle between two lines, could also be employed to prune the search, but was not used in Cheng and Don's algorithm to save computation time for the angle calculation. The second test which is used to verify the consistency of a vertex with its ancestors is based on the error accumulated in traversing a path. Essentially, this test ensures that the accumulated error between corresponding points in traversing a given path is smaller than the error of the best previous candidate solution; hence, only paths which are progressively closer to the optimal solution, with respect to the accumulated error, are explored.

For illustration, consider the set of template points $\{\mathbf{y}_i\}$ as a polygon, such that $\mathbf{y}_i$ is a vertex of T and the distance $d(\mathbf{y}_i, \mathbf{y}_{i+1})$ is the length of the ith edge $\mathbf{y}_i\mathbf{y}_{i+1}$. In this case, the objective is to construct a polygon from the points in the sensed data set such that the geometric structures of the two matched point sets are congruent. This is accomplished using a backtracking tree search procedure, in combination with the two pruning mechanisms previously described. The matching algorithm begins by finding two points in the sensed data set, say $\mathbf{x}_i$ and $\mathbf{x}_j$, whose distance $d(\mathbf{x}_i\mathbf{x}_j)$ is within a threshold ϵ of the length of the first edge $\mathbf{y}_1\mathbf{y}_2$ in the template graph T. Specifically, the condition on the pair of points $(\mathbf{x}_i, \mathbf{x}_j)$ is:

$$|d(\mathbf{x}_i, \mathbf{x}_j) - d(\mathbf{y}_1, \mathbf{y}_2)| \leq \epsilon, \tag{5.2}$$

where $|\cdot|$ denotes the absolute value of a real number. According to Cheng and Don, the value chosen for the so-called *edge threshold* ϵ is not crucial in their algorithm, provided that it is large enough. That is, beyond some critical value, the choice of a larger ϵ does not affect the results of the algorithm [1]. If the points in both data sets contain Gaussian noises with zero mean and a variance σ^2, it can be shown that an upper bound on ϵ is $8\sqrt{3}\sigma$ [1].

Having found the points $\mathbf{x}_i$ and $\mathbf{x}_j$ which satisfy equation (5.2), two distinct

initial pairings are possible: $\{(\mathbf{y}_1, \mathbf{x}_i)\ (\mathbf{y}_2, \mathbf{x}_j)\}$ and $\{(\mathbf{y}_1, \mathbf{x}_j)\ (\mathbf{y}_2, \mathbf{x}_i)\}$. Both of the previous cases are separately considered as possible initial pairings in the matching process. When the first case is used as a starting point, the algorithm proceeds to search for a third point in the sensed data set, say $\mathbf{x}_k$, such that the triangle $\triangle \mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ formed by the sensed points $\{\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k\}$ is congruent to the triangle $\triangle \mathbf{y}_1\mathbf{y}_2\mathbf{y}_3$ formed by the corresponding template points $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$. This implies that $\mathbf{x}_i\mathbf{x}_k$ must be closed to $\mathbf{y}_1\mathbf{y}_3$ and that $\mathbf{x}_j\mathbf{x}_k$ must be closed to $\mathbf{y}_2\mathbf{y}_3$. To find the next corresponding point, say $\mathbf{x}_l$, a similar approach is taken; that is, $\mathbf{x}_l$ is chosen such that the triangles $\triangle \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l$ and $\triangle \mathbf{y}_2\mathbf{y}_3\mathbf{y}_4$ are congruent. This procedure is repeated until all p points in the sensed data set have been visited. An example illustrating the previous steps is shown in Figure 5.7. If a match exists between the two point configurations, then

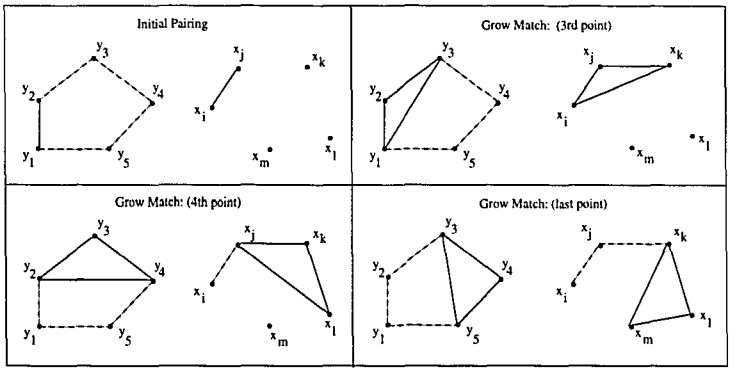


Figure 5.7: Matching congruent triangles: Dashed lines show established edges. Solid lines show edges being compared.

the last remaining point in the sensed data set to be visited when traversing a given path of the search tree, represents a leaf. Once the leaf has been visited, the algorithm backtracks to the previous vertex and continues to search for other solutions with a smaller accumulated error. After the entire tree has been explored (*i.e.* , all possible solutions have been enumerated), a number of potential solutions, or so-called *candidate*

graphs, have been found.

To determine which of the candidate graphs is the optimal solution $\hat{T}$, each graph is rated according to a similarity metric called the *kth order error*. Essentially, the kth order error measures the accumulated error between the vertices in a candidate graph $\tilde{T}$ and the corresponding vertices in the template graph T, when k edges in both T and $\tilde{T}$ are used to test for congruent structures. The triangle comparison previously described is a specific example in which $k = 2$. A formal definition of the k-th order error measure between the graphs $\tilde{T}$ and T is [1]

$$|\tilde{T} - T|_k = \sum_{m=2}^{p} \sum_{n=1}^{min(k,m-1)} |d(\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}_{m-n}) - d(\mathbf{y}_m, \mathbf{y}_{m-n})|, \tag{5.3}$$

where $T = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_p)$, $\tilde{T} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)$, and the vertices $\{\tilde{\mathbf{x}}_i\}$ are points in the sensed data set that have been selected by the matching algorithm. Note that this similarity metric is somewhat different than the least-squares error metric that is used in the SVD-based algorithm. The k-th order error involves the mutual distances between the points *within* each data set; specifically, it attempts to reorder the points in the sensed data set such that the mutual distances between the feature points match the corresponding mutual distances between the feature points in the template data set. In the case of the least-squares error metric, the aim is to minimize the square of the distance between each point in the sensed data set and each point in the template data set.

After a candidate graph is found, its kth order error measure from the reference graph T is computed using (5.3). The measure is then used to compare the newly generated graph with previously obtained candidate solutions. If the new candidate graph has the lowest kth order error of any other graph previously generated by the algorithm, it is kept as the best solution; otherwise, it is discarded, and a better solution is sought. Once the entire search tree has been explored, only the candidate graph with the smallest k-th order error is retained; consequently, the optimal solution $\hat{T}$ is the

candidate graph which satisfies [1]

$$|\hat{T} - T|_k \leq |\tilde{T}_l - T|_k, \quad \text{for all } l = 1, 2, 3, \ldots , \tag{5.4}$$

where T_l, $l = 1, 2, 3, \ldots$, are the candidate graphs that have been found by the tree-search procedure.

In addition to its role as a similarity metric , the k-th order error is closely related to a so-called *graph threshold*, which is used to prune the search tree. Initially, the graph threshold δ is set to the value $p(p-1)\epsilon/2$, since there are $p(p-1)/2$ edges in a p-node complete graph and each edge difference is not more than ϵ [1]; an assumption that represents a worst-case scenario. After a candidate graph is found, its kth order error is adopted as the new graph threshold, and the search is backtracked to find the next lower-error solution. In general, a sensed point $\mathbf{x}_l$ is added as the $(i+1)$th vertex of the working candidate graph only if it passes two kinds of tests. First, edge differences in the candidate graph must be less than the edge threshold ϵ. Second, the accumulated k-th order error of the working candidate graph, with the new point included as the $(i+1)$th vertex $\tilde{\mathbf{x}}_{i+1}$, must be less than the graph threshold δ. The accumulated kth order error α in a working candidate graph with $i+1$ vertices is given by [1]

$$\alpha = \sum_{m=2}^{i+1} \sum_{n=1}^{min(k,m-1)} |d(\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}_{m-n}) - d(\mathbf{y}_m, \mathbf{y}_{m-n})| \ , \tag{5.5}$$

where $\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_{i+1}\}$ are the vertices of the working candidate graph.

To see how the threshold tests are used in the tree search procedure, consider an example in which $k = p$ and a working candidate graph with i nodes has been generated. In this case, adding the lth point in the sensed data set $\mathbf{x}_l$ as the $(i+1)$th vertex of the working candidate graph produces i new edges. To establish whether or not the new vertex is consistent with the other vertices in the working graph, the i new edges are compared with the associated edges in T. If any edge difference exceeds the edge threshold ϵ, or if the accumulated kth order error α exceeds the graph threshold δ,

then $\mathbf{x}_l$ is not considered to be the $(i+1)$th vertex of $\tilde{T}$, and the point $\mathbf{x}_{l+1}$ is considered next. If all the remaining points $\mathbf{x}_l, \mathbf{x}_{l+1}, \ldots, \mathbf{x}_p$ in the sensed data set fail to satisfy the threshold tests, the ith vertex $\tilde{\mathbf{x}}_i$ is identified as a bad node and is deleted from the working candidate graph. The traversal is then backtracked to the previous vertex, and the search continues. A pseudocode description of the entire matching algorithm can be found in Appendix A.

5.3.2 Occluded Points

The matching algorithm described in Section 5.3 can also be used to identify occluded points in one or both of the point sets being matched. In this case, the sensed data set and the template data set are given by $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$ and $\{\mathbf{y}_i : i = 1, 2, \ldots, q\}$, respectively, where p may be different than q. For the case in which $p > q$, if every point in the template data set has a corresponding point in the sensed data set, the matching algorithm can be used to find the q sensed points that have a matching point in the template data set. The remaining $p - q$ points in the sensed data set are identified as occluded points. Conversely, if $q > p$ and all points in the sensed data set have corresponding points in the template data set, the same procedure is employed but with the roles of the two data sets switched.

If occluded points can occur in both of the data sets to be matched, a similar matching algorithm can still be used. In this case, every point in the template set is initially viewed as an extra point; hence, the template points are all matched to so-called *null points* at the outset of the tree search procedure. The objective of the modified algorithm is to find a feasible graph which satisfies edge constraints and has a maximal number of successfully paired points. If several candidate graphs exist which have the same number of matched points, then the one whose accumulated kth order error is the least is chosen as the optimal solution.

5.3.3 A Data Splitting Strategy

In principle, the computation time of a backtracking tree search procedure grows exponentially with the number of vertices to be visited in the tree [44]. Having recognized this fact, Cheng and Don devised a so-called *data splitting strategy* which reduces the computation time of their algorithm. The strategy is initiated by dividing the set of template points into m subsets. Then, each of the subsets is matched, one at a time, against the sensed point set, using the algorithm in Section 5.3. The matching process is handled as described in Section 5.3.2, for the case in which $p > q$. Each time a subgraph isomorphism is found, the size of the sensed data set is reduced by removing all of the points for which correspondences have been established. The splitting algorithm is as follows [1]:

Splitting Algorithm

Notation:

$E_1 = \{\mathbf{y}_i : i = 1, 2, \ldots, q\}$: template point set

$E_2 = \{\mathbf{x}_i : i = 1, 2, \ldots, p\}$: sensed point set

e_i : subset of E_1, such that $\bigcup_{i=1}^{m} e_i = E_1$ and $e_i \bigcap e_j = \emptyset$, for all $i \neq j$

Algorithm:

Begin

1. Split E_1 mutually exclusively into m subsets $e_1, e_2, \ldots, e_m$.
2. **For** $i = 1$ **to** m **do** steps 3-4.
   Begin
3. Call matching algorithm for e_i and E_2.
4. Remove points in $\hat{T}$ from E_2. Recall that the vertices of $\hat{T}$ are the points in E_2 which match the points in e_i.
   End

End

When E_2 and e_1 are matched, the points in $e_2, e_3, \ldots, e_m$ are considered as occluded points. Having established the correspondences between the two previous point sets, the points in E_2 which are found to match those in the subset e_1 are subsequently removed. The remaining points in E_2 are then matched with the subset e_2. In this case, the points in $e_3, e_4, \ldots, e_m$ are considered as occluded points. The same procedure as before is repeated until all m subsets of E_1 are matched.

As discussed in [1], a significant reduction in the computational requirements of the matching algorithm can be achieved with the data splitting strategy. The gain in speed is largely attributed to a reduction in the depth of the search tree to be traversed when subgraph isomorphisms are attempted. When E_1 and E_2 are matched, the depth of the longest path in the tree is q; consequently, the total number of edges visited, in the worst case, is [1]

$$C\begin{pmatrix} p \\ q \end{pmatrix} \doteq \frac{p!}{q!(p-q)!} \ , \tag{5.6}$$

where $a! \doteq a(a-1)(a-2)\cdots 1$, such that a is a nonzero integer. Now, let s_i denote the size of the i-th subset e_i of the pattern E_1. Using the same argument as above, when E_2 and e_1 are matched using the data splitting strategy, the depth of the search tree is s_1; moreover, when E_2 and e_2 are matched, the algorithm traverses only the subtree beneath the path $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_{s_1}$, whose depth is s_2. The total number of edges visited in the worst case, for the data splitting strategy is, therefore [1],

$$C\begin{pmatrix} p \\ s_1 \end{pmatrix} + C\begin{pmatrix} p-s_1 \\ s_2 \end{pmatrix} + \cdots C\begin{pmatrix} p-\sum_{i=1}^{m-1} s_i \\ s_m \end{pmatrix} \ , \tag{5.7}$$

which is much smaller than worst case scenario described by (5.6), since $s_i < p$, for all i.

5.4 Summary

This chapter illustrates a graph-based approach for solving point matching problems. The structural approach is based on the premise that auxiliary information may be available which can be used in addition to the features in the matching process. Graphs are used to represent the patterns, such that the vertices describe the features and the edges hold relational information between pairs of vertices. In this study, only the vertices are considered, as no additional information is assumed; nevertheless, such information can easily be embedded in the formulation.

The point-matching problems which are described in Chapter 4 can be formulated as equivalent graph-matching problems; specifically, the problem of matching point sets with the same cardinality, when a one-to-one correspondence exists between all points, involves the search for a graph isomorphism, while other problems involve the search for a subgraph isomorphism. Both types of graph-matching problems can be solved using a tree-search procedure, which enumerates all possible solutions to the problem. If an appropriate pruning mechanism is used, a large number of paths in the search-tree can be eliminated without traversing them; hence, the search need not be completely exhaustive.

The method proposed by Cheng and Don [1] uses a backtracking tree-search procedure, in conjunction with two pruning mechanisms. The first mechanism is the so-called edge threshold, which exploits the principle of invariance of distance measures under rigid body motion. The second pruning mechanism is the so-called graph-threshold, which ensures that the error accumulated in traversing new paths decreases monotonically. For each candidate solution, the accumulated kth order error is computed, and a lower-error solution is sought by backtracking through the tree and traversing unexplored paths. The final solution is the candidate matching graph which has the smallest kth order error.

When extra or missing points exist in one of the data sets being matched, the graph-matching algorithm can be used to identify them, as described in Section 5.3.2. The case in which occluded points exist in both of the data sets can also be handled using the same approach, but requires a significant modification to the basic strategy. The approach for handling occluded points in one of the data sets can be used to execute a data splitting strategy which can significantly improve the speed with which a solution is found; the price to be paid for this increase in speed is a decrease in the accuracy of the final solution in terms of the number of incorrect matches.

Chapter 6

Comparison of Algorithms

6.1 Introduction

The two algorithms compared in this study are clearly different in many respects. The graph-matching algorithm, in Chapter 5, is a graph-theoretic approach for solving the point-correspondence problem. The search strategy is based on an exhaustive tree search procedure which is guided by a specially constructed similarity metric (*i.e.* the k-th order error). In contrast, the SVD-based algorithm is a function optimization approach that is largely founded on algebraic group theory. The search strategy in this approach is based on straightforward iterative computations which are aimed at minimizing a predetermined error function in the least squares sense. Both of the algorithms yield an optimal solution to the point correspondence problem; yet, the search strategies which are employed by the algorithms to find this solution are quite different.

In this chapter, both approaches for solving the point-correspondence problem are compared through computer simulations. The algorithms are compared from two viewpoints. First, the dynamic behaviour of the two algorithms is considered in Sec-

tion 6.3. This is achieved by observing the progress of each algorithm from an initial solution, through several intermediate results, to the final solution. The latter observation can provide insight into the computational requirements of the algorithms for finding a solution, under various sets of running conditions. The second basis of comparison, which is discussed in Section 6.4, is the speed and accuracy with which an optimal match is obtained. Here, the trends in both speed and accuracy under various conditions are compared, rather than the absolute values.

6.2 Experimental Set-Up

6.2.1 Hardware/Software Considerations

Both the SVD-based algorithm and the graph matching algorithm were implemented in C and were run on a SUN SPARC10 computer. This common hardware/software platform was used to reduce any hardware or software related dependencies which could affect the interpretation of the results.

All programs that were used to generate input data for the matching algorithms were written in C. A number of three-dimensional graphical objects were created using the PHIGS programming library for 3D graphics [71] to obtain point features derived from a 3-D scene. Point features were also randomly generated within volumes of predefined sizes. A complete description of the data sets, and the methods that were used for generating them, is given in Section 6.2.2.

6.2.2 Data Sets

Two types of synthetic data were used as inputs to the matching algorithms. In the first type, 3D data points were generated randomly within a volume of size $l \times l \times l$

and centered at the point $(l/2, l/2, l/2)$. This was done by randomly selecting a real number in the range $[0, l]$, according to a uniform probability distribution function, for each coordinate of a given data point. The data points generated in this way were subsequently used as the template feature points: $\{\mathbf{y}_i : i = 1, 2, \ldots, p\}$. The sensed feature points $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$, were obtained from the template points according to the noisy rigid motion model

$$\mathbf{x}_i = \Theta \mathbf{y}_i + \mathbf{b} + \mathbf{n}_i \,, \tag{6.1}$$

where Θ is a 3D rotation matrix, $\mathbf{b}$ is a 3D translation vector and the $\mathbf{n}_i$ are 3D stochastic noise vectors which are mutually statistically independent. The noise vectors were generated such that each component of a noise vector was a normal (Gaussian) random variable with zero mean and variance σ^2. The order of the sensed points was then shuffled, using a known permutation matrix Π, to establish new correspondences between the feature points in the two data sets. The centroids of the patterns were removed from the data points (*c.f.* Section 4.2.1) as a preprocessing step in both of the matching algorithms.

An example of two randomly generated data sets is shown in Table 6.1. The template points in the table were obtained by randomly generating ten points in a volume of size $50 \times 50 \times 50$ and centered at $(25, 25, 25)$. These points were then rotated 0.27 radians about the axis passing through the origin and the point $(0.20, 0.50, 0.84)$, and were translated by the vector $\mathbf{b} = [-1 \quad 1 \quad 0]$. The order of the rotated points was then arbitrarily shuffled, and the permutation was recorded in order to verify the result of running the algorithm. Gaussian noises with zero mean and variance $\sigma^2 = 0.1$ were added to the reordered points. The resulting "sensed" data points are shown in Table 6.1, such that corresponding points are listed in the same row.

To generate two point sets with different cardinalities: $\{\mathbf{y}_i : i = 1, 2, \ldots, q\}$, and $\{\mathbf{x}_i : i = 1, 2, \ldots, p\}$, where $p \neq q$, the following procedure was followed. For the case in which $p < q$, a set of q template points was randomly generated and was then used

Reference Points				Sensed Points			
point number	Coordinates			point number	Coordinates		
	x	y	z		x	y	z
1	48.2426	12.7479	3.5168	2	43.0443	24.1119	- 1.4199
2	6.1924	26.3169	8.0042	3	0.2728	28.0280	8.6220
3	25.8854	5.2592	42.0788	8	28.9075	10.0625	38.8497
4	18.4281	21.1945	16.3614	6	14.0978	25.1482	15.6389
5	35.6773	28.8388	43.5942	7	33.0104	35.9032	40.3433
6	37.9123	41.8299	34.9896	9	31.4444	49.1609	32.3048
7	19.4530	46.7621	30.2801	4	11.6454	50.1038	30.3856
8	18.1042	26.9752	22.7536	5	13.6982	30.1867	22.3697
9	20.0846	12.0220	4.1543	1	15.9446	17.3434	2.5225
10	21.0789	23.6190	14.4883	10	16.2216	27.8915	13.5441

Table 6.1: Two randomly generated point sets

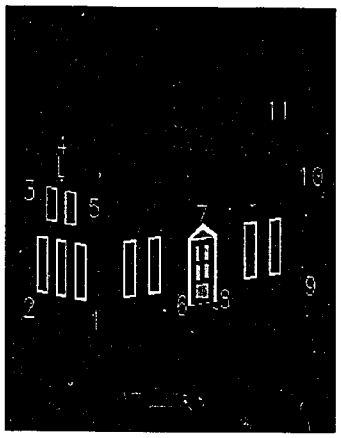
to compute q sensed points, as described previously. Then, $q - p$ points were removed from the sensed data set, leaving p points. These p points were then shuffled according to a predetermined p-dimensional permutation matrix. For the case in which $p > q$, the same procedure was used, except that $p - q$ points were added to the q sensed points. An example of two data sets with $p > q$ is shown in Table 6.2. In this case, nine template points were randomly generated in a volume of size $50 \times 50 \times 50$ and centered at $(25, 25, 25)$. The same motion parameters and noise as in the previous example were used to compute the nine sensed points with correspondences in the reference set. Three additional points were generated in the same volume that was used for the template points; these were appended to the list of sensed points. The order of the twelve sensed points was then arbitrarily shuffled, and the permutation was recorded. The two data sets are shown in Table 6.2 such that corresponding points are listed in the same row.

The second type of synthetic data sets involved feature points that were obtained from one or more artificially generated 3-D graphics objects. A graphical user interface

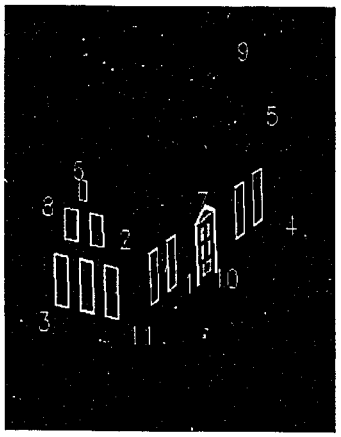
Reference Points				Sensed Points			
point number	Coordinates			point number	Coordinates		
	x	y	z		x	y	z
1	10.9480	2.3522	33.9432	12	13.4161	4.6849	32.5308
2	33.9648	46.7346	19.1751	9	23.7653	53.7317	17.9755
3	25.9708	41.5483	1.7286	8	14.9329	47.1077	1.6211
4	2.6731	26.4850	33.5575	2	0.3778	25.7955	35.0103
5	0.3849	19.1708	3.3421	11	- 4.7116	19.9326	4.5613
6	20.8743	34.3386	29.4488	4	16.0089	37.7784	28.7652
7	46.5218	42.3083	26.3464	6	38.3476	52.0146	22.6924
8	4.5982	32.6959	20.8000	5	- 0.6090	32.7235	22.2999
9	35.0595	45.5160	38.1099	3	27.8748	52.1977	36.2233
				1	48.8668	19.2696	12.0344
				7	11.7977	3.6799	36.8224
				10	15.6421	32.7621	37.4600

Table 6.2: Two randomly generated point sets with $p > q$

(GUI) was created, which permitted the user to either select the template points on the object(s) manually, or to employ a predetermined set of reference points. The GUI also provided a means of observing a number of intermediate results in the matching process, using 3-D objects. Given the feature points in the template data set, the sensed points were obtained in the same way as in the case of the randomly generated data sets. This type of data was used to simulate applications, such as in computer vision or image processing, for which the patterns to be matched represent three-dimensional objects. In the previous type of application, the template pattern and the sensed pattern are obtained from two views of the same scene, which are taken from different perspectives. Figure 6.1 illustrates an example in which the scene consists of a single object. The numbers shown in Figure 6.1 illustrate both the physical location of data points on the object, and the order in which they appear in the respective pattern.



(a)



(b)

Figure 6.1: Two Views of an artificially generated 3-D object: (a) View 1 (Template Image) (b) View 2 (Sensed Image)

6.3 Algorithm Dynamics

In this section, the dynamic behaviour of the two matching algorithms is examined. Specifically, the two algorithms are compared in terms of how they progress from an initial solution to the final result. By observing this type of progression, some insight into the nature of the search strategies, for both of the matching algorithms, can be gained.

6.3.1 Graph Matching Algorithm

One way to observe the dynamic behaviour of the graph matching algorithm is to examine the tree that is traversed by the algorithm as it searches for the optimal solution. Consider an example in which the two sets of randomly generated points in Table 6.1 (*c.f.* Section 6.2.2) are used as inputs to the algorithm. The tree that was generated by running the algorithm with these two data sets is shown in Figure 6.2. This tree can be used to describe the progression of the matching process, as follows. The algorithm first tried to find pairs of sensed points whose mutual distances are within a tolerance ϵ (*i.e.*, the edge threshold) of the two reference points $\mathbf{y}_1$ and $\mathbf{y}_2$. Seven pairs of sensed points which satisfied the distance criterion were found by the algorithm, in total. Fourteen initial pairings were generated for the two reference points $\mathbf{y}_1$ and $\mathbf{y}_2$, since two different orders are possible for each pair of sensed points. Five of these initial pairings, which grew to at least the second level of the tree, are shown in Figure 6.2. These pairings are represented by nodes in the first level of the tree. Initial pairings which did not grow past the first level of the tree are not shown in the figure. The numbers in parentheses next to each node in the figure denote the pairing between a reference point and a sensed point; specifically, the left number in the parentheses denotes a reference point and the right number denotes the corresponding sensed point.

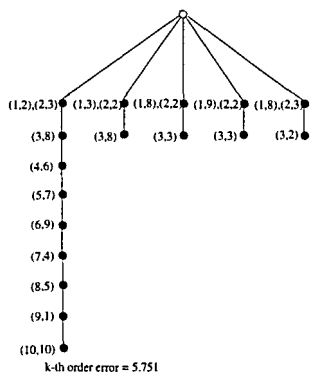


Figure 6.2: Result of running the graph matching algorithm

Each of the pairings at the first level of the tree initiated a separate matching process. For the first partial match (*i.e.* $(1,2),(2,3)$), reference point 3 was matched to sensed point 8, since the latter point satisfied the distance criterion and since the accumulated error with this new point was less than the current graph threshold. Other nodes along the same branch of the search tree were established in the same way, until a complete match was found. The k-th order error of the first candidate solution, which is shown below the last node in the branch, was then used as the graph threshold for finding the next best solution. In backtracking up the same path, none of the untried sensed points were able to satisfy both the distance criterion and the new graph threshold. All of the matching processes which were initiated by the other initial pairings proceeded in this manner; however, none of the partial matches were able to grow beyond the second level of the tree.

The effects of noise on the dynamic behaviour of the graph matching algorithm is illustrated by the following example. Consider the data sets shown in Table 6.3, which were generated in exactly the same way as in the first example, except that the

Reference Points				Sensed Points			
point number	Coordinates			point number	Coordinates		
	x	y	z		x	y	z
1	40.6633	27.8418	36.9498	6	36.6700	36.7538	33.0836
2	15.8004	6.7666	26.4274	7	16.9167	10.5210	23.7908
3	15.5369	29.4060	25.9042	5	11.8337	31.4830	25.6312
4	21.5424	12.9421	18.5113	10	20.1889	17.0706	16.5221
5	19.6509	22.3433	23.7948	2	17.0278	25.5883	22.6240
6	19.3916	13.9647	3.9132	4	15.9867	18.3827	2.0326
7	18.4871	12.6961	33.5892	8	19.5363	15.9299	31.3038
8	33.8118	25.6968	36.4304	1	31.7224	31.6081	33.5448
9	36.0384	47.2376	23.0349	3	25.7509	54.5714	21.9845
10	47.0082	16.0780	23.0217	9	44.7324	26.5068	17.1901

Table 6.3: Randomly generated point sets with $\sigma^2 = 0.5$

variance σ^2 of the noise in the sensed data set was increased from 0.1 to 0.5. The tree that was generated by running the algorithm with these two data sets is shown in Figure 6.3. In this case, more partial matches were generated than in the previous example. This is because the increased noise level adds more embedded local symmetries in the sensed data set; hence, a greater number of sensed points were able to satisfy the distance constraint. However, only four out of the twenty-two initial partial matches generated by the algorithm grew past the second level of the tree, and only two grew past the third level. The maximal match, which grew to the ninth level of the tree, is the correct match.

The search tree in Figure 6.3 shows that incorrect matches are detected early in the matching process. This is due to the power of the two pruning mechanisms which are employed by the algorithm. Naturally, the effectiveness of the pruning constraints is weakened when the similarity in the interdistances between the feature points increases; this is especially true when the distance between points is smaller than the edge threshold, ϵ. In general, computer simulations have shown that the number of

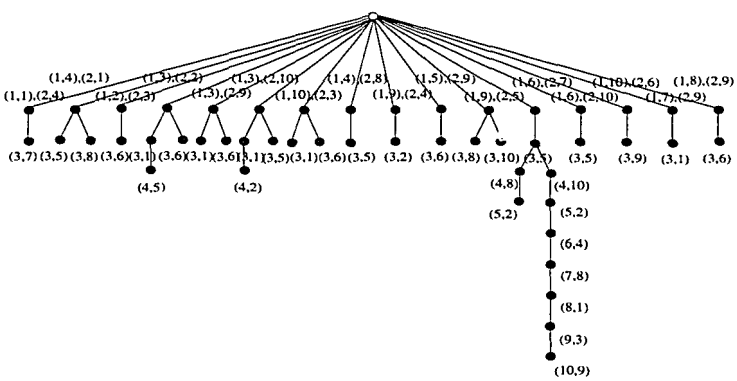


Figure 6.3: Result of running the graph matching algorithm when the noise variance σ^2 is increased

partial matches which are obtained by the matching algorithm rapidly increases as the number of feature points lying within an enclosing volume of a given size is increased and/or the noise added to the points is increased. This observation is supported by the curves shown in Figure 6.4. These curves were obtained by running the graph matching algorithm with data sets that were randomly generated within an enclosing volume of size 200^3. For each curve, the noise variance that was added to the sensed data set was kept fixed, while the number of feature points being matched was varied from 10 to 100.

Another way to observe the dynamic behaviour of the graph matching algorithm is illustrated in Figure 6.5. In this case, the two views of the 3-D "house" object shown in Figure 6.5 (a) – (b) were matched using the graph matching algorithm. Three candidate solutions were obtained, which are shown in Figure 6.5 (c) – (e). The first candidate solution, in Figure 6.5 (c), contains only 4 correct point correspondences (*i.e.*, points 8–11); however, it can be seen that the sensed feature points are already grouped together in the appropriate regions. For example, the points $\{1, 2, 3, 4, 5\}$,

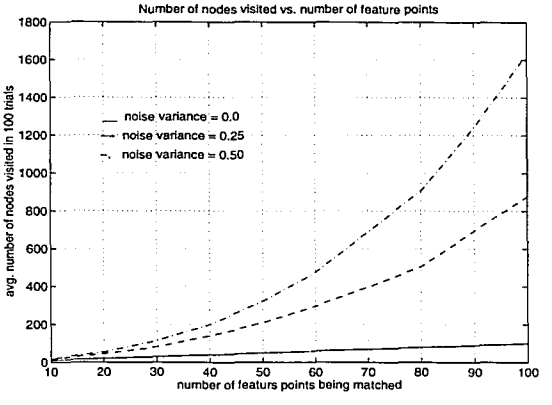
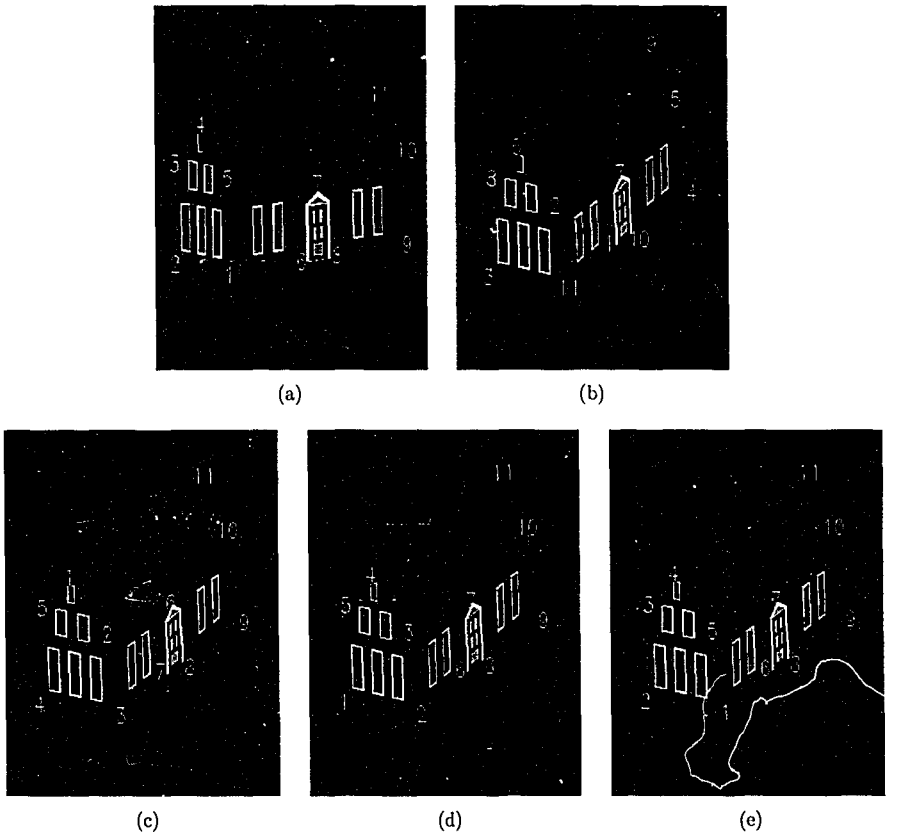


Figure 6.4: Number of nodes visited in the matching process, under various conditions

$\{6, 7, 8\}$, and $\{9, 10, 11\}$ are grouped together in the same regions on the object, in both Figure 6.5 (a) and Figure 6.5 (c). This observation can be explained in terms of the triangle comparisons that are carried out by the algorithm in traversing the search tree, when $k = 3$. The triangles formed by the points marked $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{4, 5, 6\}$, and so on, in both Figure 6.5 (a) and Figure 6.5 (c) are matched such that, for each pair of triangles compared, the edges are the same (to within the edge threshold). Also, the accumulated error of all such triangles is less than the graph threshold. In the second candidate solution, the k-th order error of the first solution is used as the graph threshold; consequently, the triangles formed by the points marked $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{4, 5, 6\}$, and so on, in Figure 6.5 (d) are more closely matched with the corresponding triangles in Figure 6.5 (a) than in the case of the first candidate solution. For this reason, the number of erroneous correspondences in the second candidate solution is significantly lower than in the case of the first candidate solution. The same argument can be used to explain the results in the third, and final, solution in Figure 6.5 (e) in which all point correspondences are correct.



(a) (b)

(c) (d) (e)

Figure 6.5: Result of matching two views of the 3D "house" object: (a) View 1 (Template Image) (b) View 2 (Sensed Image) (c) First candidate solution (d) Second candidate solution (e) Final solution

6.3.2 SVD-Based Algorithm

The dynamic behaviour of the SVD-based algorithm can be observed by tracking the success of each of the component parts of the algorithm in minimizing the least squares mismatch between the two data sets. To see how this is done, consider the block diagram in Figure 6.6, which summarizes the three-step matching procedure in Section 4.3. In the first step of the procedure (*i.e.* SVD1), a rough estimate $\widetilde{\Theta}$ of the optimal

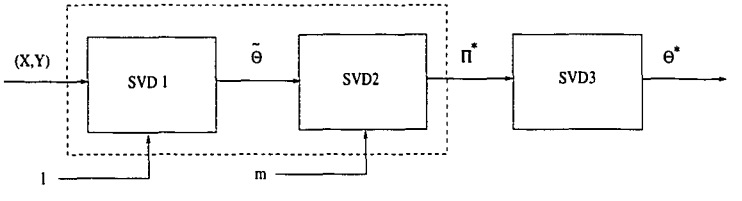


Figure 6.6: A Block Representation of Morgera and Lie Chin Cheong's algorithm

rotation matrix Θ^* is obtained by matching the normalized second moments of the patterns: $\mathbf{Q} = \frac{1}{p}\mathbf{X}\mathbf{X}^T$ and $\mathbf{N} = \frac{1}{q}\mathbf{Y}\mathbf{Y}^T$, where $\mathbf{X}$ and $\mathbf{Y}$ represent the sensed pattern and the template pattern, respectively. This requires l iterations of the SVD-based algorithm described in (4.32) - (4.36). After each iteration i of the algorithm, the least squares error of the matching process can be computed according to

$$\sigma(i) = ||\widetilde{\Theta}(i)^T\mathbf{Q}\widetilde{\Theta}(i) - \mathbf{N}||_F \tag{6.2}$$

$$= \mathrm{tr}[(\widetilde{\Theta}(i)^T\mathbf{Q}\widetilde{\Theta}(i) - \mathbf{N})(\widetilde{\Theta}(i)^T\mathbf{Q}\widetilde{\Theta}(i) - \mathbf{N})^T], \tag{6.3}$$

where $\widetilde{\Theta}(i)$ denotes the estimated rotation matrix after i iterations of SVD1. The matching process, for SVD1, can be observed by plotting the least squares error function $\sigma(i)$ for $i = 1, 2, \ldots, l$. A similar least squares error function $\eta(i)$ can be computed, which describes the progression of the search for the permutation matrix $\mathbf{\Pi}^*$, in the second step of the matching procedure (SVD2); specifically,

$$\eta(i) = ||\widetilde{\Theta}(l)^T\mathbf{X}\widetilde{\mathbf{\Pi}}(i) - \mathbf{Y}||_F \tag{6.4}$$

$$= tr[(\widetilde{\Theta}(l)^T \mathbf{X}\widetilde{\Pi}(i) - \mathbf{Y})(\widetilde{\Theta}(l)^T \mathbf{X}\widetilde{\Pi}(i) - \mathbf{Y})^T], \tag{6.5}$$

where $\widetilde{\Pi}(i)$ is the estimated permutation matrix after i iterations of SVD2 and $\widetilde{\Theta}(l)$ is the estimated rotation matrix after l iterations of SVD1. The progression of SVD2 can be observed by plotting the least squares error function $\eta(i)$ for $i = 1, 2, \ldots, m$. The permutation matrix $\widetilde{\Pi}(m)$ which is obtained after m iterations of SVD2 is taken as the optimal permutation matrix Π^* in subsequent calculations; specifically, it is used in computing the optimal rotation matrix Θ^* in SVD3, which consists of evaluating (4.25).

The result of running the SVD-based algorithm with the two randomly generated point sets in Table 6.1 and $l = m = 50$ is shown in Figures 6.7 - 6.8. The least squares error curve in Figure 6.7 shows that approximately 30 iterations of step 1 (SVD1) were required for the algorithm to converge to a stable initial estimate of the rotation matrix. Step 2 of the matching algorithm was initiated after $l = 50$ iterations of SVD1 and progressed in roughly the same manner, as illustrated by the least squares error curve in Figure 6.8. The matrix $\widetilde{\Pi}(50)$ obtained after $m = 50$ iterations of SVD2 was very close to $\Pi\mathbf{D}_I$, where Π is the exact permutation matrix and $\mathbf{D}_I$ is the diagonal square root of the identity matrix $\mathbf{I}$. In the final step of the algorithm (SVD3), the matrix $\widetilde{\Pi}(50)$ was used to compute the optimal rotation matrix Θ^*. This matrix was used to obtain the final estimated motion parameters, which are shown in Table 6.4. Note that the accuracy in estimating the motion parameters is improved as the number of iterations in SVD1 and SVD2 are increased.

In general, it is difficult to determine ahead of time how many iterations of SVD1 and SVD2 are needed for the algorithm to converge to accurate solutions for Θ^* and Π^*; however, the following trends were observed, for a number of experiments similar to the previous example:

- The rotation matrix $\widetilde{\Theta}$ in SVD1 did not need to converge to a stable solution, to obtain the expected results for Θ^* and Π^*; in fact, for small rotations (*i.e.*

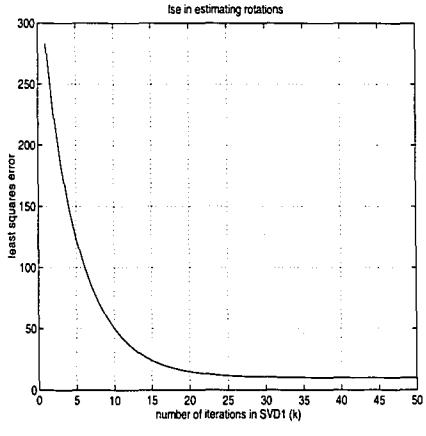


Figure 6.7: Least Squares Error Curve for SVD1

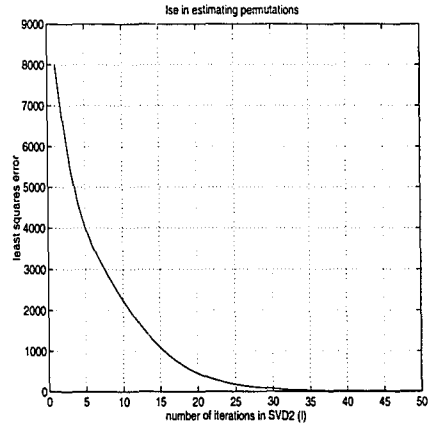


Figure 6.8: Least Squares Error Curve for SVD2

	Rotation Axis			Angle	Translation Vector		
	n_x	n_y	n_z		b_x	b_y	b_z
True	0.200	0.500	0.843	0.270	- 1.000	1.000	0.000
Estimated	0.175	0.519	0.836	0.265	- 1.377	0.950	0.319

Table 6.4: Motion parameters obtained by the SVD-based algorithm

rotation angles $\theta < 0.5$ rad), very few iterations of SVD1 were required for the algorithm to converge to the expected results (for both SVD1 and SVD2)

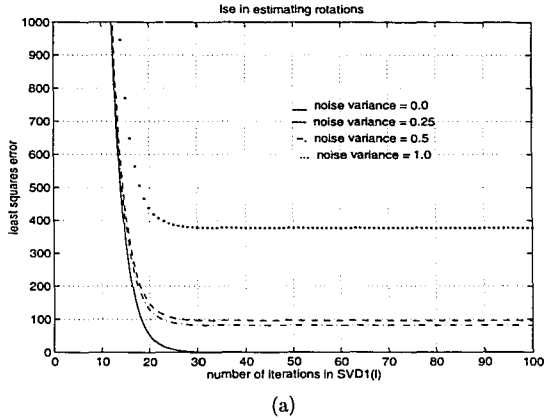
- As the dimensionality and/or cardinality of the point sets being matched was increased, so did the requirement on the number of iterations needed for the algorithm to converge to the expected results.

The results that were obtained in matching the data sets in Table 6.1 provide a good example of the first observation. Recall that approximately $l = 30$ iterations of SVD1 were required before obtaining a stable solution $\widetilde{\Theta}$. When the same experiment was run with $m = 50$ and various values of l, it was found that the results for $\mathbf{\Pi}(50)$ and Θ^* were virtually independent of the number of iterations (l) used in SVD1; in fact, it was possible to obtain the same results without running SVD1 altogether (*i.e.*, $l = 0$). An explanation for the previous observation is that the initial estimate $\widetilde{\Theta}(0) = \mathbf{I}$ was close enough to the optimal solution Θ^* that SVD2 could converge, using this rough estimate.

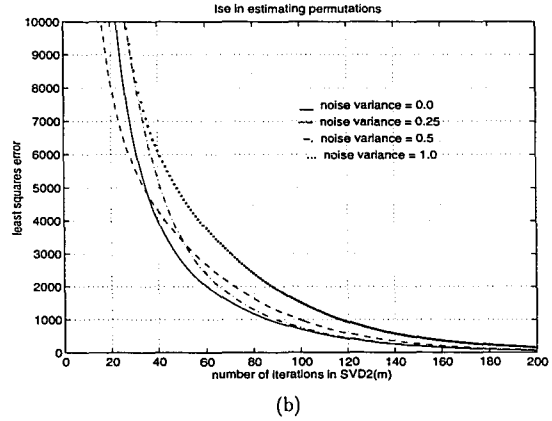
The effect of noise on the dynamic behaviour of the SVD-based algorithm is illustrated in Figures 6.9 (a) – (b). The least squares error curves in the two previous figures are the results of matching four pairs of data sets that were created as follows. A reference data set containing twelve randomly generated 3-D feature points was created. This reference pattern was then used to compute a sensed pattern, as described in Sec-

tion 6.2.2, with the motion parameters: $\theta = 0.785$ rad, $[n_x \;\; n_y \;\; n_z] = [0.27 \;\; 0.89 \;\; 0.37]$, $\mathbf{b} = [b_x \;\; b_y \;\; b_z] = [-2.7 \;\; 0.85 \;\; 12.4]$, and no noise (*i.e.* $\sigma^2 = 0$). The order of the sensed points was then shuffled using a predetermined permutation matrix $\mathbf{\Pi}$. Three other sensed patterns were obtained in exactly the same way, except that Gaussian noises with variances: $\sigma^2 = 0.25, 0.50$, and 1.0 were added to the points in the three sensed patterns, respectively. Each of the four sensed patterns was then matched with the reference pattern, using the SVD-based algorithm. Two least squares error curves (one for SVD1 and one for SVD2) were obtained for each of the noise cases. The eight resulting curves are shown in Figures 6.9 (a)- (b). These curves show that the rate of convergence of both SVD1 and SVD2 is not significantly affected by differences in noise variance. The rotation estimate $\widetilde{\mathbf{\Theta}}$ which is obtained by SVD1 varies significantly with differences in noise variance, as illustrated in Figure 6.9 (a); however, this variation does not significantly affect the permutation estimate $\widetilde{\mathbf{\Pi}}$, which is obtained by SVD2, and the overall least squares error of the matching process, as shown in Figure 6.9 (b).

Another way to illustrate the matching process using the SVD-based algorithm is shown in Figure 6.10. In this case, the two views of the 3-D "house" object shown in Figure 6.10 (a) - (b) were matched using the SVD-based algorithm. The image in Figure 6.10 (c) is the result after $l = 25$ iterations of SVD1. The orientation in the previous image was adjusted according to the estimated rotation matrix that was obtained in SVD1. The estimated motion parameters were quite close to the actual values, as shown in Table 6.5. The result of matching after $m = 100$ iterations of SVD2 is shown in Figure 6.10 (d). Comparing the previous figure with the template image in Figure 6.10 (a), it is seen that all point correspondences are correct. The final solution, after executing SVD3, is shown in Figure 6.10 (e). The small adjustment in the orientation of the house that is applied in the final solution is hardly noticeable, since the initial rotation estimate was already very close to the actual solution.

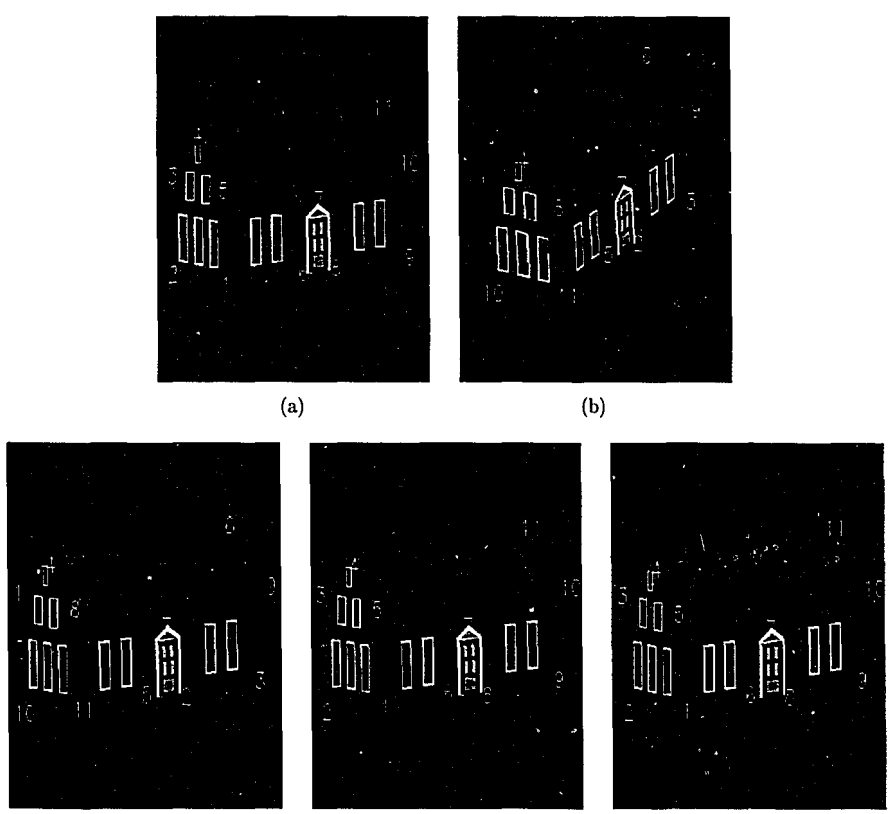


(a)



(b)

Figure 6.9: Effects of noise on the SVD-based algorithm: (a) LSE curves for SVD1 (b) LSE curves for SVD2



(a) (b)

(c) (d) (e)

Figure 6.10: Result of matching two views of the 3D "house" object: (a) View 1 (Template Image) (b) View 2 (Sensed Image) (c) Result after 25 iterations of SVD1 (d) Result after 100 iterations of SVD2 (e) Result after SVD3 (final solution)

	Rotation (Euler) Angles			Translation Vector		
	θ_x	θ_y	θ_z	b_x	b_y	b_z
Estimated Values	0.185	0.324	-0.119	-1.075	1.808	-0.172
Actual Values	0.20	0.30	-0.11	-1.0	2.0	0.0

Table 6.5: Motion parameters associated with the 3-D house object

6.4 Speed/Accuracy Comparisons

In this section, the two matching algorithms are compared in terms of the speed and accuracy with which a solution to the point correspondence problem is found; in particular, the attention is placed on the following:

1. The effects of both noise and "object size" [1] on the speed and accuracy of the two algorithms. The average cpu time (in milliseconds) that is required to find a solution to the correspondence problem is used as a measure of computational speed. Accuracy is measured as the total number of incorrect matches, summed over the number of trials used for averaging.

2. The average cpu time required to find a solution to the correspondence problem, as a function of the number of point features to be matched.

The two previous points describe important factors which typically affect the performance of a matching algorithm. Virtually all practical matching applications must be able to handle data sets which are corrupted by noise; consequently, it is important to determine to what extent the speed and accuracy of the matching algorithms are affected when the data sets are noisy. It also important to understand how the speed of the matching algorithms is affected when the number of points to be matched is varied,

[1] This is actually the size of the enclosing volume in which the feature points were randomly generated. See section 6.2.2 for more details.

since some applications require many points to be matched, while others require very few.

It should be noted that the term "incorrect match" only applies, in a strict sense, to the graph matching algorithm. This is because the graph matching algorithm yields a permutation matrix whose entries are exactly 1 or 0; consequently, it is possible to determine whether a match is correct or incorrect by comparing the result of the matching algorithm with the expected result. In the case of the SVD-based algorithm, the permutation matrix $\widetilde{\Pi}$ does not have entries which are exactly 1 or 0; instead, the permutation matrix approaches $\Pi \mathbf{D}_I$, where $\mathbf{D}_I$ is the diagonal square root of the identity matrix $\mathbf{I}$, such that the overall error of the estimated solution is minimized, in the least squares sense. To compare the results of the two algorithms, some processing was applied to the permutation estimate $\widetilde{\Pi}$ such that all entries were either 1 or 0. Specifically, the largest element in each column of $\widetilde{\Pi}$ that exceeded a predetermined threshold ξ was rounded up to 1.0, while all other elements were rounded down to 0.0. Note that this modified version of $\widetilde{\Pi}$ was used only to determine the number of "error" correspondences and was not used in any computations (*e.g.*, in SVD3).

The data sets that were used in the speed and accuracy measurements were all randomly generated, as described in Section 6.2.2. A program was written in C to measure the cpu time that was used in running the algorithms. All of the speed and accuracy results were averaged over 100 trials.

6.4.1 Graph Matching Algorithm

Matching of Point Sets With the Same Cardinality

The speed and accuracy measurements in this section reflect the performance of the graph-matching algorithm under the following conditions:

- point patterns having 10 point features each,
- no occluded or extra points in either of the two data sets
- the following set of motion parameters: rotation axis: $n_x = 0.23$, $n_y = 0.44, nz = 0.87$; rotation angle: $\theta = 0.97$ radian; translation vector: $t_x = 70$, $t_y = -9$, $t_z = 0.5$,
- noises with variances σ^2 ranging from $0.01 - 3.0$,
- three different enclosing volumes: $25^3, 50^3$ and 100^3,
- two different values of k: $k = 3, 10$, [2]

The effect of k on both the speed and accuracy of the algorithm can be observed by comparing the results in Tables 6.6 - 6.7, for which $k = 3$ and 10, respectively. It can be seen that the accuracy of the solutions obtained with the matching algorithm improves as k is increased. On the other hand, the computational requirements of the algorithm increase as k is increased. The previous trend becomes increasingly marked as the size of the volume in which feature points are generated decreases and as the noise variance increases. An increase in the computational requirements of the algorithm is expected as k is increased, since the number of comparisons that are performed whenever a new node in the search tree is visited is directly proportional to k; however, if the edge threshold ϵ is chosen appropriately, the chances that the node will be rejected are greater as k increases.

The results in Tables 6.6 - 6.7 are plotted in Figures 6.11 - 6.12, respectively, to illustrate the computational requirements of the algorithm as a function of the noise variance. In general, it can be observed that the computational requirements of the algorithm increase rapidly with increases in the noise variance; moreover, the sharpness of the increase in computational requirements is strongly dependent on the size of the volume in which the feature points have been randomly generated.

[2]Recall, from Chapter 5, that the parameter k determines the k^{th} order error measure to be used in the matching process.

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise	"Object" Size			"Object" Size		
σ^2	100^3	50^3	25^3	100^3	50^3	25^3
0.00	0	0	0	2.7	2.8	3.2
0.10	0	0	0	3.3	4.1	7.6
0.25	0	0	2	3.8	5.3	23.7
0.50	0	0	6	4.4	9.9	96.9
0.75	0	0	21	5.3	15.0	275.1
1.00	0	0	23	6.0	21.8	528.6
1.50	0	4	53	7.3	43.1	1006.8
2.00	0	6	88	9.2	95.0	1494.3
3.00	2	21	197	14.5	248.4	3016.1

Table 6.6: Results of graph matching algorithm for several noise variances and $k = 3$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise	"Object" Size			"Object" Size		
σ^2	100^3	50^3	25^3	100^3	50^3	25^3
0.00	0	0	0	2.8	3.1	3.0
0.10	0	0	0	3.3	4.7	8.2
0.25	0	0	0	4.1	6.7	23.7
0.50	0	0	4	5.0	10.7	128.8
0.75	0	0	6	5.6	15.2	353.2
1.00	0	0	8	6.4	23.4	962.2
1.50	0	0	12	7.9	46.2	4343.2
2.00	0	2	31	10.3	133.8	9074.8
3.00	2	4	68	17.3	312.7	24923.6

Table 6.7: Results of graph matching algorithm for several noise variances and $k = 10$

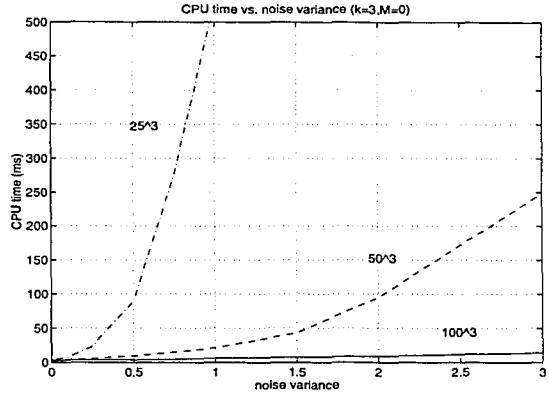


Figure 6.11: Trends in CPU requirements as a function of noise variance, $k = 3$

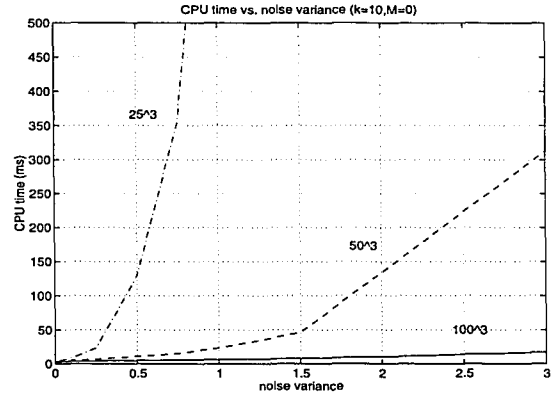


Figure 6.12: Trends in CPU requirements as a function of noise variance, $k = 10$

The result of using Cheng and Don's data splitting strategy is shown in Tables 6.8 - 6.9. The differences in the performance of the algorithm with and without data splitting is seen by comparing the speed measurements in Tables 6.8 and 6.6 and also in Tables 6.9 and 6.7, respectively. Comparisons can also be made between Figures 6.13 and 6.11 and between Figures 6.14 and 6.12. Note that the variable M has been introduced to indicate the size of the subsets to be matched when Cheng and Don's data splitting strategy is invoked. When $M = 0, 1$, no data splitting is used. When $M = 2$, the data sets are split equally into two subsets, and so on. In Figures 6.13 - 6.14, it can be seen that the data splitting strategy significantly reduces the computational requirements of the algorithm; however, the cost of this improvement in speed is an increase in the number of incorrect matches, as shown in Tables 6.8 - 6.9.

The computational requirements of the algorithm as a function of the number of feature points to be matched was investigated under the following conditions:

- three different noise variances: $\sigma^2 = 0.1, 0.25, 0.5$
- two different enclosing volumes: 100^3 and 200^3,
- k fixed at $k = 3$,
- $M = 0$ (*i.e.* no data splitting) and $M = 2$.

The results are shown in Tables 6.10 - 6.12, for $M = 0$, and in Tables 6.13 - 6.15, for $M = 2$. Once again, the effects of noise and "object size" were investigated. In general, the following is observed:

- The computational requirements of the algorithm increase rapidly as the number of feature points to be matched increases; this is clearly seen in Figure 6.15.
- The sharpness of the increase in computational requirements is strongly dependent on both the enclosing volume in which the feature points were generated and the variance of the noises added to the feature points; this is clearly illustrated in Figure 6.15.
- When the mutual distances between the feature points is large and the the perturbations in point positions due to additive noises is small, data splitting provides

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise	"Object" Size			"Object" Size		
σ^2	100^3	50^3	25^3	100^3	50^3	25^3
0.00	0	0	0	2.5	2.6	2.7
0.10	0	2	17	3.1	3.9	6.8
0.25	0	9	36	3.7	5.8	11.7
0.50	0	18	123	4.2	7.6	17.4
0.75	0	24	190	4.7	10.1	22.7
1.00	12	26	249	5.5	12.2	28.2
1.50	8	48	383	6.6	15.9	35.7
2.00	29	124	440	7.6	19.8	40.9
3.00	40	215	615	9.4	25.2	47.5

Table 6.8: Results of the graph matching algorithm using the data splitting strategy with $k = 3, M = 2$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise	"Object" Size			"Object" Size		
σ^2	100^3	50^3	25^3	100^3	50^3	25^3
0.00	0	0	0	2.5	2.4	2.4
0.10	0	0	0	3.3	3.6	6.4
0.25	0	7	21	3.8	5.5	14.5
0.50	4	18	78	4.3	7.7	24.7
0.75	11	24	133	4.9	10.4	32.5
1.00	7	52	229	5.6	13.3	40.5
1.50	11	55	345	6.5	19.8	50.2
2.00	17	81	493	8.2	23.6	57.5
3.00	28	163	544	10.7	32.6	70.4

Table 6.9: Results of the graph matching algorithm using the data splitting strategy with $k = 10, M = 2$

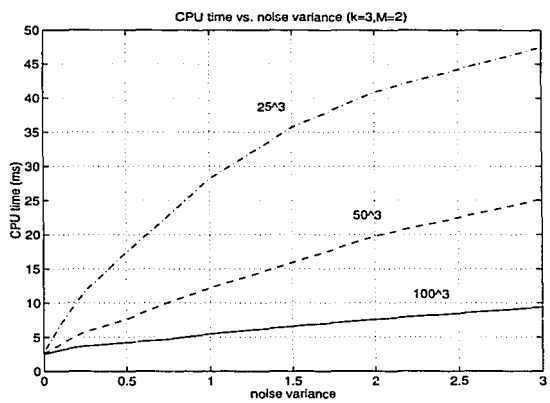


Figure 6.13: Plot of data splitting results, $k = 3, M = 2$

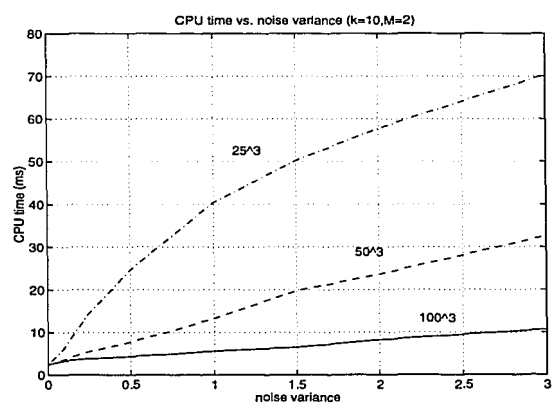


Figure 6.14: Plot of data splitting results, $k = 10, M = 2$

Noise σ^2 0.1	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	0	0	3.3	2.8
20	0	0	17.5	14.8
30	0	0	50.2	38.3
40	2	0	107.4	79.2
50	0	0	205.8	140.7
60	0	0	365.0	225.3
70	2	0	562.0	344.3
80	0	0	838.2	504.2
90	0	2	1296.1	683.2
100	0	0	1915.9	919.5

Table 6.10: Results using point sets with various cardinalities and $\sigma^2 = 0.10$

Noise σ^2 0.25	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	0	0	4.1	3.0
20	0	0	22.3	17.3
30	2	0	73.8	44.3
40	0	0	171.5	94.3
50	4	0	374.6	171.8
60	2	0	700.4	290.2
70	0	0	1168.7	443.2
80	2	0	2121.6	660.1
90	2	0	3248.9	985.9
100	4	2	6113.3	1400.3

Table 6.11: Results using point sets with various cardinalities and $\sigma^2 = 0.25$

Noise σ^2 0.50	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	0	0	4.2	3.2
20	0	0	32.3	19.1
30	2	0	128.0	52.6
40	0	0	325.8	117.3
50	2	0	952.4	226.9
60	4	0	2658.9	378.4
70	4	0	5608.4	635.5
80	4	0	12439.7	1000.7
90	6	0	31724.7	1475.1
100	8	2	69080.1	2443.2

Table 6.12: Results using point sets with various cardinalities and $\sigma^2 = 0.50$

Noise σ^2 0.10	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	5	0	3.9	3.2
20	0	0	16.9	13.8
30	30	30	55.4	37.5
40	122	40	108.6	76.1
50	100	0	205.0	140.6
60	32	30	358.7	224.2
70	39	142	563.2	340.1
80	329	80	905.8	491.9
90	818	180	1240.5	694.1
100	854	200	1850.9	998.6

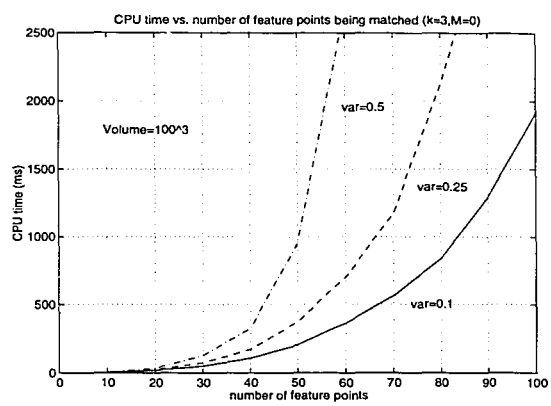
Table 6.13: Results using point sets with various cardinalities, $M = 2$ and $\sigma^2 = 0.10$

Noise σ^2 0.25	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	12	0	3.6	3.5
20	16	0	23.3	15.9
30	92	45	80.9	43.3
40	180	20	175.3	93.8
50	504	100	362.5	173.6
60	307	152	669.4	288.7
70	747	105	1169.8	458.5
80	488	322	1944.5	656.4
90	866	540	3028.6	940.4
100	1156	650	5008.2	1401.3

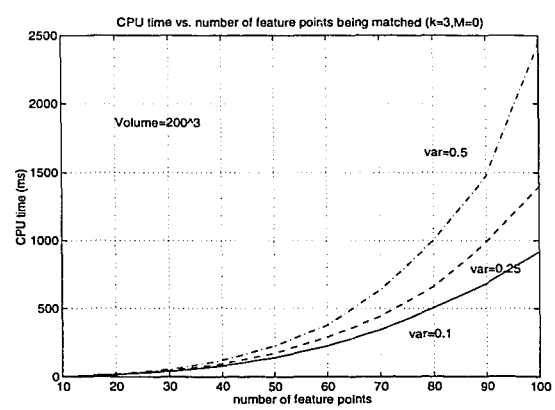
Table 6.14: Results using point sets with various cardinalities, $M = 2$ and $\sigma^2 = 0.25$

Noise σ^2 0.50	# incorrect matches in 100 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	4	0	4.7	3.4
20	44	20	31.9	17.7
30	137	47	121.9	54.9
40	331	200	306.0	117.3
50	481	352	792.7	233.5
60	758	214	1654.2	402.5
70	1241	420	3602.4	678.9
80	1303	724	7289.3	1025.0
90	1814	458	18729.4	1433.7
100	1778	1010	35422.5	2284.2

Table 6.15: Results using point sets with various cardinalities, $M = 2$ and $\sigma^2 = 0.50$



(a)



(b)

Figure 6.15: CPU requirements as a function of the number of points being matched (a) Enclosing volume: 100^3, (b) Enclosing volume: 200^3

little, if any, reduction in the computational requirements of the algorithm; this is seen by comparing the results in Tables 6.10 – 6.12 with those in Tables 6.13 – 6.15.

- As the mutual distances between the feature points decreases and/or the perturbations in the point positions increases, significant improvements in computational speed are obtained at the expense of an increased number of incorrect matches. The results shown in Tables 6.16 – 6.17, which were obtained using an enclosing volume of $(50)^3$ and noises having variance $\sigma^2 = 1.0$, support this observation.

Noise σ^2 1.0	# incorrect matches in 100 simulations	Average cpu time (ms) for each simulation
# feature pts	"Object" Size	"Object" Size
	50^3	50^3
10	58	11.4
15	159	82.7
20	297	696.6
25	382	3535.1
30	517	40946.4

Table 6.16: Results using point sets with various cardinalities, $k = 3$, $M = 2$, $\sigma^2 = 1.0$

Noise σ^2 1.0	# incorrect matches in 100 simulations	Average cpu time (ms) for each simulation
# feature pts	"Object" Size	"Object" Size
	50^3	50^3
10	2	19.7
15	4	283.7
20	18	8915.6
25	24	207793.5
30	--	--

Table 6.17: Results using point sets with various cardinalities, $k = 3$, $M = 0$, $\sigma^2 = 1.0$. Entries marked – took too long to execute 100 times

Matching of Point Sets With Different Cardinalities

For the correspondence problem in which the point sets to be matched have different cardinalities, Cheng and Don's algorithm [1] was implemented as described in Section 5.3.2. The results for $p = 30$ and $q = 5, 15, 25$ [3] are shown in Table 6.18. Once again, it can be seen that both the enclosing volume and the variance of the added noise have a strong influence on the computational requirements of the algorithm. The average cpu time required to find the solution is always greater in the case of the smaller enclosing volume (100^3) than in the case of the larger one (200^3); moreover, the effects of noise on both the speed and accuracy of the algorithm are significantly less pronounced in the case of the larger enclosing volume. In all cases, the computational requirements of the algorithm increase as the number of points in the template set increase from 5 to 25 points, but the average cpu time required to find a solution to the correspondence problem is always less than in the case of $p = q = 30$ feature points. The previous observation is supported by the results in Tables 6.10 - 6.12, for the noise variances $\sigma^2 = 0.1, 0.25$, and 0.5.

6.4.2 SVD-based Algorithm

Matching of Point Sets With the Same Cardinality

The results for Morgera and Lie Chin Cheong's SVD-based iterative algorithm are presented in this section. The point patterns to be matched were generated in exactly the same manner as in the case of the graph matching algorithm, and the experiments were also organized in a similar manner; that is, the effects of noise, "object size", and the number of feature points to be matched on both the speed and accuracy of the algorithm were investigated.

[3] Here, p is the cardinality of the sensed data set, and q is the cardinality of the template, or reference, data set

p= 30		# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
"Object" Size	Noise σ^2	q			q		
		5	15	25	5	15	25
100^3	0.00	0	0	0	13.6	19.5	26.3
	0.10	0	0	0	32.1	38.8	46.3
	0.25	0	0	2	47.8	59.2	67.7
	0.50	5	1	0	72.5	98.4	115.9
	1.00	9	3	4	127.8	246.2	321.1
	2.00	32	16	17	247.2	1793.5	6272.2
	3.00	63	31	21	325.7	9754.8	12761.0
200^3	0.00	0	0	0	12.0	19.0	26.4
	0.10	0	0	0	22.3	28.1	34.1
	0.25	0	0	0	26.4	33.7	40.8
	0.50	1	0	0	35.4	42.4	52.2
	1.00	1	1	1	46.9	59.2	69.8
	2.00	2	0	2	68.6	105.8	117.6
	3.00	3	3	0	102.1	154.8	196.7

Table 6.18: Results of the graph matching algorithm in the case of unmatched points

For this algorithm, two parameters which are specific to Morgera and Lie Chin Cheong's algorithm are of interest:

1. The number of iterations l used in SVD1 to establish a rough estimate $\widetilde{\Theta}$ of the rotation matrix Θ.

2. The number of iterations m used in SVD2 to find the permutation matrix $\widetilde{\Pi}$, which represents the desired solution to the correspondence problem.

The role of these two parameters in the matching algorithm is illustrated in the block diagram in Figure 6.6 (*c.f.* Section 6.3.2). Note that the last step of the SVD-based procedure (SVD3) was not executed when making the speed and accuracy measurements, since it does not contribute to the solution of the point correspondence problem.

Since the SVD-based algorithm is iterative, it is expected that the accuracy of the estimates for $(\widetilde{\Theta}, \widetilde{\Pi})$ depends on the values of l and m; therefore, results have been compiled for various combinations of values for l and m. The results given in Tables

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	269	269	270	569.8	568.2	587.4
0.10	254	271	268	615.1	592.0	588.8
0.25	244	278	274	590.6	582.0	585.0
0.50	282	294	254	589.6	569.0	579.0
1.00	284	312	308	599.3	568.4	567.9
2.00	296	264	372	569.7	590.6	571.6
3.00	323	374	515	589.6	580.3	586.6

Table 6.19: Results of SVD-Based algorithm for several noise variances and $l = 10, m = 25$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	195	264	225	1169.6	1173.2	1127.1
0.10	197	253	261	1142.1	1124.9	1120.6
0.25	242	204	285	1149.0	1136.1	1121.4
0.50	260	252	217	1095.5	1101.0	1112.0
1.00	257	251	244	1095.1	1146.0	1118.4
2.00	292	277	431	1122.7	1099.8	1101.8
3.00	262	277	516	1098.1	1100.0	1098.2

Table 6.20: Results of SVD-Based algorithm for several noise variances and $l = 10, m = 50$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	75	102	106	2188.7	2178.2	2158.3
0.10	108	157	78	2187.2	2181.3	2185.1
0.25	125	107	115	2180.5	2172.0	2175.9
0.50	105	100	113	2168.9	2171.9	2179.1
1.00	101	96	165	2173.2	2183.6	2178.9
2.00	164	137	329	2166.1	2181.8	2169.2
3.00	148	230	521	2187.3	2192.0	2185.6

Table 6.21: Results of SVD-Based algorithm for several noise variances and $l = 10, m = 100$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	201	203	202	581.6	583.5	586.6
0.10	197	188	170	587.3	580.4	588.8
0.25	156	178	233	580.9	602.3	609.4
0.50	195	201	175	599.3	592.8	582.3
1.00	192	223	264	610.2	594.5	584.9
2.00	167	266	329	597.7	580.6	575.7
3.00	237	297	518	584.5	579.9	574.8

Table 6.22: Results of SVD-Based algorithm for several noise variances and $l = 25, m = 25$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	119	107	144	1136.6	1124.5	1119.8
0.10	145	159	130	1120.2	1153.8	1119.5
0.25	154	161	131	1119.2	1130.9	1125.4
0.50	89	153	157	1117.6	1116.0	1120.1
1.00	139	148	176	1181.5	1113.3	1139.6
2.00	139	184	329	1124.0	1118.2	1127.7
3.00	185	278	508	1121.1	1124.5	1119.3

Table 6.23: Results of SVD-Based algorithm for several noise variances and $l = 25, m = 50$

p=10 q=10	# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
Noise σ^2	"Object" Size			"Object" Size		
	100^3	50^3	25^3	100^3	50^3	25^3
0.00	120	99	134	2198.6	2210.5	2237.1
0.10	86	107	102	2209.3	2210.8	2182.7
0.25	87	112	131	2218.7	2199.9	2192.2
0.50	128	134	99	2224.2	2179.8	2187.3
1.00	115	119	139	2196.3	2199.0	2194.2
2.00	123	158	300	2180.7	2182.4	2188.2
3.00	146	195	495	2189.7	2184.2	2207.1

Table 6.24: Results of SVD-Based algorithm for several noise variances and $l = 25, m = 100$

6.19 – 6.24 illustrate the performance of the algorithm, in terms of speed and accuracy, for several noise variances and enclosing volumes. The following observations can be made from these results:

- The computational requirements of the algorithm are determined solely by the number of iterations performed in SVD1 and SVD2 (*i.e.* the values of l and m); consequently, the noise variance and object size are not crucial factors, as in the case of the graph matching algorithm. However, given an "acceptable" level of least-squares error in matching, both the noise and the object size can have an effect on the number of iterations that are required by the SVD-based procedure.
- The accuracy of the algorithm is not strongly affected by variations in noise, for variances ranging between 0.0 and 3.0.
- Increasing the number of iterations in SVD1 and SVD2 (especially m) helps to improve the accuracy of the algorithm; however, the price paid, in terms of computational requirements, is quite high.

Comparing the results in Tables 6.6 – 6.7 with those in Tables 6.19 – 6.24, it is clear that the accuracy of the graph matching algorithm is far superior to that of the SVD-based algorithm, in all cases. This is largely due to the fact that the SVD-based procedure can converge to any one of several equivalued local extrema. Depending on which of these extrema the algorithm converges to, the final correspondences obtained may or may not be correct; although, the overall least squares error of the final solution is the same in all cases. Also, the processing of the permutation matrix that was used to determine whether or not the correspondences were "correct" may have contributed to some of the observed errors.

The final solution to the point correspondence problem is generally obtained much more quickly using the graph matching algorithm than the SVD-based method, as shown in Tables 6.6 – 6.7 and Tables 6.19 – 6.24. As the mutual distances between

the feature points decrease (*i.e.*, the enclosing volume decreases) and the noise corrupting the data increases, the computational speed of the two algorithms becomes more comparable. Eventually, the SVD-based method is faster than the graph matching algorithm in finding the optimal solution; however, the size of the enclosing volume and the noise level required for the SVD-based method to overtake the graph matching algorithm render the solutions obtained with the SVD-based method virtually useless, in terms of accuracy.

As the number of feature points to be matched increases, the computational requirements of the SVD-based algorithm increase very rapidly, as shown in Tables 6.25 – 6.26. For the two enclosing volumes in Tables 6.25 – 6.26 and in Tables 6.10 – 6.11, the results indicate that the rate of increase in computational requirements as the number of feature points increases is greater for the SVD-based algorithm than for the graph matching algorithm. Clearly, the computational speed of the graph matching algorithm is far superior to that of the SVD-based algorithm. The same can be said regarding the accuracy of the two methods, as the number of feature points increases.

The results in Tables 6.25 – 6.26 also show that the sharpness of the increase in computational requirements is independent of both the noise variance and the object size; however, there is a limit on the magnitude of the noise variance beyond which the SVD-based algorithm breaks down; specifically, the magnitude of the additive noise must be less than one-half of the shortest distance between any two feature points in the template data set [54].

Matching of Point Sets With Different Cardinalities

When the point sets to be matched have different cardinalities, the SVD-based algorithm must be modified as described in Section 4.5. These modifications are described here once again, for convenience.

Noise σ^2 0.1	# incorrect matches in 10 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	11	21	1091.0	1109.0
20	76	49	7861.0	7777.0
30	137	169	29779.0	28814.0
40	317	221	62514.0	61703.0
50	352	357	120754.0	121516.0

Table 6.25: Results of the SVD-based algorithm using point sets of various sizes and $\sigma^2 = 0.1$

Noise σ^2 0.25	# incorrect matches in 10 simulations		Average cpu time (ms) for each simulation	
# feature pts	"Object" Size		"Object" Size	
	100^3	200^3	100^3	200^3
10	13	21	1103.0	1094.0
20	57	48	7810.0	7812.0
30	150	136	29227.0	29078.0
40	197	217	62265.0	61766.0
50	358	346	122341.0	120841.0

Table 6.26: Results of the SVD-based algorithm using point sets of various sizes and $\sigma^2 = 0.25$

In the case for which $p > q$, the goal is to minimize the function

$$h(\Theta, \Pi(_q^p)) = tr[\Pi^T(_p^q)\mathbf{D}_Q\Pi(_q^p)] - 2tr[\Theta^T\mathbf{X}\Pi(_q^p)\mathbf{Y}^T], \tag{6.6}$$

with respect to $(\Theta, \Pi(_q^p))$, where $\Pi(_q^p)$ is a $(p \times q)$-dimensional permutation matrix and $\mathbf{D}_Q = \text{diag}(\mathbf{X}^T\mathbf{X})$ is a p-dimensional matrix whose elements are the diagonal elements of the symmetric matrix $\mathbf{X}^T\mathbf{X}$ [54] [4]. The minimization of the function $h(\Theta, \Pi(_q^p))$ in 6.6 is only required in Step 2 of the SVD-based algorithm (SVD2); hence, SVD1 and SVD3 are the same as in the case for which $p = q$ and SVD2 is modified as follows:

1. The expression for $\mathbf{A}_k$ in (4.35) is replaced by

$$\mathbf{A}_k(_q^p) = \mathbf{D}_Q\Pi_k(_q^p) - 2\sum_{i=1}^{n}\mathbf{D}_{\tilde{Z}_i^*}\Pi_k(_q^p)\mathbf{D}_{y_i},$$

2. The *negative* definite inverse square root matrix $[\mathbf{A}_k^T\mathbf{A}_k]^{-1/2}$ is computed, such that $h(\Theta, \Pi(_q^p))$ converges to its minimum point.

The negative definite inverse square root of the matrix $\mathbf{A}_k^T\mathbf{A}_k$ is obtained by computing the singular value decomposition of $\mathbf{A}_k^T\mathbf{A}_k$, which is given in 4.27, and using the orthogonal matrix $\mathbf{V}$ and the diagonal matrix $\mathbf{\Sigma}^{-1/2}$ in the following equation:

$$(\mathbf{A_k}^T\mathbf{A_k})^{-1/2} = \mathbf{V}(-\mathbf{\Sigma})^{-1/2}\mathbf{V}^T \tag{6.7}$$

The performance of the SVD-based algorithm in the case of unmatched points was evaluated through computer simulations, under the following conditions:

- sensed data sets contained $p = 30$ feature points,
- template data sets contained $q = 5, 15, 25$ feature points,
- all of the feature points were generated within an enclosing volume of 100^3,
- the sensed data sets were obtained from the template feature points as described in Section 6.2.2, subject to the following motion parameters: $\theta = 0.97$ rad, $[nx\ \ ny] = [0.23\ \ 0.44]$, and $[t_x\ \ t_y\ \ t_z] = [0\ \ 2\ \ -5]$, and

[4] A similar function is minimized in the case for which $p < q$.

p=30		# incorrect matches in 100 simulations			Average cpu time (ms) for each simulation		
"Object" Size	Noise σ^2	q			q		
		5	15	25	5	15	25
100^3	0.00	241	655	1247	2 847.3	15 631.7	37 678.6
	0.10	238	658	1275	2 833.5	16 748.1	36 379.8
	0.25	221	673	1265	2 833.8	14 208.2	35 763.3
	0.50	226	684	1199	2 871.2	13 278.9	35 727.2
	1.00	228	686	1231	2 826.6	14 260.8	35 629.5
	2.00	229	699	1279	2 849.1	13 257.2	35 526.2
	3.00	222	686	1214	2 829.6	13 353.0	35 599.8

Table 6.27: Results of the SVD-based algorithm in the case of unmatched points

- the number of recursions used in SVD1 and SVD2 were $l = 25$ and $m = 100$, respectively.

The results of the computer simulations are shown in Table 6.27. These results indicate that:

- The computational complexity of Step 2 of the SVD-based algorithm (SVD2) is not dominated by the size of the larger data set, when $p \neq q$; this is illustrated by the much smaller cpu requirements for $q = 5$ and $q = 15$ (where $q \ll p$), as compared with those for $q = 25$ (where $q \approx p$)
- The accuracy of the SVD-based algorithm is poor when unmatched feature points exist in the data sets to be matched.

The first observation regarding Table 6.27 is to be expected, since the matrix operations that are required in SVD2 involve $(p \times q)$-, $(p \times p)$-, and $(q \times p)-$ dimensional matrices; hence, the number of feature points q in the template data set have a significant impact on the total number of computations that are required in Step 2 of the algorithm. Most importantly, the inverse square root matrix $(\mathbf{A}_k^T \mathbf{A}_k)^{-1/2}$ which is computed for each iteration $k = 1, 2, \ldots, m$ of SVD2 requires the SVD of a $(q \times q)$-

dimensional matrix. This computation accounts for a large part of the total number of computations in SVD2. The observation regarding the accuracy of the the algorithm when $p \neq q$ is explained by the fact that the SVD-based procedure can converge to any one of $2^p p!$ local minima of the function $h(\Theta, \Pi_q^p)$, which may contain any number of correct or incorrect correspondences. In the case where $p \neq q$, the recursive procedure in Step 2 of the SVD-based algorithm seems to converge less frequently to the expected solution than in the case where $p = q$; nonetheless, the estimated permutation matrix Π^* is always a partial isometry matrix and is, indeed, a solution to the function optimization problem.

Once again, the results obtained with the SVD-based algorithm are inferior to those obtained with the graph matching algorithm. Comparing the results in Table 6.27 with those in Table 6.18, it is clear that the performance of the graph matching algorithm, in terms of both the computational speed in finding the solution and of the accuracy of the estimated solution, is far superior to that of the SVD-based algorithm in the case of unmatched points.

Chapter 7

Conclusions

7.1 Summary of the Matching Algorithms

In this thesis, we have compared two algorithms for solving various point pattern matching problems. The patterns involved in the matching process were assumed to contain feature points in the Euclidean n-space $\mathbb{E}^n$; although, specific attention was placed on the case of 3-dimensional feature points. Also, a global geometric transformation based on rigid motions was assumed to be sufficient for aligning the patterns, once the point correspondences were all known.

The focus in this study was mainly placed on solving the point correspondence problem, since it is known to be a highly complex part of the matching process. Two specific cases were considered. In the first case, the patterns contained an equal number of feature points and no extra or occluded points were allowed in the data sets. In the second case, one of the two patterns contained some extra or occluded points for which no correspondence could be established with the points in the other pattern. Computer simulations were conducted to compare the efficiency of the two algorithms in both cases.

7.1.1 The SVD-Based Procedure

The first algorithm that was reviewed in this thesis is the SVD-based method by Morgera and Lie Chin Cheong [2, 54]. This method is a function optimization approach for solving both the motion estimation problem and the point correspondence problem, and is based on the theory of Lie groups and Lie algebras. For the general unordered point matching problem, the solution is obtained by jointly optimizing two functionals: one in terms of Θ, an orthogonal rotation matrix, and the other in terms of Π, a permutation matrix. This is accomplished by following a three-step procedure. In the first step, an initial estimate of Θ (or Π) is obtained; this initial estimate is used in the second step, to obtain the optimal estimate for Π (or Θ); the third step is used to refine the initial estimate obtained in step 1.

To implement this procedure, the singular value decomposition of a matrix having the form $(\mathbf{A}^{\mathbf{T}}\mathbf{A})^{-1/2}$ is used at each step. This SVD-based implementation of the 3-step procedure is executed in an iterative fashion and can converge to any one of several equivalued local extrema. The number of possible solutions to the point correspondence problem (*i.e.*, step 2 of the 3-step procedure) is given by $2^p p!$, where p is the number of point features being matched; consequently, the probability of obtaining incorrect matches rapidly increases as the cardinality of the point patterns increases. This observation is supported by the simulation results presented in Chapter 6. The specific extremum to which the algorithm converges is somewhat dependent on the choice of the initial solution; nevertheless, the initial guess does not need to be close to the true solution for the algorithm to converge.

7.1.2 The Tree-Based Algorithm

The second algorithm that is reviewed in the thesis is the graph matching algorithm of Cheng and Don [1]. This method is a graph theoretic approach for solving the

point correspondence problem. The graph-based formulation makes it possible to include auxiliary information, such as the relationships between features, in the matching process. This attribute of the graph matching algorithm was neither exploited nor examined in this study for purposes of comparison with the SVD-based algorithm.

The graph matching algorithm differs from the SVD-based method in two important respects. First, the graph-based approach uses a backtracking tree search procedure as the search strategy. In the SVD-based method, the search strategy consists of straightforward iterative computations which are aimed at minimizing the similarity metric. The backtracking approach uses the similarity metric in the search strategy to guide the search for the optimal solution to the correspondence problem. Specifically, the k-th order error is incorporated into the two pruning mechanisms that are used to reduce the number of paths that need to be traversed in the search procedure. The first mechanism is the so-called edge threshold, which exploits the principle of invariance of distance measures under rigid body motion. The second pruning mechanism is the so-called graph-threshold, which ensures that the error accumulated in traversing new paths decreases monotonically. For each candidate solution, the accumulated kth order error is computed, and a lower-error solution is sought by backtracking through the tree and traversing unexplored paths. The final solution is the candidate matching graph which has the smallest kth order error. The differences in the similarity metrics that are employed by the two algorithms is the second important distinguishing factor; namely, the k-th order error, in the case of the graph matching algorithm, and the least squares error, in the case of the SVD-based method.

7.1.3 Simulation Results

Computer simulations were performed to illustrate the differences in the computational requirements of the two algorithms in finding a solution to the point correspondence problem. Two distinct types of results were obtained from the simulations. The first

type of results dealt with the dynamic behavior of the algorithms. In this case, experiments were conducted to observe the progress of each algorithm from an initial solution, through several intermediate results, to the final solution. The second type of simulation results involved direct measurements on the speed and accuracy of the algorithms, for specific matching problems. In all of the experiments, the effects of both noise and the density of feature points within an enclosing volume on the computational requirements of the algorithms were considered.

The simulation results in the algorithm dynamics section illustrate the differences in the search strategies that are employed by the matching algorithms; moreover, these results provide insight which helps to interpret the simulation results on computational speed and accuracy. In the case of the graph matching algorithm, the solution to the point correspondence problem is obtained both quickly and accurately when the distances between the feature points are largely dissimilar. That is a consequence of the distance constraint which is used as a pruning mechanism in the search procedure. As the similarity in the inter-point distances between the feature points in a data set increases, the effectiveness of the pruning mechanisms is weakened; accordingly, a greater number of partial matches is generated by the algorithm, in this case. This effect has a direct impact on the computational requirements of the graph matching procedure, as illustrated by the results on computational speed and accuracy. In particular, it is shown that the computational requirements of the algorithm significantly increase as the density of feature points within an enclosing volume increases. Equivalently, for a fixed number of feature points and a fixed level of additive noise, the cpu time required to find the optimal solution significantly increases as the enclosing volume in which the feature points are (randomly) generated is reduced. The addition of Gaussian noise to the feature points has the effect of increasing the similarity in the inter-point distances between the feature points; hence, as the noise variance is increased, the effect of reducing the enclosing volume in which the feature points are generated is greatly accentuated. This effect is not observed in the case of the SVD-based procedure, due

to the fundamentally different search strategy that is used in this method. The computational requirements of the SVD-based algorithm are entirely determined by the number of iterations l and m that are used in the first two steps of the procedure, SVD1 and SVD2, respectively. Still, the graph matching algorithm is shown to be much faster than the SVD-based procedure in finding the optimal solution, in all of the cases attempted; however, the effects of increasing the additive noise limit this advantage in computational speed. The accuracy of the solutions obtained with the graph matching algorithm is also shown to be superior, in comparison with the SVD-based algorithm. The solutions obtained using the SVD-based procedure are seen to be somewhat dependent on the initial guesses $\widetilde{\Theta}(0)$ and $\widetilde{\Pi}(0)$ that are used in SVD1 and SVD2, respectively; consequently, the solutions given by the SVD-based algorithm are not as consistently accurate as those obtained by the graph matching algorithm.

7.2 Concluding Remarks

This study has revealed the strengths and weaknesses of two matching algorithms that are based on widely differing search strategies. The graph matching algorithm is found to be both very accurate and exceptionally efficient in obtaining the optimal solution to the point correspondence problem, under certain conditions. The most significant limitation of this algorithm is its sensitivity, in terms of computational speed, to both noise and the similarity of the inter-distances between the feature points in the data sets to be matched. The computational speed of the graph matching algorithm is largely dependent on the effectiveness of the pruning mechanisms in reducing the number of paths to be traversed in the search tree. When the distance-based pruning mechanisms cease to function effectively, the algorithm breaks down, from a computational standpoint. The simulation results have demonstrated that perturbations in the positions of the feature points and the density of feature points in a fixed enclosing volume have a compound effect on the ability of the tree search procedure to prune out invalid paths.

The SVD-based algorithm, on the other hand, is much more robust in the presence of noise and is not affected by the distribution of inter-distances between the feature points. However, the solution obtained by the SVD-based procedure is not as consistently accurate as the one obtained by the graph matching algorithm, due to the multiplicity of local extrema to which the algorithm can converge. As the number of feature points to be matched increases, the number of potential solutions to which the SVD-based algorithm can converge, albeit equivalued in terms of least squares error, rises very rapidly. The computational speed of the SVD-based procedure is clearly inferior to that of the graph matching algorithm, as demonstrated by the simulation results, except in cases when the previously mentioned noise and distance factors cause the graph matching algorithm to break down.

7.3 Future Research

Given that the robustness of a matching algorithm in the presence of both noisy data and occluded or missing points is an important consideration, future research involving tree search algorithms should concentrate on the efficiency of the pruning mechanisms. In the case of Cheng and Don's algorithm [1], the pruning mechanisms are simple and are, in fact, efficient for matching data sets containing a small number of feature points. It may be possible to increase the number of points that can be efficiently accommodated by the graph matching algorithm, under most practical conditions, by enhancing the existing pruning mechanisms. This might be done by including additional geometric constraints in the search strategy, such as those used by Chen and Huang [47]. Another possibility would be to combine mechanisms such as "looking-ahead" [44] with the geometric constraints in the tree search. The effectiveness of using *a priori* information in the graph matching procedure to guide the search should also be investigated.

Regarding the SVD-based procedure of Morgera and Lie Chin Cheong [2], the computational speed of the algorithm might be improved by using the hybrid SVD/Steepest-Ascent approach that is described in [54]. There may also be considerable merit in investigating the use of other techniques for maximizing/minimizing the functionals that are described in Section 4.3. Also, the effect of statistical outliers on the performance of the SVD-based algorithm was considered to be beyond the scope of this study and was not investigated; consequently, the robustness of the algorithm in the presence of one or more severely corrupted or mismatched feature points remains to be tested.

Bibliography

[1] J.C. Cheng and H.S. Don. A Graph Matching Approach to 3-D Point Correspondences. *Intern. Journ. Pattern Recognition and Artificial Intelligence*, 5:399–412, 1991.

[2] S.D. Morgera and P. Lie Chin Cheong. Rigid Body Constrained Noisy Point Pattern Matching. *IEEE Trans. Image Proc*, 4(5):630–641, May 1995.

[3] R.W. Brockett. Least Squares Matching Problems. *Linear Algebra and its Applications*, 122/123/124:761–777, September/October/November 1989.

[4] F. Arman and J.K. Aggarwal. Model-Based Object Recognition in Dense-Range Images-A Review. *ACM Computing Surveys*, 25(1):5–43, March 1993.

[5] Y.F. Wang, N. Karandikar, and J.K. Aggarwal. Analysis of Video Image Sequences Using Point and Line Correspondences. *Pattern Recognition*, 24(11):1065–1084, March 1991.

[6] A. Mitiche and J.K. Aggarwal. A Computational Analysis of Time Varying Images. In T.Y. Young and K.S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*. Academic, New York, 1986.

[7] J. Weng, T.S. Huang, and N. Ahuja. 3-D Motion Estimation, Understanding, and Prediction from Noisy Image Sequences. *IEEE Trans. Patt. Anal. Machine Intell.*, PAMI-9(3):370–389, 1987.

[8] L.G. Brown. A Survey of Image Registration Techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[9] D.N. Levin, C.A. Pelizzari, G.T. Chen, *et al.* Retrospective Geometric Correlation of MR, CT, and PET Images. *Radiology*, 169(3):817–823, December 1988.

[10] D.L. Hill, D.J. Hawkes *et al.* Registration of MR and CT Images for Skull Base Surgery Using Point-Like Anatomical Features. *British Journal of Radiology*, 64(767):1030–1035, November 1991.

[11] F. Murtagh. A New Approach to Point-Pattern Matching. *Publications of the Astronomical Society of the Pacific*, 104:301–307, April 1992.

[12] P.H Schönemann and R.M. Carroll. Fitting One Matrix To Another Under Choice of A Central Dilation and A Rigid Motion. *Psychometrika*, 35:245–255, 1970.

[13] A.N. Netravali and J.D. Robbins. Motion-Compensated Television Coding: Some New Results. *Bell Systems Technical Journal*, 59:1735–1745, 1980.

[14] J.R. Jain and A.K. Jain. Displacement Measurement and Its Application in Interframe Image Coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, December 1981.

[15] R. Lenz and A. Gerhard. Image Sequence Coding Using Scene Analysis and Spatio-Temporal Information. In T.S. Huang, editor, *Image Sequence Processing and Dynamic Scene Analysis*, NATO ASI, pages 264–274. Springer-Verlag, Berlin, 1983.

[16] E.H. Thompson. An Exact Linear Solution of the Problem of Absolute Orientation. *Photogrammetria*, 15(4):163–179, 1958.

[17] H.L. Oswal and S. Balasubrammanian. An Exact Solution of Absolute Orientation. *Photogrammetric Engineering*, 34:1079–1083, 1968.

[18] B.K.P. Horn. Closed-form Solution of Absolute Orientation Using Orthonormal Matrices. *Journ. Opt. Soc. Amer. A*, 5:1127–1135, 1987.

[19] C. Eckart and G. Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1:211–218, 1936.

[20] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-9:698–700, September 1987.

[21] S. Umeyama. Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. *IEEE Trans. Patt. Anal. Machine Intell.*, 13:376–380, April 1991.

[22] R.M. Endlich, D.E. Wolf, D.J. Hall, and A.E. Brain. Use of a Pattern Recognition Technique for Determining Cloud Motions from Sequences of Satellite Photographs. *J. Appl. Meteorol.*, 10:105–117, February 1971.

[23] J.A. Leese, C.S. Novak, and V.R. Taylor. An Automated Technique for Obtaining Cloud Motion from Geosynchronous Satellite Data using Cross-Correlation. *J. Appl. Meterol.*, 10:118–132, February 1971.

[24] F. Rocca and S. Zanoletti. Bandwidth Reduction Via Movement Compensation On A Model of the Random Video Process. *IEEE Trans. Commun*, COM-20:960–965, October 1972.

[25] J.O. Limb and J.A. Murphy. Measuring the Speed of Moving Objects From Television Signals. *IEEE Trans. Commun.*, COM-23:474–478, April 1975.

[26] J.W. Roach and J.K. Aggarwal. Determining the Movement of Objects From a Sequence of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(6):554–562, 1980.

[27] J. Weng, N. Ahuja, and T.S. Huang. Optimal Motion and Structure Estimation. *Trans. Patt. Anal. Machine Intell.*, 15(9), September 1993.

[28] T.J. Broida and R. Chellappa. Estimation of Object Motion Parameters from Noisy Images. *IEEE Trans. Patt. Anal. Machine Intell.*, PAMI-8:90–99, 1986.

[29] H. Shariat and K.E. Price. Motion Estimation with More Than Two Frames. *IEEE Trans. Patt. Anal. Machine Intell.*, 12(5):417–434, May 1990.

[30] S. Ullman. *The Interpretation of Structure From Motion*, chapter The Interpretation of Visual Motion, pages 145–170. MIT Press, Cambridge, MA., 1979.

[31] J.Q. Fang and T.S. Huang. Uniqueness and Estimation of 3-D Motion Parameters of Rigid Bodies with Curved Surfaces. *PAMI-6*, 1:13–27, 1984.

[32] T.S. Huang. Determining Three-Dimensionsal Motion and Structure from Two Perspective Views. In T.Y. Young and Eds. K.S. Fu, editors, *Handbook of Pattern Recognition and Image Processing.* Academic, New York, 1986.

[33] R.Y. Tsai and T.S. Huang. Estimating 3-D Motion Parameters of a Rigid Planar Patch. *IEEE Trans. ASSP*, 29(6):1147–1152, 1981.

[34] R.Y. Tsai and T.S. Huang. Uniqueness and Estimations of 3-D Motion Parameters of Rigid Bodies with Curved Surfaces. *IEEE Trans. on Patt. Anal, and Machine Intell*, PAMI-6(1):13–27, 1984.

[35] A. Goshtasby and G.C. Stockman. Point Pattern Matching Using Convex Hull Edges. *IEEE Transactions On Systems, Man and Cybernetics*, SMC-15(5):631–637, Sept/Oct 1985.

[36] H.C. Longuet-Higgins. A Computer Program for Reconstructing a Scene from Two Projections. *Nature (London)*, 293:133–135, 1981.

[37] A.R. Bruss and B.K.P. Horn. Passive Navigation. *Computer Vision, Graphics, and Image Processing*, 21(3):3–20, 1983.

[38] D.H. Ballard and O.A. Kimball. Rigid Body Motion from Depth and Optical Flow. *Computer Vision, Graphics, and Image Processing*, 22(1):95–115, 1983.

[39] G.A. Korn and T.M. Korn. *Mathematical Handbook for Scientists and Engineers.* McGraw Hill, New York, 1968.

[40] M.D. Shuster. Approximate Algorithms for Fast Optimal Attitude Computation. In *Proc. AIAA Guidance and Control Conf.*, pages 88–95, Palo Alto, CA, August 1978.

[41] O.D. Faugeras and M. Hebert. The Representation, Recognition, and Locating of 3-D Objects. *Int. J. Robotics Research*, 5(3):27–52, 1986.

[42] B.K.P. Horn. Closed-Form Solution of Absolute Orientation Using Unit Quaternions. *Journ. Opt. Soc. Amer. A*, 4:629–642, April 1987.

[43] W.E.L. Grimson and T. Lozano-Perez. Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. Patt. Anal. Mach. Intell.*, PAMI-9(4):469–482, 1987.

[44] L.G. Shapiro and R.M. Haralick. Structural Descriptions and Inexact Matching. *IEEE Trans. Patt. Anal. Mach. Intell.*, PAMI-3(5):504–519, 1981.

[45] N. Ansari, M-H Chen, and E.S.H. Hou. Point Pattern Matching By A Genetic Algorithm. In *IECON '90: 16th Annual Conference of IEEE Industrial Electronics Society*, volume II, pages 1233–1238, Pacific Grove, California, November 1990.

[46] M. Herman. Matching Three-Dimensional Symbolic Descriptions Obtained from Multiple Views of a Scene. *Proc. Computer Vision Pattern Recognition*, pages 585–590, 1985.

[47] H.H. Chen and T.S. Huang. Maximal Matching of 3-D Points for Multiple-Object Motion Estimation. *Pattern Recognition*, 21:75–90, 1988.

[48] E. Gmür and H. Bunke. 3-D Object Recognition Based on Subgraph Matching in Polynomial Time. In T. Pavlidis R. Mohr and A. Sanfeliu, editors, *Structural Pattern Analysis*, volume 19, pages 131–147. World Scientific, 1990.

[49] K.S. Fu A. Sanfeliu. A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, 1983.

[50] E.K. Wong and K.S. Fu. A Graph-Theoretic Approach to 3-D Object Recognition and Estimation of Position and Orientation. In J.T. Tou, editor, *Computer Based Automation*, pages 305–343, New York and London, 1985. Plenum Press.

[51] A.K. Wong and M. You. Entropy and Distance of Random Graphs with Application to Structural Pattern Recognition. *IEEE Trans. Patt. Anal. Machine Intell.*, PAMI-7(5):599–609, September 1985.

[52] Z-C. Lin, T.S. Huang, S.D. Blostein, H. Lee, and E.A. Margerum. Motion Estimation from 3-D Point Sets With and Without Correspondences; Part I. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 194–198 (Part I), 198–201 (Part II), Miami Beach, FL, June 1986.

[53] P.J. Besl and N.D. McKay. A Method for Registration of 3-D Shapes. *IEEE Trans. Patt. Anal. Machine Intell.*, 14(2):239–256, 1992.

[54] P. Lie Chin Cheong. *Geometrically Constrained Matching Schemes.* PhD thesis, McGill University, Montreal, Quebec, Canada, May 1992.

[55] T.L. Faber, R.W. McColl, R.M. Opperman et al. Spatial and Temporal Registration of Cardiac SPECT and MR Images: Methods and Evaluation. *Radiology*, 179(3):857–861, 1991.

[56] T.L. Boullion and P.L. Odell. *Generalized Inverse Matrices.* Wiley-Interscience, 1971.

[57] R.A. van den Elsen, E.D. Pol, and M.A. Viergever. Medical Image Matching–A Review with Classification. *IEEE Engineering in Medicine and Biology*, 12(1):26–39, March 1993.

[58] R.A. Jarvis. A Perspective on Range Finding Techniques for Computer Vision. *IEEE Trans. Patt. Anal. Mach. Intell.*, 5(2):122–139, 1983.

[59] P.J. Besl. *Advances in Machine Vision*, chapter Active optical range imaging. Springer-Verlag, New York, NY, 1989.

[60] D.I. Barnea and H.F. Silverman. A Class of Algorithms for Fast Digital Registration. *IEEE Trans. Comput.*, C-21:179–186, 1972.

[61] S. Chaudhuri and S. Chatterjee. Robust Estimation of 3-D Motion Parameters in the Presence of Correspondence Mismatches. In *Conference Record of the Twenty-fifth Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1195–1199, Los Alamitos, California, November 1991.

[62] D. Mintz, P. Meer, A. Rosenfeld. Analysis of the Least Median of Squares Estimator for Computer Vision Applications. In *1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 621–623, Los Alamitos, California, June 1992.

[63] W.K. Pratt. *Digital Image Processing.* Wiley and Sons, New York, 1978.

[64] E. De Castro and C. Morandi. Registration of Translated and Rotated Images Using Finite Fourier Transforms. *IEEE Trans. Pattern Anal. Machine Intell*, PAMI-9:700–703, 1987.

[65] V. Salari and I.K. Sethi. Feature Point Correspondence in the Presence of Occlusion. *IEEE Trans. Patt. Anal. Machine Intell.*, 12(1):87–91, January 1990.

[66] J.G.F. Belinfante and B. Kolman. *A Survey of Lie Groups and Lie Algebras with Applications in Computational Methods.* SIAM, Philadelphia, 1972.

[67] S. Haykin. *Adaptive Filter Theory.* Prentice Hall Inc., New Jersey, 2 edition, 1991.

[68] G.W. Stewart. *Introduction to Matrix Computations.* Academic Press, New York, 1973.

[69] D.E. Knuth. *The Art of Computer Programming,* volume 1. Addison Wesley, 1968.

[70] A. Tucker. *Applied Combinatorics.* John Wiley and Sons, New York, 2 edition, 1984.

[71] T. Gaskins. *PHIGS Programming Manual.* O'Reilly and Associates, Inc., Sebastopol, California, 1992.

Appendix A

Graph Matching Algorithm

Cheng and Don's graph matching algorithm is given as follows [1]:

Notation:

$E_1 = \{\mathbf{y}_i : i = 1, 2, \ldots, p\}$: template point set.

$E_2 = \{\mathbf{x}_i : i = 1, 2, \ldots, p\}$: sensed point set.

$T = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_p)$: template graph.

$\tilde{T} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)$: working graph whose vertices are found from E_2.

$\hat{T} = (\mathbf{x}_{\pi(1)}, \mathbf{x}_{\pi(2)}, \ldots, \mathbf{x}_{\pi(p)})$: optimal solution graph. Vertex $\mathbf{x}_{\pi(i)}$ corresponds to the i-th vertex of T.

σ = standard deviation of the zero-mean Gaussian noise added to the point features.

Initialization:

$\hat{T}$ is empty

edge threshold $\epsilon = 8\sqrt{(3)}\sigma$

graph threshold $\delta = p(p-1)/\epsilon/2$

Algorithm:

Begin

1. Calculate the distances $d(\mathbf{y}_i, \mathbf{y}_j)$ and $d(\mathbf{x}_i, \mathbf{x}_j)$, $i \geq 1, j \leq p$.

2. **For** $i = 1$ **to** $p - 1$ **do**
3. **For** $j = i + 1$ **to** $p - 1$ **do** steps 4–6.

Begin

4. **If** $|d(\mathbf{x}_i, \mathbf{x}_j) - d(\mathbf{y}_1, \mathbf{y}_2)| < \epsilon$ **then do** steps 5–6.

Begin

5. $\tilde{\mathbf{x}}_1 \rightarrow \mathbf{x}_i$, $\tilde{\mathbf{x}}_2 \rightarrow \mathbf{x}_j$, **Next(2).**
6. $\tilde{\mathbf{x}}_1 \rightarrow \mathbf{x}_j$, $\tilde{\mathbf{x}}_2 \rightarrow \mathbf{x}_i$, **Next(2).**

End

End **End**

Procedure **Next**(i) (i is the number of vertices in the working graph):

Begin

1. **If** $i = p$ **then do** steps 2–3.

Begin

2. $\hat{T} \rightarrow \tilde{T}$
3. $\delta \rightarrow k$-th order error of $\hat{T}$

End

4. **Else do** steps 5–8.

Begin

5. **For** $l = 1$ **to** p **do**

If $\mathbf{x}_l$ not in $\tilde{T}$, **then do** steps 6–8

Begin

6. **If** $|d(\mathbf{x}_l, \tilde{\mathbf{x}}_{i-j}) - d(\mathbf{y}_{i+1}, \mathbf{y}_{i-j})| < \epsilon$ and accumulated error $\alpha < \delta$,

for all $0 \leq j \leq min(k, i) - 1$ **then do** steps 7–8

Begin

7. $\tilde{\mathbf{x}}_{i+1} \rightarrow \mathbf{x}_l$
8. **Next**$(i + 1)$

End

End

End

End