DISSERTATION

TWO DIMENSIONAL PROJECTIVE POINT MATCHING

Submitted by

Jason Denton

Computer Science

In partial fulfillment of the requirements

for the Degree of Ph.D.

Colorado State University

Fort Collins, Colorado

Summer 2002

COLORADO STATE UNIVERSITY

July 1st, 2002

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY JASON DENTON ENTITLED TWO DIMENSIONAL PROJECTIVE POINT MATCHING BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF PH.D..

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

TWO DIMENSIONAL PROJECTIVE POINT MATCHING

Point matching is a problem which occurs in several forms in computer vision and other areas. Solving practical point matching problems requires that a point matching algorithm allow for an appropriate class of geometric transformations between the points in the model and their instance in the data. Many real world point matching problems require a two dimensional projective transformation to relate the model to the data. Point matching under this class of transformations has received little attention, and existing algorithms are inadequate. Existing, general, polynomial time point matching algorithms by Baird, Cass, and Barrel are formulated for lower order transformation classes and have difficulty scaling. The RANSAC algorithm, which represents the current best solution to the problem under the projective transformation, can not solve the problem when there are significant amounts of noise and clutter in the data sets; a condition likely to occur in many real problem instances.

Presented here is a new algorithm for point matching based on local search. This algorithm is a general solution to the two dimensional point matching problem under all transformation classes; although the focus is on the projective transform case. This algorithm gracefully deals with more clutter and noise than the existing algorithms, while still providing an efficient solution to easier problem instances. A randomized version of the algorithm is presented, and a superior version which uses a

key feature algorithm to identify partial matches which may be a part of the optimal solution is detailed. The effectiveness of these algorithms is validated for image registration and model recognition problems using data obtained from real imagery; point sets of various sizes containing varying amounts of noise and clutter.

Jason Denton
Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 2002

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 The Need for Point Matching

Consider the problem of matching points in two or more dimensions. Given two sets of points, there is a need to determine which points in the first set correspond to points in the second set. In the realm of computer vision points often arise as corners or other features of images and are found by some feature extraction algorithm. Point matching is then employed to provide a meaningful interpretation of this data. Usually this is an attempt to solve one of two problem. Image registration (24) requires the correspondence between points be determined so that the geometric relationship between two images can be determined. Model recognition (22) requires that some correspondence between a point set representing the object model and a subset of the data point set be found, along with the geometric pose of the model within the data. Uses for point matching also exist outside of the realm of computer imagery. Localization of DNA markers (29), finger print analysis (12), and sonar processing (26) are all domains where point matching is an essential and difficult task.

If the two sets of points differ only by a fixed, rigid geometric transformation, such as a rotation and translation, then the problem is not difficult. The problem becomes more difficult when a larger class of transformations is allowed; for example, scaling, skewing or foreshortening. Once the complexities of the allowed transformation class have been dealt with there are other difficulties. In many problem domains the exact positions of points in at least one of the point sets can not be known. Algorithms must account for this uncertainty. Sensor errors may cause some points of interest to be missed, or unexpected objects in the scene may introduce points into one of the sets that are not in the other. These missing or spurious points cause further computational difficulties.

To be practical, a point matching algorithm must deal with these complexities. It must accurately find matches between subsets of points in both the model and the data, ignoring spurious points in the data and gracefully allowing for some model points to go unmatched because they have no corresponding point in the data. A good algorithm must also deal with the uncertainty in point locations. When the class of allowed transformations is limited there exist algorithms which provide high order polynomial time solutions. Published algorithms generally address the two-dimensional rigid body problem (11; 24), allowing model and data to be related by translation and rotation. Many algorithms provide extensions to allow similarity transforms (3; 20). A few deal with affine transformations (7). All these cases deal with two-dimensional motion of two dimensional points. Unfortunately, this is not sufficient for some real world problem domains.

Consider ariel photography. Points in such an image may not be truly two dimensional, but two dimensional points provide a practical and reasonable approximation to the truth. If two photos of the same general area are taken, perhaps at different times or different altitudes, or by planes on different headings, the resulting images will be related by a three dimensional transformation. Star maps present a similar difficultly, two maps may represent views of the sky from vastly different locations. If a set of correspondences between points can be determined, then the geometric relationship between the two image may be solved and a new, larger, more accurate image or map produced. Both of these problems require a projective transformation to relate one set of two dimensional points to the other by a three dimensional camera movement.

The projective transformation class presents difficulties not seen in other two-dimensional transformation classes. The projective transform is non-linear, and has eight degrees of freedom. This raises the size and complexity of pose space dramatically. Previous algorithms quickly encounter efficiency problems when scaled up to deal with the projective case. This failing explains, in part, the relative scarcity of prior work on general projective matching problems. To date the best existing algorithm for solving projective point matching problems appears to be a variant of the RANSAC algorithm (19), which has difficulty solving problems which contain noise and clutter.

1.2 Local Search Point Matching

This dissertation examines the use of local search to solve the point matching problem, with an emphasis on point matching under the projective transformation. Local search is a straight forward search algorithm which requires two things. First, it must be possible to evaluate each possible solution such that two solutions can be compared to each other. In the context of point matching this means that there must be an objective function that can evaluate a set of correspondences between points. This objective function measures the distance between paired points under some optimal geometric transformation, and penalizes those instances where the optimal transformation is unlikely or physically impossible. Evaluation of the objective function for different correspondences presupposes that an optimal pose can be found; the geometric position of the model within the data such that the distance between paired points is minimized. This globally optimal pose determination relative to a fixed correspondence between points actually serves to guide the search through correspondence space. The fine-grained interplay between search in the discrete space of correspondences and the determination of the optimal pose leads us to call this approach pose directed search. Second, local search requires that for any given solution a neighborhood of adjacent solutions be defined. This neighborhood defines the search space to be examined for both the next step in the search and for the algorithm as a whole. The algorithm examined here defines the neighborhood to be all those solutions which differ by only one pairing. Local search begins with an initial solution and then examines the entire neighborhood for an improvement on this solution. Local search moves to the best solution found and repeats the process until no further improvements are impossible given the neighborhood definition.

The choice of an initial solution at which to start the local search algorithm is of great importance. If an initial solution is chosen at random then it is unlikely that the resulting optima will be the globally optimal solution. However, when each random initial solution is independent, it can be shown that the chance of failure is an exponentially decreasing function of the number of trials run (25). Because each trial of local search can be run quickly, running a large number of trials can be an effective search strategy. Alternatively, initial solutions can be chosen in a fashion which makes them more likely to be in the basin of attraction for the global optima. The approach taken here

is to construct key features from an original point set by grouping those points which are spatially proximal. The points making up these key features are then paired to create a partial match which is used to initialize local search. The algorithm presented here generates a large number of key feature based on partial solutions with a minimal computational effort, and then ranks them using the same objective function used by the search algorithm. Often the top ranked solution lies in the basin of attraction for the global optima, and many point matching problems can be solved by running trials from only a small portion of the heuristically generated starting points. However, this is not always the case, as will be shown by example later.

1.3 Overview

Prior work has used a wide range of terminology and notation, and so prior to discussing the existing literature in the field a consistent set of terms and definitions will be set out. This will facilitate discussion of those algorithms which are noteworthy from the prior literature. The emphasis will be on those papers and algorithms which provide efficient solutions to certain classes of point matching problems. The strengths and weaknesses of these algorithms will be assessed, particularly with respect to which problems can and can not be solved. Other work will also be of interest, in some cases authors have made important theoretical advances without providing effective algorithms or have provided algorithms which have since been superseded by later work. A few novel approaches to point matching also exist, and these are of interest more because they show that there are many ways in which the problem may be formulated than for the contribution made towards solving the problem.

The general problem will then be examined in more depth. There are several dimensions to point matching which effect the formulation of the problem and its solution, and these are laid out and examined. Two problem domains from computer vision, image registration and model recognition, will be introduced. These will be the focus of the experiments done in this study. These two problem domains are similar to point matching problems found in other areas and an algorithm which can solve problems from both domains may be expected to generalize well beyond them. The class of allowed transformations relating two points sets is a major differentiating factor between point matching algorithms, and the four most commonly considered transformation classes will be described. These

are: translation and rotation, similarity, affine, and projective. Their characteristic equations will be described, and the meanings of the relevant variables with respect to real world geometry will be discussed.

As just discussed, the use of local search for point matching requires that point matching be stated as a combinatorial optimization problem. The objective function used by local search is presented in detail, with a discussion of the various components. Key to the use of this function is the determination of an optimal pose for the set of correspondences in question. The optimal pose is defined as the pose which minimizes the squared distance between paired points, and a least squares approach is taken to find this pose. In the case of similarity and affine transformations there are a set of linear equations which can be solved to find the optimal pose and these equations are derived. The case of the projective transformation is somewhat more difficult because there is not a linear relationship between the original and transformed position of the points. The conventional method for finding the optimal transformation in this case is presented, as is a faster alternative. Because this formulation of the problem differs significantly from the approach taken in the existing literature its relationship to the bounded error formulation more commonly used is explored.

In order to make a meaningful comparisons between algorithms several key algorithms are presented in sufficient detail to allow their implementation. Pose equivalence by Cass (11) and RAST by Breuel (7) are presented, mainly for theoretical interest. The random starts local search algorithm is presented in detail, as is the heuristic starts variant. The key feature algorithm is explained and examples are provided, along with experimental observations about it use.

Many previous researchers have validated their algorithms on synthetic data (3; 7), but this practice can make it difficult to predict how an algorithm may perform on real data. Data for this study was obtained directly from real imagery. Most of this imagery was taken specifically for this study, but some was obtained from other sources. This data is presented, organized by the type of experiment the data in question was used to run.

Finally, experimental results for the local search algorithm are presented. These are divided by problem domain, the primary areas of interest being the use of point matching for image registration and for model recognition. Detailed results of experiments in each domain are presented, in order to answer key questions about the use of local search for point matching. Local search is shown to be

effective at solving a variety of problems and the power of the heuristic starts algorithm over random starts is clearly demonstrated. A direct comparison of local search and RANSAC (19) is provided and an explanation for the superior ability of local search to solve point matching problems under real world conditions is offered. Although the focus is on the use of local search to solve projective point matching problems, local search is also shown to be effective at solving problems from more limited transformation classes. This makes it possible to directly compare local search to existing algorithms, and a comparison with the RAST (8) algorithm shows local search to be competitive for these problems.

The study concludes with an overview of the results and contribution of the work. Where there are open questions these are presented with an emphasis on those areas which hold promise for further research.

Chapter 2

Prior Work and Existing Literature

2.1 Introduction

A large body of literature addresses point matching (12; 13; 21). Previous researchers have generally addressed point matching as a necessary step in larger real world problems (1; 17; 26). Consequently, there exist a number of algorithms which are adequate for solving specific variations of the problem. Often these solutions do not generalize well beyond their original problem domain.

Other researchers have addressed the general problem without the introduction of domain specific constraints (3; 7; 11). Out of this latter work several algorithms which provide polynomial time solutions have been developed, provided that certain assumptions are met. These constraints often take the form of limitations on the class of allowable transformations which related the point sets being matched (12; 26). Algorithms which constrain the composition of the point sets can also be found (20).

Baird's tree search is particularly notable because it cast point matching as constraint satisfaction and demonstrated the effective use of polygonal error bounds. Pose equivalence by Cass (11) builds on this idea by recognizing that there are a finite number of poses that give rise to distinct matches under a bounded error test. Breuel's RAST algorithm (7) provides an efficient solution for a larger class of point matching problems, and Hong and Tan's canonical form algorithm (20) presents an efficient and novel approach to point matching in the absence of clutter. Early work in the field of point matching relied on various generalized hough transform methods (12; 17; 26). Simulated annealing (29) and neural networks (31) have been used to solve point matching problems with limited success. To date, the best algorithm for point matching under the projective transformation appears

to be RANSAC (19). Finally, the line matching work by Beveridge provides the first instance of pose directed local search (5), and this work is a natural extension of that work into a higher dimensional transformation space and using a more generic feature: points rather than line segments.

2.1.1 Definitions

To facilitate discussion, a consistent set of notation and terminology is presented here.

Point

A *point* is an ordered pair of real numbers: the first number giving the location of the point along the x axis and the second number giving the location along the y axis. Only points of two dimensions are considered in this thesis, although they are often represented as three place vectors using homogeneous coordinates.

Point Set

A *point set* is an un-ordered list of points. References are often made to the *model* and *data* sets; which are the point sets which define a given problem instance. The model set is denoted by M, the data set by D. The size of these sets is denoted by m and d, respectively. Individual points within these data sets are denoted as m_i and d_i. Depending on the problem, the model set may be an *object model* representing the points of interest on a particular object. The total number of potential pairings may be of use at times, and this value is given as $n = md$.

Through out this work there is a distinction between model and data sets, and the point matching problem is formulated as the problem of finding a match for the model points in the data set. However, in many problem domains this choice is arbitrary and the distinction not meaningful. When discussing image registration problems in particular, every problem of matching one set to another has an inverse problem where the roles of the model and data sets are reversed.

Noise and Error

Noise or *point wise error* is the difference between the observed location of a data point and the true location of that point. Error in the location of data points is to be expected in any practical application, and so it is important that any point matching algorithm be able to deal with it.

Error Bounds describe how much error can be expected in any given point measurement and usually take one of two forms. Some researchers (2) have expressed these bounds as polygons, usually with a small number of sides. Under certain problem formulations this helps to reduce the computational complexity. Alternatively, circular error bounds often provide a better means of representing pointwise error. In this work error bounds are taken to be implicitly circular.

Model Pose

Model pose refers to a transformation on the model intended to bring it into alignment with the data. Generally this term refers to the optimal transformation which aligns a model with a data set, with minimal error as defined by some error function.

Match Set or Correspondence

A *match set* is a mapping between two point sets, such that a point in one set is paired with one or more points in the other. *Complete match sets* map every point in the model to at least one point in the data. *Partial match sets* leave some of the model points un-paired with data points.

Match sets can be further classified by the number of allowed pairings between particular points in the model and data sets. *One-to-one* matching allows each model point to match with only one data point, and each data point maps to at most one point in the model. The *one-to-many* variation allows for a model point to match to more than one data point. The *many-to-one* mapping class allows each data point to possibly match with more than one model point. With the *many-to-many* variation each model and data point may possible match to many points in the opposite set.

Exact and Inexact Point Matching

The term *exact point matching* refers to the case where there is a complete mapping between model and data with no error. The goal of exact point matching is to find an exact model pose which aligns the model with some subset of the data. *Inexact point matching* refers to a problem variant where the goal is not to explicitly find a perfect or optimal mapping, but rather to

find a model pose such that many points in the model are within the explicit error bounds of some subset of the data.

When the goal is inexact point matching, a given pairing is considered to be *feasible* if for that pairing the transformed model point falls within the error bounds on the data point to which it is being mapped. The *maximum feasible match set* is the model to data mapping which contains the most feasible model to data mappings for a given pose.

Spurious and Missing Data

Spurious data refers to points which appear in the data set, but which do not match any point in the model under an acceptable or optimal mapping. A *missing data* point occurs when a point appears in the model, but no corresponding point can be found in the data under an applicable model transformation.

In model recognition problems, models are often crafted by hand. In this instance spurious points arise as a result of additional objects in the image or as sensor artifacts. Missing points are infrequent, but do occur as the results of sensor errors or occlusion of the model instance. Image registration presents a very different situation. In such problems sensors are unlikely to accurately extract every point in each image so there may be many points that occur only in the model or the data. In such problems, missing and spurious points are effectively equivalent.

Collectively these two conditions are referred to as *clutter*, and represent a serious challenge to point matching.

2.2 Baird's Tree Search Algorithm

In "Model Based Image Matching Using Location" (3), Baird examines the problem of point matching under a similarity transform; the case where the only transformations allowed to the model are translation, rotation and scale. Spurious and missing data points are not considered by Baird, but he does make some observations involving their effects on his algorithm, which is flexible enough to allow small numbers of such points. Baird is the first to cast the problem as a set of linear inequalities describing the error bounds on points, and lays the groundwork for Cass's pose equivalence algorithm (11).

Baird's linear equations follow directly from his introduction of polygonal noise regions with few sides: typically three or four. These bounded error regions are significant for several reasons. First, it provides a very flexible means of characterizing noise in the data. Depending on how the data was obtained, it is entirely possible that the variation in point location in one direction is different from the variation in another. Baird's technique allows this to be modeled explicitly. Secondly, by introducing the idea of an acceptable region about each data point in which transformed model points must lie, Baird moves away from the idea of fitting the model. Instead of looking for the best possible match, or the closest match, it is now possible to look for all valid, or feasible, matches. This variation is called inexact point matching and is common in later work.

When a possible match is considered, the noise bounds on the data points provide a system of constraints on the possible transformation of the model. If this system has a solution, then the match is said to be feasible. Baird employs a breadth first tree search to exhaustively search the space of possible matchings. Starting with all possible pairs of matches, the search considers all possible additions to these pairings. If the resulting addition yields an infeasible match, then that branch is pruned from the tree. This continues until a match is found or the entire tree is searched.

An interesting feature of Baird's work is the use of the soviet ellipsoid algorithm. Rather than using the simplex algorithm to determine if a matching is feasible, Baird uses the soviet ellipsoid algorithm which runs somewhat faster when it only has to determine that a solution exists, rather than an actual solution. This algorithm has the property that the additional constraints induced by considering a larger match set does not invalidate the previous solution; which can be used as a starting point for determining the feasibility of the new match.

Baird shows that when there is zero noise the worst case number of feasible matchings is $O(n^2 \log d)$. Average run times for the algorithm under moderate noise are on the order of $O(d^2)$, provided there are no spurious or missing data points. Such run times are obtained largely due to the ability of the algorithm to quickly eliminate branches of the search tree which are infeasible.

Baird only explicitly considers the case where model and data are related by a similarity transform. The algorithm might be extended to allow transformations beyond rotation, scale, and translation but at a cost of increased branching within the search tree. Similarly, Baird only considers one-to-one match sets but the algorithm might be extended to allow multiple pairings between model and

data points at an increased cost.

2.3 Pose Equivalence Analysis by Cass

Cass expands on the work of Baird, presenting a solution to the inexact point matching problem which runs in polynomial time even when the data contains spurious or missing data points (11). The pose equivalence algorithm searches pose space to determine all possible model poses which satisfy the error bounds on the data. By recognizing that this real valued space can be divided into regions, where each cell is associated with a different feasible mapping between model and data points, Cass is able to avoid searching a potentially infinite space of real values. Like Baird, the final version of the Cass work makes use of polygonal error bounds on the data points. However, earlier versions of the work develop the technique with respect to circular error bounds (9; 10).

Cass observes that for each of the *md* possible model-data pairs there exists a set of inequalities describing the transformations that will place the model point within the error bounds of the observed data point. These inequalities describe hyper-planes in transform space, with k possible hyper-planes per possible model-data pairing where k is the number of sides in the polygonal error bound surrounding each data point. The maximal match set will be the same for any pose within a given cell of this arrangement. Since every model to data pairing is represented in the arrangement, including the correct ones, good matches can be found by examining every cell in the arrangement and choosing those which meet some performance criteria such as placing a large number of transformed model points within the error bounds of the data points. An arrangement of *kmd* hyper-planes can be enumerated in $O(k^4m^4d^4)$ time using a well known result from computational geometry (14), assuming that the class of allowable transformations is limited to translation, rotation, and scale. This upper bound rises to $O(k^8m^8d^8)$ when the class of allowable transformation is expanded to a six dimensional affine transformation.

Cass observes that the data for this problem is highly structured, and the algorithm set forth in his work takes advantage of this to obtain a faster runtime. The Pose Equivalence Algorithm begins by arbitrarily choosing some rotation. It then computes the set of all translations which will map a model point to within one of the error bounds. Geometrically this results in a set of polygons in two dimensional translation space. This set of translations provides a corresponding set of maximal

match sets. The model is then rotated. As it is rotated the set of allowable translations changes, and with it the set of maximal match sets, whenever the intersections between polygons defined by the set of allowable translations change. This partitions rotation space into a set of equivalent rotations, and allows the set of all maximal match sets to be updated incrementally. This algorithm has an upper run time bound of $O(m^3d^2 \log m)$; in practice Cass finds the actual run time to approximate $O(m^2d^2)$.

Cass obtains further run time improvement with a randomized algorithm. By selecting just one initial match at random rather than the whole set of possible *md* pairings at the beginning of the algorithm, Cass greatly reduces the amount of computational effort required to find a good match or determine that no match exists. This process must be repeated until a good match is found, but the expected number of iterations is only $O(d)$ before the initial match chosen is a member of the correct maximal match set and the algorithm succeeds. In practice Cass finds this lowers the runtime to $O(md^2)$. This randomized algorithm will not give all maximal feasible match sets as will the base algorithm, and in the case where no model instance occurs in the data it must consider all *md* matches as the initial pairing, so it is not appropriate in all cases.

Cass suggests his algorithm operates in the four dimensional space of translation, scale and rotation. However, it is not clear that the proposed algorithm actual operates when the scale is not fixed. It could in theory, but it is not clear in the text that it is implemented in this manner. If in fact scale is not accounted for in the proposed algorithm, then run times for the variable scale version of the problem may be closer to the theoretical upper bound of $O(k^4m^4d^4)$ than the observed case of $O(m^2d^2)$.

Cass considers two variations on pose equivalence. In (11) the effects of including orientation information with each model and data point are considered. Cass finds that adding this information increases the asymptotic complexity but decreases the actual run times. In exchange for a more difficult analysis of the translation space as rotation is allowed to vary there is a great reduction in the number of constraints which are interacting. In (9) and (10) the case of circular error bounds is considered. Cass finds that this makes the constrains quadratic, and the result is asymptotic run times on the order of $O(m^6d^6)$. This is substantially slower than the algorithm developed with respect to polygonal error bounds, which can be used to approximate circular error bounds if necessary.

Cass's work is important because it sets out theoretical upper bounds on the difficulty of the in-

exact point matching problem and suggests an algorithm for obtaining it. Cass presents an algorithm which does better than this upper bound on two dimensional data, although there might still be room for improvement.

2.4 Breuel's RAST Algorithm

Breuel presents an algorithm for point matching in (7), and develops it further in (8). This algorithm, Fast Recognition using Adaptive Subdivisions of Transformation Space, or RAST, is formulated as a depth first search through pose space. The RAST algorithm attempts to maximize the number of pairings in the final solution, and once an area of the search space has been determined to provide fewer pairings than the best known solution it is pruned. Breuel develops the algorithm for a wide variety of transformation classes including translation and rotation, similarity, and affine transforms. Orthogonal projection with translation and out of plane rotation is discussed, but the projective transform is not.

Each search state is composed of a region of the appropriate transformation space and a match list of model and data correspondences which may possibly be satisfied by some transformation in the region. Regions of transformation space are given as lower and upper bounds on each dimension, resulting in a hyper-cube. Search is initialized to an arbitrarily large portion of transformation space, and the match list is set to all possible model and data pairs. This can possibly create problems if the range of possible transformations is not known a priori and can not be accurately guessed or bounded; although in practice this is not likely to be a serious limitation.

At each step in the search, this region of transformation space is subdivided into a set of sub-spaces. A new search state is then constructed by checking every model-data correspondence on the parent match list to determine if the new, smaller sub-space could possibly contain a transformation which will take the model point to within the error bounds on the data point. Those points which might match become part of the match list for the newly created search state. The sub-space with the largest number of possible matches is then chosen as the new search state. Search terminates when the size of the sub-spaces drops below some pre-determined threshold, or when a transformation is found which takes every model point on the match list to within the error bounds on the corresponding data points.

If the size of a match list ever drops below the size of the known best solution, the the search tree can be pruned. It is worth noting that this heuristic is admissible and this search scheme can be formulated as A* search. In practice, however, memory requirements force the use of depth first search on any problem large enough to be of practical interest.

The critical aspect of Breuel's work is the ability to determine if a given region of transformation space might contain a transformation which will take a model point to a data point. This is accomplished by determining the center of the transformation region and applying this transformation to each model point. When the transformed model point falls within the error bounds on a data point, that pair is added to the match list. Search terminates when every model point falls within the error bounds of some data point. If a model point does not fall within the error bounds on a data point, it is still possible that some transformation in the region will cause it to do so. A radius, based on the size of the transformation region and the distance of the original model point from the origin is then computed. If the transformed model point is not farther from the data point and its error bounds than this distance, then the pairing remains on the match list.

It is important to note that search progresses through transformation space, but search states are evaluated by projecting the results into image space. The procedure to do this is key to the RAST algorithm, and is the only portion of the algorithm that needs to change to enable searching different transformation classes.

It is possible to view the final leaves of the search tree as a discretization of pose space. Under this view the RAST algorithm resembles a hierichal hough transform. Pairings which fall within the projected error bounds for a region vote for that region, and regions with the most votes are subdivided and examined further until they reach some predetermined minimum size or the algorithm has found a transform which will match no more points than any other transform in its region.

The RAST algorithm as presented by Breuel is capable of solving the point matching problem for a large number of transformation classes, but not the projective transform class. In theory, it should be relatively straight forward to extend RAST to this case, but difficulties exist. All cases considered by Breuel are linear, but the projective transform is not. Every dimension added increases the size of projected error bound, and thus the number of potential matches. This has a direct effect on the ability of the algorithm to prune the search space and keep run times reasonable.

2.5 Hong and Tan's Canonical Form Algorithm

Hong and Tan present an efficient and novel algorithm based upon transforming a set of points to a canonical form (20). To convert a point set to conical form their algorithm first adjusts the origin of the point set to be at the point sets center of gravity, effectively removing any translation between the model and data sets. Hong and Tan then show that in $O(m)$ time an affine transformation can be found which normalizes the scale of the point set. The model and data set then differ only by a rotation, and Hong and Tan suggest a simple matching algorithm which aligns the first point in one set which each point in the other until a match is found; this algorithm runs in $O(m \log m)$ time as well, the complete algorithm can produce a solution in $O(m^2 \log m)$ time.

Sprinzak and Werman expand this algorithm from similarity transformations to transformations and point sets of arbitrary dimension (28). They describe how to find a transformation to the conical form in $O(m^{t-2} \log m)$ time, where t is the dimensionality of the transformation space.

Hong and Tan allow for explicit error bounds on each data point, and these bounds may take the form of circles, ellipses, or convex polygons without effecting the basic workings of the algorithm. Unfortunately, the algorithm they present for finding a match between the canonical forms of the point sets does not properly deal with this and may fail to find a feasible match when one is present. Their algorithm will work correctly for the exact point matching problem, where there are no error bounds and points must match exactly. However, the algorithm that Hong and Tan use to match point sets under rotation is not the important contribution of this work, and algorithms that do not suffer from this flaw may be used with the canonical form to produce an algorithm which can solve the inexact point matching problem (2).

The canonical form algorithm is the fastest available algorithm for point matching, and allows for noise in the location of the data points. However, it depends the center of gravity for the model and data sets being similar. The presence of even a single missing or spurious point is enough to violate this assumption and prevent the algorithm from finding a match. This makes renders the algorithm of little practical value, since real wold problems almost always involve some degree of spurious or missing data.

2.6 Hough Transform Algorithms

Several early approaches to point matching made use of a modified Hough transform, using an accumulator to keep track of evidence supporting various model poses. This approach is straightforward and easy to implement, but can quickly reach the limits of a computers memory. If the class of allowable transformations is large, then an accumulator which keeps track of evidence for a given pose must have coarse discretization, sacrificing precision in order to keep memory consumption reasonable. If the space of possible mappings is being searched, then the accumulator must usually keep track of all possible point pairings. Various approaches exist to keep these requirements reasonable, but they all suffer problems as transformation dimensionality increases. Despite these drawbacks, several researchers have found accumulator algorithms to be effective in certain, limited, problem domains.

Kahl, Rosenfeld, and Danker present one of the first point matching algorithms in "Some Experiments in Point Pattern Matching" (24). Their goal is image registration between two similar images. Their simple accumulator algorithm discretizes translation space and examines the distance between every pair of points in the first image and every pair of points in the second image. If the distance is within a given tolerance the appropriate accumulator cell is incremented. The final translation is then the one with the most votes in the accumulator. This algorithm can not handle any significant rotation or scale effects, and only small translations. Like many accumulator algorithms, however, it is insensitive to large numbers of missing or spurious points. Because the range of possible transformations is extremely limited, the accumulator does not require a great deal of memory.

Skea et. al present a simple accumulator point matching algorithm in "A Control Point Matching Algorithm" (26). Skea is interested in image registration for sonar images and geographical maps, and the algorithm reflects this. Skea assumes that the the two images to be registered have a large number of points, in the range of one to four hundred, and that the majority of these points will have a one to one match in the other image. Both images are assumed to have points which do not appear in the other image, but this is the exception and not the rule. The algorithm compares all possible point triples in each image and compares the centroids and triangles formed by these triples against set tolerances to determine if the three point matches between a pair of triples receive votes in the

accumulator. The accumulator is then scanned to determine those matches with the most votes, and from this the corresponding transformation. Because Skea et al. expect that the general orientation of any two images is known, the class of geometric orientations allowed is not specifically addressed. However, Skea does suggest a direction for modifications to allow three-dimensional projection effects. The nature of the algorithm rules out large translations and any significant rotations. Scale changes might be matched, but Skea does not test this and it is clear that the algorithm is not intended to deal with scale changes. No analysis of the algorithm is provided, but good run times on relatively modest hardware are reported. Memory requirements for the accumulator are $O(md/2)$, limiting the scalability of the algorithm.

Goshtasby and Stockman propose selecting points on the boundary of the convex hull of the data sets as a means of reducing the the number of points which must be considered (17). They determine the translation, rotation, and scale needed to match edges of the two convex hulls, and then transform the entire model by these parameters. Each point in the transformed model which falls within a given distance of a data point then gives one vote for that transformation. After all edges have been considered, the transformation with the most votes is re-applied to find a one-to-one mapping between model and data points, and least squares is applied to refine the transformation to minimize the error in the match. The formulation of this algorithm allows for many points to go unmatched, although Goshtasby and Stockman seem to expect that this will often not be the case. It is not guaranteed to find a solution, but a more costly variant which examines edges between all pairs of points on the convex hull makes success much more likely. The authors do note that as the number of points in each data set rises the likelihood of success increases, provided the number of points on the convex hull also increases, thus increasing the likelihood of some edges being held in common between the two graphs. It seems that in certain applications, data at the edges of the data set may be extremely noisy or at least less reliable than data on the interior, but for other applications this algorithm seems reasonable.

Chang et al. propose an interesting accumulator algorithm (12) based on the observation that the orientation of the vector between one point in an image and a second point stays the same under translation, rotation, and scale. They also observe that the distance between any two sets of matched points can be used to determine the scale factor of the transformation. Based on these observations

they develop an algorithm which checks every pair of correspondences and determines the number of feasible matchings for any given scale and orientation. They then select the scale and orientation with the most support, and produce a mapping based on which vector orientation and lengths match. From this correspondence they compute the optimal translation, scale and rotation. This scheme automatically accounts for translations and keeps the accumulator confined to two dimensions. Thresholding allows for some noise in the exact position of points, and for missing and spurious points in the data. Final run times are on the order of $O(m^2d^2)$, keeping this algorithm competitive with other algorithms using the same class of transformations. The authors report fast run times on both synthetic and real world data. However, an examination of their test cases suggests that they assume that the two point sets to be matched do not have greatly varying orientations. In the authors test cases rotation is not more than seventeen degrees, and scale does not vary by more than ten percent; translation in the test cases is more significant but still not large. The algorithm proposed should be capable of handling much larger rotation and scale changes between point sets, and so there is some question as to why this is not tested by the authors. Their test data also appears to fit the assumption that most points in each point set will be matched. Should this assumption be violated, then the chance for multiple partial or misleading matches in the data increases, making it hard to find the peak in the accumulator. The notion of using the orientation and magnitude of vectors between pairs of matched points is interesting, but the algorithm proposed in this case appears to be limited in its ability to solve problems with a larger class of allowable transformations.

Grimson and Huttenlocher provide a detailed analysis of the Hough Transform (18) and show that these algorithm can not be expected to be accurate in the presence of noisy point measurements. When measurements of feature location are noisy there is a corresponding uncertainty in the range of transformations which a given pairing may represent. This results in many transformation regions receiving the support of a given pairing. Grimson and Huttenlocher show that the chance that a large number of pairings will support the same region of transformation space at random is also high. Consequently, in complex environments the hough transform is likely to yield a prohibitive number of false positives, making location of the true pose based on the results of the hough transform impractical.

2.7 Weak Methods

Point matching is a difficult problem and algorithms which provide polynomial solutions to specific variations of the problem are not readily extended to more general versions of the problem. Although heuristics exist to guide these algorithms and improve their run times, these too are often tied closely to the exact problem variant being considered. Deterministic algorithms are further hampered by an inability to separate the various dimensions of the problem. The optimal translation for the occurrence of a model in a given data set is often dependent on the scale and rotation, and vice versa. Likewise, the optimal pairing of one model point to a point in the data is always dependent on the pairings of the other points.

Given these difficulties it seems reasonable to consider the class of algorithms known as weak methods. These algorithms do not require extensive information about the nature of the search space, they need only some means of evaluating solutions to determine their relative fitness. Weak methods are generally also non-deterministic, requiring some random factor in their initialization or later execution. This can mean that they are not guaranteed to find a solution if one exists, if the algorithm might find one given a different random seed. It also means that the run times on these algorithms are not guaranteed, although they can usually be analyzed to determine the expected run time. Usually these run times are relatively modest.

Starink and Backer, in "Finding Point Correspondences Using Simulated Annealing" (29), apply simulated annealing to the problem of point matching. They are focused primarily on finding point correspondences to register stereo image pairs, although they do present favorable results of their algorithm being used for locating markers in DNA. The algorithm they apply seems to be a straightforward application of the basic simulated annealing algorithm; with each pair added to the mapping increasing or decreasing the cost, or energy function, of the mapping. Unfortunately, they do not describe the specifics of how this function is formulated. One interesting element of this work is that they do allow for both one-to-many and many-to-one matches between the first and second data sets. They observe that allowing such matches enables their algorithm to find a better solution at the cost of a longer convergence time. It should be noted that because their problem domain is stereo correspondence, an optimal solution is not needed, and the model and data sets are of roughly

the same size. The expectation is that most points in each set will be matched. Because they are ultimately looking for a set of registration parameters and not a model instance in the data, this technique is sufficient. However, in cases where all possible model instances in a data set are required, this technique is not suitable.

Vinod and Ghose apply an asymmetric neural network to the task of finding point matches (31). They formulate the problem of finding a match in the presence of translation, rotation, and noise as zero-one integer matching problem. It is interesting that they represent solutions as a binary string of length $m \times d$ where m is the number of points in the first data set and d is the number of points in the second. This representation allows for many-to-many mappings, but the neural network is developed in such a way that it is constrained to produce a one-to-one mapping. They allow for missing and spurious points in the point sets to be matched, but require that the amount of noise present in the data be known *a priori*. They require approximately $O(d^3)$ neuron updates to find a solution; and report very fast run times on old hardware, and neural nets scale extremely well when simulated on multi-processor machines, so the algorithm seems to have the potential to be used for very large problems.

Their approach suffers from its failure to address scaling, and it seems that such capability is not easily added. They also operate under the assumption that there exists a single match which matches the greatest number of points, and places all matches within the given error bounds. In practice this might not be the case. While a valid assumption for their stated problem domain of image registration, this assumption breaks down quickly in other domains where we might expect multiple model instances or that there might be a number of large matches consistent with the error bounds.

Both of the previous attempts at applying weak methods to point matching seem forced and unnatural. Neither algorithm works naturally with the intuitive representation of the solution space, and each forces the problem to conform to the boundaries of the algorithm rather than adapting the algorithm to the problem. Consequently, it is no surprise that both approaches suffer serious weaknesses. The neural network suggested by Vinod and Gosh can not handle scaling and a new network needs to be trained for each problem. Starink and Backer can not determine absence of a match or multiple matches, and seem to be limited in the class of allowable transformations.

prevent the algorithm from correctly determining a correct correspondence.

2.9 Beveridge's Line Matching Work

Local search for model recognition was first proposed by Beveridge in (5). Beveridge observes that the model recognition problem can be stated in terms of combinatorial optimization; finding the optimal set of model pairings in the data. Beveridge's work is formulated for line segments; where multiple fragmented data lines will likely have to be matched against a single model line. For this reason it explicitly allows and checks for many-to-many matches. This results in a search space of 2^{md}. Beveridge, however, finds that for many small models local search is capable of sorting through a great deal of clutter to successful locate models in an efficient manner.

In (32), Whitley et al. apply a modification of the messy genetic algorithm to the optimization problem set forth by Beveridge in (5). They use what they term feature subset selection to initialize the algorithm with a population composed of partial mappings, each consisting of a pairing between three distinct model lines and three distinct data lines. These pairings map a line and its two closest neighbors to another line and its two closest neighbors. They find that on large problems this algorithm performs several orders of magnitude better than local search.

This work is significant for several reasons. First, because it provides a formulation of model recognition as combinatorial optimization which can be adapted to point matching. Second, this work successfully makes use of weak methods to solve a problem similar to point matching, and suggests algorithms which might be adapted and applied to point matching.

2.10 Conclusion

When there are no spurious or missing points in the data set, the conical form transformation of Hong and Tan, as extended by Sprinzak and Werman, provides a solution to the point matching problem in $O(m^{t-2}\log m)$ time; where t is the dimensionality of the transformation space. This procedure is relatively tolerant of noise in the exact position of the data points. When the constraints on the data are met this is as good an algorithm as is likely to be found. Unfortunately it fails immediately in the the presence of even a single missing or spurious point, making it unsuitable for most real world applications.

Cass's pose equivalence algorithm provides a polynomial time solution to the inexact point matching problem. This algorithm allows for noise in the data set and missing or spurious points. It has the desirable property of finding all possible consistent maximal match sets, making it suitable for both the model recognition and image registration cases we outline in chapter 3. In the case where the class of allowable model transformations is limited to translation and rotation Cass provides a series of optimizations to his algorithm which allow it to operate in $O(m^2d^2)$ time. When the class of allowable transformations allows three dimensional transformation of two dimensional points those optimization no longer apply and the run time rises to $O(m^8d^8)$. Although the algorithm has a polynomial run time, it is clear that large problem instances will require a different approach to be solved efficiently.

The RAST algorithm provides effective solutions to low order point matching problems, but has not been developed to deal with the projective transformation class and it is not entirely clear how such an extension may be formulated. The RANSAC algorithm provides the existing solution to the projective point matching problem, and can solve certain problem instances very quickly, but may not be appropriate in all cases. In the presence of clutter it may fail to consider a correct pairing, a potentially fatal weakness that local search does not share. The key feature algorithm purposed here may also be used to overcome this difficulty, but without the ability to explicitly drop points from a match RANSAC can not completely overcome the problem of clutter.

Chapter 3

The Point Matching Problem

3.1 The Problem to be Examined

The point matching problem has several dimensions, depending on what it is used for and what is needed from a solution. Some applications require finding the geometric pose relating one set of points to another, with little concern for the correspondences between points. Other require exactly the reverse. Fortunately, if the pose is known the correspondences can generally be determined, and conversely, a correspondence between points may be used to determine the optimal pose. Occasionally, point matching has been formulated as the task of finding an exact match between points in two sets, but more frequently only a subset of the points in each set can be made to match and so most algorithms deal with this case. Real world problem domains can place additional demands on a point matching algorithm.

The class of allowable transforms is the major distinction between point matching algorithms. Many authors have presented point matching algorithms for the rigid body problem, considering transforms composed only of translation and rotation (8; 11). Similarity transforms are also well studied (3; 20), and most algorithms can deal with at least these two transformation classes. Affine transformation classes are less frequently dealt with, probably due to their limited application. The projective transformation class is not well studied, although some previous work addresses it at a theoretical level (11), and the RANSAC family algorithms can solve some problems in this class (15; 19).

3.2 Exact vs. Inexact Point Matching

Exact point matching requires that all model points match to some point in the data set. Occasionally the additional requirement is made that all data points match to some model point (20). This may be the case, for instance, in certain highly controlled environments, where sensor behavior is consistent and repeatable. The exact point matching task is to find a pose which takes all model points to within the error bounds on the data points, or to find those data points which correspond to the model. In this case, a potential solution is either correct or incorrect; if one or more model points is unmatched the solution is incorrect. For many real world cases, however, this requirement can not be meet.

Inexact point matching assumes that some model points will not have a match in the data set. This may be for a number of reasons, such as sensor error or the presence of occluding objects in the scene. In this case point matching becomes the problem of finding a pose which will cause the maximum number of model points to fall within the error bounds on the data. Alternatively, inexact point matching is the task of finding subsets of the model and data such that points in one subset map to points in the other. In this case it becomes possible to examine partial solutions. A solution which matches some points may be a good place to start looking for a solution which matches more points. Even if more matches can not be found, the partial solution may be a good solution.

3.3 Problem Domains

Point matching is a well studied problem with many practical applications. The most versatile algorithms have been developed to address the problem in the abstract (3; 7; 11). Unfortunately, these algorithms have generally been verified only with synthetic data; details of their application to real world problems are not addressed. Other work has address point matching strictly in the context of practical applications (12; 17; 21). These algorithms can solve the problem in the domains they where developed for, but encounter difficulty when applied to other problem domains. There is a need for a point matching algorithm which is general and widely applicable, and can be shown to be applicable on a wide range of real world problems.

While point matching has many possible practical applications, the algorithms examined here will focus on two problem domains. Image registration is the task of finding the pose relating two

images. Model recognition is the task of finding an instance of an object model within a data set. Many other point matching problems can be seen as a specialization of one of these two cases, and they are different enough that an algorithm that can solve both problems can be expected to do well in novel problem domains.

3.3.1 The Image Registration Case

In many problems of practical interest, the size of the model and data sets is approximately the same. This is often the case in image registration problems. In such problems, both data sets are typically very large, on the order of several hundred points. Although knowledge of the exact point correspondences may be desirable, the real task is to determine the geometric relationship between two images. The focus on pose, and the number of points in both the model and data, mean that an algorithm may miss a large number of correspondences and still produce a solution which is close enough to the optimal to provide a practical solution to the image registration task.

Because the model and data sets are close in terms of size it is unlikely that more than one instance of the model will appear in the data. This means that the search for a solution can terminate as soon as a feasible solution is found. Missing and spurious points can be assumed to exist, and may constitute a sizable portion of the data. However, these will be the exception rather than the rule, most points in both sets will be part of a pairing. The large number of matches between model and data minimizes the effects of missing or spurious points; one point out of a hundred missed in a match will not greatly effect the final orientation of the model.

3.3.2 The Model Recognition Case

The problem is somewhat different when the model is substantially smaller than the data. This is often the case when attempting to locate an instance of an object model within a larger image. It is not unreasonable to expect the number of data points to be an order of magnitude larger, or more, than the number of model points. Thus, a problem in this domain may involve matching a 10 point model to 100 data points, or perhaps 100 model points to 10,000 data points.

Most points in the data set will be spurious, with only a few actually belonging to a correct match or the being part of the same match. Missing points also present some difficulty because the smaller size of the object model makes each point that can not be matched more significant. Model recogni-

tion raises the possibility of multiple correct matches for the model in the data set, and an algorithm operating in this problem domain should locate all correct matches.

3.4 Transformation Classes

The class of allowable transforms is the single most important difference between point matching algorithms. Many algorithms work well for one or two transformation classes, but have difficulty or fail completely when applied to problems requiring a higher order transformation class. The reverse, however, is not true. Each transformation class is a proper super-set of the lower order transformation classes. An algorithm capable of solving the similarity problem can solve the rigid body problem, and ideally an algorithm capable of solving problems in the projective class can in theory solve these problems as well as those in two-dimensional cases such as general affine or similarity transforms.

In practice it is often best to restrict the transformation class as much as possible. Additional degrees of freedom dramatically increase the size of the pose space, but also increases the size of the pose space, and this in turn makes searching correspondence space much more difficult. In determining which model points might match to which data points, the extra degrees of freedom may allow additional pairings to be considered, increasing the difficulty of distinguishing between good and bad sets of pairings or causing the optimal match set to be missed entirely. The presence of clutter only increases this difficulty as it allows missing or spurious points to find matches that would not be considered under a more restrictive transformation class.

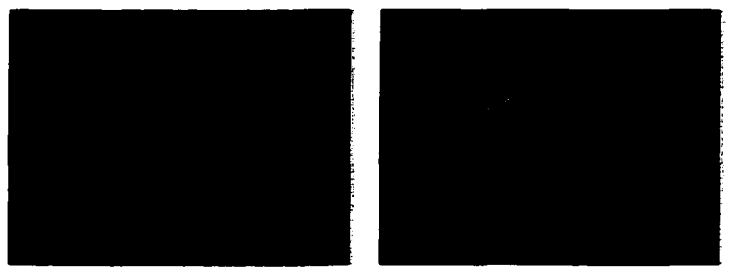
3.4.1 Rigid Body Transforms

Translation and Rotation, the rigid body problem, is perhaps the most well studied transformation class for point matching. Consider a camera pointing straight down at an assembly line, and the need to identify the exact position and orientation of widgets coming off this assembly line. The widgets will all be lying flat on the conveyor belt, but may be at different positions across the belt or relative to the camera as the belt moves. They may also be rotated as the results of previous operations. However, because the position and focal length of the camera and the widget size are both fixed scale or projection effects do not need to be considered.

In this case there are only three degrees of freedom, translation in x and y, and the angle of rota-

tion. This transformation can be specified as

$$P_r = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.1}$$



Two images related by translation and rotation

3.4.2 Similarity Transforms

The similarity transform class is well studied, and most published algorithms can solve this case. Here allowable transformations include translation, rotation, and scale. The similarity transform is given by

$$P_s = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.2}$$

$$a = s\cos\theta \tag{3.3}$$

$$b = s\sin\theta \tag{3.4}$$

Stating the transformation in this way, rather than separating scale s from rotation, simplifies the task of pose determination as detailed in chapter 4. Since $\cos^2\theta + \sin^2\theta = 1$ for all θ it can be shown that $s = \sqrt{a^2 + b^2}$. Then, $\theta = \arcsin(a/s)$.

3.4.3 Affine Transforms

The affine transformation adds the possibility of shear to the class of allowed transforms. Point matching under this class of transforms is not well studied relative to the simpler transformation

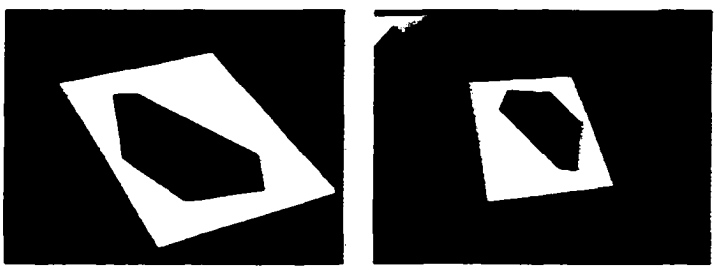
classes, although Breuel presents an algorithm which specifically addresses it (8). It adds two degrees of freedom to the similarity transform class, but does not greatly increase the number of practical problems that can be solved. An affine transformation takes the form

$$P_a = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.5}$$

3.4.4 The Projective Transform

Unlike the affine case, the projective transform greatly increases the number of practical problems that can be solved. The projective transform can fully describe three dimensional camera motion around a three dimensional object, when all points of interest are co-planar. For instance, aerial photography and matching star maps both require a point matching algorithm to operate on this transformation class. The projective transform adds two additional degrees of freedom to the affine case, to account for the effects of projection. These additional degrees of freedom make the transformation non-linear. The projective transform is given by

$$P_p = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \tag{3.6}$$



Two images related by a projective transform

Interpreting a projective transformation in terms of the associated placement of two 3D cameras is difficult. Partial interpretations are possible but may not be generally useful. Consider the composition of the projective transformation.

If decomposed into its component parts of translation, scale, rotation, skew, and projection, it is possible to solve the resulting equations and relate the parameters of the projective transform back

to its component parts. However, the results of this decomposition depend on the order in which the four component matrices are composed. Considering each of the twenty-four possible orderings, and in each case solving for the values of component parameters such as rotation and scale in terms of the parameters of the projective transform, gives one of four different interpretations of the parameters. Which interpretation class results depends on the order of the rotation and shear matrices, and the projection and translation matrices.

The four component matrices of the projective transform can be given as

$$Pr = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p_x & p_y & 1 \end{bmatrix} \tag{3.7}$$

$$sR = \begin{bmatrix} \phi_x & -\phi_y & 0 \\ \phi_y & \phi_x & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$Sh = \begin{bmatrix} 1 & q_x & 0 \\ q_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

(3.11)

When projection precedes translation, and scaled rotation precedes shear, such as in the case where

$$P_p = Pr \times sR \times Sh \times T \tag{3.12}$$

the parameters of the projective transformation decompose as

$$t_x = \frac{-fb+ce}{-db+ea} \tag{3.13}$$

$$t_y = \frac{-fa+cd}{-db+ea} \tag{3.14}$$

$$p_x = \frac{-hd+eg}{-db+ea} \tag{3.15}$$

$$p_y = \frac{gb-ha}{-db+ea} \tag{3.16}$$

$$q_x = \frac{-b^2 - db - e^2 + ea}{ab + de} \tag{3.17}$$

$$q_y = \frac{db + a^2 + d^2 - ea}{ab + de} \tag{3.18}$$

$$\phi_x = \frac{(b+d)(ab+de)}{(b^2 + 2db + a^2 + d^2 + e^2 - 2ea)} \tag{3.19}$$

$$\phi_y = \frac{-(-de^2 - bea + dea + a^2 b)}{(b^2 + 2db + a^2 + d^2 + e^2 - 2ea)} \tag{3.20}$$

When scaled rotation proceeds shear, this becomes

$$P_p = Pr \times Sh \times sR \times T \tag{3.21}$$

and

$$t_x = \frac{-(-ec + fb)}{(ea - db)} \tag{3.22}$$

$$t_y = \frac{(fa - cd)}{(ea - db)} \tag{3.23}$$

$$p_x = \frac{(-dh + ge)}{(ea - db)} \tag{3.24}$$

$$p_y = \frac{-(-ha + gb)}{(ea - db)} \tag{3.25}$$

$$q_x = \frac{(a^2 - ea + b^2 + db)}{(ad + be)} \tag{3.26}$$

$$q_y = \frac{-(-d^2 - e^2 + ea - db)}{(ad + be)} \tag{3.27}$$

$$\phi_x = \frac{(ad^2 + edb + dba + eb^2)}{(d^2 + 2db + b^2 + e^2 + a^2 - 2ea)} \tag{3.28}$$

$$\phi_y = \frac{(-e + a)(ad + be)}{(d^2 + 2db + b^2 + e^2 + a^2 - 2ea)} \tag{3.29}$$

When translation precedes projection, regardless of the ordering of shear and rotation, the resulting equations are extremely long and complex, making an understanding of them extremely difficult.

As can be seen, these interpretations are not convenient, mathematically complex, and it is difficult to relate them back to real world geometry. To establish an unambiguous interpretation of the eight parameters of the projective transform relative to simpler constituent transformations requires selecting one of the cases above as the standard for interpretation. Unfortunately, it is not obvious which is most informative or natural.

The traditional approach to understanding the projective transformation is through the fundamental matrix, but Hartley and Zisserman show that this is not possible when the transformation relates points on a plane (19). The fundamental matrix, F, describes the relationship between the two camera angles used to view a scene based on the epipolar geometry. The epipolar geometry is the geometry of the intersection of the image planes, using the line between the center of the two cameras as a baseline (19). Let m_i represent the model point of the i^{th} pair, and let d_i represent the data point paired with m_i. Then, for all pairs, the fundamental matrix satisfies the equation

$$d_i^T F m_i = 0 \tag{3.30}$$

In practice, the position of each point will not be precise, but rather a noisy estimate of the points true location. This means that equation 3.30 can not be completely satisfied. Standard techniques exist for the estimation of F in this case (19), but the goal here is an interpretation of the pose P, and its relationship to the fundamental matrix, not the estimation of the fundamental matrix itself. In this case each point d_i is the transformed version of point m_i, rather than a point taken directly from another point set. This means that

$$d_i = P m_i \tag{3.31}$$

$$m_i = P^{-1} d_i \tag{3.32}$$

which allows equation 3.30 to be rewritten as

$$d_i^T (F P^{-1}) d_i = 0 \tag{3.33}$$

$$d_i^T W d_i = 0 \tag{3.34}$$

$$W = F P^{-1} \tag{3.35}$$

$$F = W P \tag{3.36}$$

This shows that obtaining the fundamental matrix F, and thus a clear interpretation of the pose, depends on finding W. Expanding equation 3.34 gives

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \tag{3.37}$$

which can be rewritten as

$$w_1x^2 + w_2xy + w_3x + w_4xy + w_5y^2 + w_6y + w_7x + w_8y + w_9 = 0 \tag{3.38}$$

$$w_1x^2 + w_5y^2 + w_9 + (w_2 + w_4)xy + (w_3 + w_7)x + (w_6 + w_8)y = 0 \tag{3.39}$$

From this it is clear that W, and thus F, is not unique. The last three terms of equation 3.39 allow for an infinite number of trade-offs in the parameters of W effectively leaving three free variables. Any skew symmetric matrix can meet the requirements of W.

Without a unique fundamental matrix, and thus a unique and definitive interpretation of the projective pose relating two point sets, it is difficult to evaluate the geometric pose associated with a given correspondence. The projective transformation provides enough degrees of freedom that it is possible to match a large number of points under an inappropriate, improbable, or impossible pose. Without a clear interpretation of pose, it is difficult to distinguish between matches which are correct, and those which match a large number of points by distorting the model. This information is a critical component of match evaluation, and key to the search process.

Chapter 4

Evaluation of Point Correspondences

4.1 The Evaluation Function

Here point matching is formulated as combinatorial search. The goal will be to find a set of correspondences which minimizes the error function given by equation 4.1. This error function is based on the the distance between paired points when an optimal transformation is applied to the model. Additional terms drive the search towards sets of correspondences where all model points are matched, and penalize matches which result in unlikely or impossible poses. As formulated this error function allows for the possibility of many-to-many pairings between model and data points, an important consideration in some real world cases.

$$E_{match} = \frac{1}{\sigma^2}\sum_i ||P^*(m_i) - d_i||^2 + u + S(P^*) \tag{4.1}$$

This evaluation function is asymmetric, matching a point set M designated as the model with data point set D. The set of correspondences to be evaluated is denoted C. The model point participating in the i^{th} pairing is designated m_i, and is matched to the corresponding data point d_i. P^* is the optimal transformation or pose which minimizes the sum of the squared distances between all points m_i, and their paired data points d_i. The u term is the omission penalty, the number of model points which have no corresponding data point. The value of $S(P^*)$ is a penalty on degenerate or unlikely poses, and is explained in section 4.1.4.

Beyond a certain distance, a transformed model point should not be paired with a data point. This distance, denoted by σ, can be thought of as the error bound on a given data point. If P^* causes the transformed model point to be further from its corresponding data point than σ, then this pairing

will contribute a value greater than one to the error. Since the penalty for leaving a model point unmatched is one, an algorithm attempting to minimize function 4.1 should drop this pairing from the match. Determining the distance between two points requires the finding of a square root, so it is more efficient to compare the squared distance between paired points with the value of σ^2.

Evaluating this equation for a given match C requires finding the transformation P^*. This can be done by dropping the missing points and degenerate transformation penalties from equation 4.1, and minimizing the result.

$$\sum_i ||P^*(m_i) - d_i||^2 \tag{4.2}$$

Equation 4.2 can be solved using standard least squares techniques, but the exact procedure is dependent on the transformation class. Similarity and affine transforms lead to a linear set of equations which can be solved to give the optimal transformation for any set of point correspondences. The projective case is more complicated. All three cases are presented here. In the discussion that follows the coordinates of model point m_i are denoted by x_i and y_i, and the coordinates of the corresponding data point d_i with u_i and v_i.

4.1.1 Similarity Transforms

Similarity transforms, as given by equation 3.2 have four degrees of freedom. It will help to restate this equation as

$$\begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{4.3}$$

The optimal transformation can be found by substituting equation 4.3 into equation 4.2 to obtain

$$E_{match} = \sum_i \left(\begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \right)^2 \tag{4.4}$$

To reduce clutter in the following equations, all following sum terms are assumed to be sums over every pair of points in the pairing C. Subscripts for these terms have been dropped.

Multiplying the matrices in equation 4.4 and moving the sum term directly into the equation results in

$$
\begin{aligned}
E_{match} \;=\; & N(t_x^2+t_y^2)+(a^2+b^2)\left(\sum x^2+\sum y^2\right)+\sum u^2+\sum v^2+ \\
& 2\left(t_x\sum u+t_y\sum v\right)-a\left(\sum(xu)+\sum(yv)+t_x\sum x+t_y\sum y\right) \\
& +b\left(\sum(xv)-\sum(uy)-t_x\sum y+t_y\sum x\right)
\end{aligned} \tag{4.5}
$$

where N is the number of pairs in C.

To minimize the error function, take the partial derivatives of equation 4.5 with respect to a, b, t_x, and t_y, and set each of the resulting equations to zero. This gives four equations and four unknowns given by equations 4.6 through 4.9.

$$a\left(\sum x^2+\sum y^2\right)-\sum(xu)-\sum(yv)-t_x\sum x-t_y\sum y=0 \tag{4.6}$$

$$b\left(\sum x^2+\sum y^2\right)-\sum(yu)+\sum(xv)-t_x\sum y+t_y\sum x=0 \tag{4.7}$$

$$Nt_x-a\sum x-b\sum y+\sum u=0 \tag{4.8}$$

$$Nt_y-a\sum y+b\sum x+\sum v=0 \tag{4.9}$$

It is now possible to solve equations 4.6 and 4.7 for the values of t_x and t_y in terms of a and b. This gives equations 4.10 and 4.11, which can be used to obtain the values of t_x and t_y.

$$t_x=-\frac{a\sum x-b\sum y-\sum u}{N} \tag{4.10}$$

$$t_y=-\frac{a\sum y+b\sum x-\sum v}{N} \tag{4.11}$$

Substituting equations 4.10 and 4.11 into equations 4.6 and 4.7 gives

$$a\left(\sum x^2+\sum y^2\right)-\sum(xu)-\sum(yv)+\frac{\sum x\sum u+\sum y\sum v-a((\sum x)^2+(\sum y)^2)}{N}=0$$

$$b\left(\sum x^2+\sum y^2\right)+\sum(xv)-\sum(yu)+\frac{\sum y\sum u-\sum x\sum v-b((\sum x)^2+(\sum y)^2)}{N}=0$$

which can be rearranged to form

$$a \;=\; \frac{N(\sum(xu)+\sum(yv))-\sum x\sum u-\sum y\sum v}{N\sum(x^2+y^2)-\sum x\sum x-\sum y\sum y} \tag{4.12}$$

$$b = \frac{N(\Sigma(xv) - \Sigma(yu)) + \Sigma y \Sigma u - \Sigma x \Sigma v}{N\Sigma(x^2 + y^2) - \Sigma x \Sigma x - \Sigma y \Sigma y} \tag{4.13}$$

The optimal similarity transform parameters can be obtained from equations 4.10, 4.11, 4.12, and 4.13. Solving these equations requires time linear with the number of pairings in the correspondence C. However, incremental search algorithms can do better than this. By storing the intermediate terms of the pose computation, and pre-computing the contribution to each term made by any given pairing, algorithms such as local search can compute a new pose from a prior one on the basis of which pairings have changed. This is a modest but significant optimization to the pose finding process. It should be noted that the implementation of local search used in this study does not implement this optimization for similarity transforms. For problems of a modest size, solved on modern hardware, the ability to incrementally update a similarity transform is not necessary to achieve acceptable run times. Attempts to solve larger problems may benefit from such an optimization, but it seems likely that the best investment is simply more processing power.

4.1.2 Affine Transforms

The case where the model can be transformed by a six degree of freedom affine transformation is very similar to the similarity transform. The match error can be expressed as

$$E_{fit} = \sum_i \left(\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} - \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \right)^2 \tag{4.14}$$

It is possible to find a linear solution to this problem by fully expanding the equation and differentiate with respect to each variable.

$$\frac{\partial E_{fit}}{\partial a} = \sum_i (2ax_i^2 + 2x_i b y_i + 2x_i c - 2x_i u_i) \tag{4.15}$$

$$\frac{\partial E_{fit}}{\partial b} = \sum (2ax_i y_i + 2by_i^2 + 2y_i c - 2y_i u_i) \tag{4.16}$$

$$\frac{\partial E_{fit}}{\partial c} = 2nc + \sum (2ax_i + 2by_i - 2u_i) \tag{4.17}$$

$$\frac{\partial E_{fit}}{\partial d} = \sum (2dx_i^2 + 2x_i e y_i + 2fx_i - 2x_i v_i) \tag{4.18}$$

$$\frac{\partial E_{fit}}{\partial e} = \sum (2x_i y_i d + 2ey_i^2 + 2fy_i - 2y_i v_i) \tag{4.19}$$

$$\frac{\partial E_{fit}}{\partial f} = 2nf + \sum (2dx_i + 2ey_i - 2v_i) \tag{4.20}$$

Using an approach similar to that used to solve for the parameters of the similarity transform, settings these equations to zero and solving for the parameters of the affine transformation gives a set of linear equations for the optimal affine pose. Equations 4.22 through 4.28 give the least squares estimate of the optimal affine pose, P_a. Each of these equations has a common denominator which given by 4.21.

$$denom = -n\left(\sum xy\right)^2 + 2\sum xy\sum x\sum y + n\sum y^2\sum x^2 \tag{4.21}$$

$$-\sum y^2\left(\sum x\right)^2 - \left(\sum y\right)^2\sum x^2 \tag{4.22}$$

$$a = -\frac{-n(\sum xu\sum y^2 + \sum xy\sum yu) - \sum xy\sum y\sum u}{denom} + \frac{\sum x\sum u\sum y^2 - \sum y\sum x\sum yu + \sum xu(\sum y)^2}{denom} \tag{4.23}$$

$$b = \frac{n\sum yu\sum x^2 - \sum yu(\sum x)^2 - n\sum xy\sum xu + \sum xy\sum x\sum u}{denom} + \frac{-\sum y - \sum u\sum x^2 + \sum y\sum x\sum xu}{denom} \tag{4.24}$$

$$c = \frac{\sum u(\sum xy)^2 - \sum u\sum y^2\sum x^2 + \sum x\sum xu\sum y^2}{denom} + \frac{-\sum x\sum xy\sum yu + \sum y\sum yu\sum x^2 - \sum y\sum xy\sum xu}{denom} \tag{4.25}$$

$$d = \frac{-n(\sum xv\sum y^2 + \sum xy\sum yv) - \sum xy\sum y\sum v}{denom} + \frac{\sum x\sum v\sum y^2 - \sum y\sum x\sum yv + \sum xu(\sum y)^2}{denom} \tag{4.26}$$

$$e = -\frac{n\sum yv\sum x^2 - \sum yv(\sum x)^2 - n\sum xy\sum xv + \sum xy\sum x\sum v}{denom} + \frac{-\sum y - \sum v\sum x^2 + \sum y\sum x\sum xv}{denom} \tag{4.27}$$

$$f = \frac{\sum v(\sum xy)^2 - \sum v\sum y^2\sum x^2 + \sum x\sum xv\sum y^2}{denom} + \frac{-\sum x\sum xy\sum yv + \sum y\sum yv\sum x^2 - \sum y\sum xy\sum xv}{denom} \tag{4.28}$$

4.1.3 Least Squares Estimation of the Projective Transform

Ideally the outcome of a pose estimation procedure should be independent of the coordinate system used. However, for the projective case, when there are more than four correspondences, and these correspondences are not exact due to noise, this is not the case. Hartley and Zisserman (19) demonstrate that this can lead to noise in the estimated positions of the points causing numerical

instability in the computed pose. Such instability can potentially create problems for a search routines. Consider the case where a correspondence and associated pose have been found, and there is an unmatched model point which lies close to an unmatched data point. This is an attractive match to algorithms such as local search and is probably a correct pairing. Because the existing pose already places the model point close to the data point a new estimate of the pose including the new pairing should not be greatly different. If the pose estimate is numerically unstable, then this may not be the case, the addition or removal of a pairing may significantly effect the pose and thus the value of all other pairings. Potentially this instability could cause correct pairings to be missed and make finding the global optima much more difficult.

The solution to this problem is to normalize both point sets by changing their origins and then rescaling them. The origin should be placed at the centroid of the data set, the average x and y position of the original set. Then a scale factor is applied to the data so that the average distance from one transformed point to the origin is $\sqrt{2}$. This corresponds to the average distance from the origin to any given point being the same as the distance between the origin and the point $(1,1)$.

The pose relating the original point sets can be found based on the pose between the normalized point sets and the transformations used to normalize the point sets. Let $\hat{P}$ be the 3×3 matrix corresponding to the pose P_p found between the normalized point sets. Let N_m and N_d be the normalizing transformations applied to the model and data sets respectively, again in the form of a 3×3 matrix. Then, the pose P_p relating the original point sets can be found as $P_p = N_d^{-1} \hat{P} N_m$.

Once the point sets to be matched have been normalized, least squares estimation can be applied to determine the optimal pose. Applying the projective transformation, equation 4.2 can be restated as

$$E_{fit} = \sum_i \left(\left(u_i - \frac{ax_i + by_i + c}{gx_i + hy_i + 1} \right)^2 + \left(v_i - \frac{dx_i + ey_i + f}{gx_i + hy_i + 1} \right)^2 \right) \tag{4.29}$$

The least squares solution to equation 4.29 can be found by differentiating the equation with respect to each of the eight parameters in the transformation, setting the result to zero, and solving the resulting series of eight equations. Unfortunately the resulting system of equations is non-linear and not easily solved. The common solution to this problem is to re-formulate equation 4.29 so that it is linear.

This equation contains two terms.

$$\left(u_i - \frac{ax_i + by_i + c}{gx_i + hy_i + 1}\right) \tag{4.30}$$

$$\left(v_i - \frac{dx_i + ey_i + f}{gx_i + hy_i + 1}\right) \tag{4.31}$$

Multiplying these two component terms by the common denominator yields two new equations.

$$(u_i(gx_i + hy_i + 1) - (ax_i + by_i + c)) \tag{4.32}$$

$$(v_i(gx_i + hy_i + 1) - (dx_i + ey_i + f)) \tag{4.33}$$

These equations can be substituted in to equation 4.29, resulting in

$$E_{fit} = (u_i(gx_i + hy_i + 1) - (ax_i + by_i + c))^2 + (v_i(gx_i + hy_i + 1) - (dx_i + ey_i + f))^2 \tag{4.34}$$

It is important to note that equations 4.29 and 4.34 are not equivalent. When a set of transformed model points exactly matches the data points to which they are paired the value of E_{fit} will be zero in both cases. More commonly, there will be some difference between transformed model points and their paired data points and this will effect the value of E_{fit}. The process of estimating the optimal pose is in turn effected by this. However, restating a least squares problem in this fashion is a common mathematical technique (19).

The least square problem for the projective transformation problem may be compactly described as follows. Define p, the column vector representation of the projective transform P_p, as

$$p = | \; a \;\; b \;\; c \;\; d \;\; e \;\; f \;\; g \;\; h \; |^T \tag{4.35}$$

If equations 4.32 and 4.33 are assumed to zero, then they can be rewritten as

$$ax_i + by_i + c - gu_ix_i + hu_iy_i = u_i \tag{4.36}$$

$$-dx_i - ey_i - f + gv_ix_i + hv_iy_i = v_i \tag{4.37}$$

which allows them to be stated in matrix form as

$$X_i = | -x_i \quad -y_i \quad -1 \quad 0 \quad 0 \quad 0 \quad x_i u_i \quad y_i u_i | \tag{4.38}$$

$$Y_i = | 0 \quad 0 \quad 0 \quad -x_i \quad -y_i \quad -1 \quad x_i v_i \quad y_i v_i | \tag{4.39}$$

$$X_i p = u_i \tag{4.40}$$

$$Y_i p = -v_i \tag{4.41}$$

Each pairing between model and data points results in two constraints of the form given by equations 4.40 and 4.41, which gives rise to a least squares linear algebra problem. Two different methodologies for solving this problem are presented here, but it is important to emphasis that these techniques solve the same problem. When the relevant equations are expanded, it is possible to reorder the terms of one to fit the form of the other.

4.1.3.1 The Direct Linear Transform

Hartley and Zisserman present what is the standard technique for solving this least squares problem (19). They term this procedure for estimating the parameters of the projective transformation the Direct Linear Transform, or DLT.

Each of the k pairings between model and data in a correspondence C gives rise to two constraints as given in equations 4.40 and 4.41. These constraints form a matrix L with $2k$ rows and 8 columns, and a column vector B with $2k$ rows. The constraints can then be stated as

$$Lp = B \tag{4.42}$$

This is a classic least squares problem, and can be solved in a number of ways. Using the singular value decomposition, $L = UDV^T$. The matrices U and V are orthogonal matrices of sizes $2k \times 2k$ and 8×8 respectively. The matrix D is of size $2k \times 8$, with entries vanishing off-diagonal. These values are the singular values of L, and are the square roots of the eigenvalues of L. Given this equation 4.42 can be rewritten as

$$UDV^T p = B \tag{4.43}$$

$$DV^T p = U^T B \tag{4.44}$$

Let B' be the column vector $U^T B$ and y be the column vector given by $V^T p$. Equation 4.44 can then be written as

$$\begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_8 \\ \hline & \mathbf{0} & \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_8 \end{bmatrix} = \begin{bmatrix} b'_1 \\ \vdots \\ b'_8 \\ \hline b'_9 \\ \vdots \\ b'_{2k} \end{bmatrix} \tag{4.45}$$

The closest that Dy can come to b' is when $y_i = b'_i/d_i$ for the first eight enteries and $b_i = 0$ for the rest. Since V is orthogonal and $y = V^T p$, p can be found as Vy.

The advantage of using this method to solve the least squares estimate for p is that it can detect the case where the matrix L is singular. If, for instance, all k pairings fall on the same line then there are insufficient constraints to solve the problem, even if k is greater than the minimum of four pairs needed to uniquely determine the pose. In this case L is singular and the matrix D will contain one or more zeros on the diagonal. Unfortunately, obtaining the singular decomposition of L is computationally two expensive to be used in an iterative algorithm.

4.1.3.2 Fast Projective Pose Estimation

The standard version makes use of two constraints on each pairing, but an alternative formulation rewrites the problem so that there is one constraint on each equation.

Consider again equations 4.40 and 4.41. Assuming E_{fit} to be zero and using these equations to restate equation 4.34 gives

$$\left(\sum_i (X^T X + Y^T Y)\right) p = \sum_i u_i X_i^T + v_i^T Y_i \tag{4.46}$$

$$L = \sum_i (X^T X + Y^T Y) \tag{4.47}$$

$$B = \sum_i (u_i X_i^T + v_i^T Y_i) \tag{4.48}$$

$$Lp = B \tag{4.49}$$

This also represents a standard least squares problem which will yield the optimal projective pose. It should be noted here that the matrices L and B are different from those of the previous section. However, when both versions are fully expanded, it is possible to rearrange the terms of one to match the other.

The optimal pose can be found as $p = L^{-1}B$, using a simple matrix inversion. This risks the possibility of a singular matrix L, but in practice this seems to rarely, if ever, occur. Most important, this technique is approximately twice as fast as the standard method, in part because L is symmetric.

4.1.4 Determination of Degeneracy

A mapping C between M and D can be considered degenerate in several instances. A mapping may consist of too few pairs to uniquely determine the optimal pose. The similarity transform requires at least two pairings before a solution can be found, the affine requires three, and the projective transforms require four pairings. A mapping is considered to be trivially degenerate when this requirement is not met, and in this case S is arbitrarily large. Note that in this case a unique P^* can not be found, and the rest of equation 4.1 can not be evaluated.

When many-to-many mappings are allowed it is possible to minimize the distance between paired points by mapping all model points to a single data point and shrinking the model so that all points converge. Other extreme scale changes are equally undesirable. Perspective effects and large amounts of shear indicate unlikely or impossible poses. In the case of a projective transform, the optimal transformation for a mapping may represent one edge of the model being pulled back through the focal point of the camera. This occurs when the vanishing line for the transformed model passes through the bounding box on the data. All these cases are undesirable and must be avoided, and the term $S(P^*)$ serves to bias the search against them.

4.1.4.1 Degeneracy for Similarity Transforms

When the transformation class is limited to the similarity case there are no perspective and shear effects, reducing degeneracy to a measurement of scale. The scale change of a similarity transform is $Sc = \sqrt{a^2 + b^2}$ as discussed in section 3.4.2. If a scale change up to a factor of Sc_{allow} is allowed without penalty then

$$Sc = max(max(Sc, 1/Sc) - Sc_{allow}, 0) \quad (4.50)$$

$$S(P_s^*) = Sc \times Sc_w \quad (4.51)$$

In most cases $Sc_{allow} = 2$ is reasonable. The term Sc_w adjusts the penalty for scaling to relative to the size of the model. In a large model a penalty of $Sc = 1$ is equivalent to dropping one point from the match, and this may not be appropriate. Sc_w allows the scaling penalty to be kept in line with the size of the model. Experimentally $Sc_w = m/4$ seems to be a good value. This corresponds with a factor of one scale change beyond the allowed range incurring a penalty equal to failing to match one fourth of the points in the model.

4.1.4.2 Degeneracy for Projective Transforms

Determination of degeneracy in a projective transform in hampered by the fact that there is not a unique interpretation of a projective transformation with respect to the position of the two cameras viewing the scene. Thus, it is necessary to examine the effects of a projective transformation on the points it is applied to, and evaluate the pose on that basis. In this way an effective ad hoc degeneracy function can be obtained.

The degeneracy function for the projective case is composed of two terms. The first term, Sc, measures changes in model scale. The second term, V, biases the search against cases where there are extreme perspective effects.

Because the effects of shear and scaling can not be separated in the presence of projection, Sc is computed by examining the effects of the transformation of the model's bounding box. The transformation P^* is applied to the bounding box on the model, and the length of each resulting side, l, is measured. The largest scale change of the four sides is taken as the scale change for the pose, and a linear penalty similar to the one for the similarity case is applied.

$$lc_i = max(l_i, 1/l_i) \quad (4.52)$$

$$Sc_{max} = max(lc_1, lc_2, lc_3, lc_4) \times Sc_w \quad (4.53)$$

$$Sc = max(Sc_{max} - Sc_{allow}, 0) \quad (4.54)$$

There are two possible approaches to understanding the projection basis term. First, the parameters g and h are necessary to represent the effects of perspective on the model; and in the case where there is no out of plane rotation between a model and data then both are zero. This suggests g and h can be used as an indicator of perspective effects caused by out of plane rotation.

The relationship between out of plane rotation and g and h can be found empirically by examining how these parameters change as out of plane rotations are applied to a model with a known relation to the data. It appears that the quantity $\sqrt{g^2+h^2}$ forms a sine wave, peaking when the model is rotated ninety degrees out of plane, and having a value of zero when there is no out of plane rotation. Unfortunately, the magnitude of the sine wave is dependent on the focal length of the camera. As the focal length increases, the value of $\sqrt{g^2+h^2}$ drops. A low value may be the result of very little out of plane rotation, or a distant camera. Without knowing the focal length of the camera the value can not be directly related to out of plane rotation.

Alternatively, consider that a projective transform projects some points in the model plane to infinity. This occurs for points on the line $gx+hy+1=0$. Crossing this line results in a sign change in the coordinates of the transformed point, and creates a wrap around effect as points go from having a large positive value to a large negative one. When this line is far from the bounding box on the model this is not an issue. However, when the line intersects the bounding box, the result is a discontinuity in the transformed model which has no physical corollary.

Assuming the point set has been normalized for fitting as described previously, the bounding box will be centered at the origin. The distance from a point (u,v) to a line given by $ax+by+c=0$ is given by the formula

$$\frac{|av+bu+c|}{\sqrt{a^2+b^2}} \tag{4.55}$$

Thus the distance from the origin to the vanishing line can be found to be $\frac{1}{\sqrt{g^2+h^2}}$. When there is no out of plane rotation, and thus no perspective effects, g^2+h^2 will be zero, and this quantity will be infinity; fitting with the previous interpretation of $\sqrt{g^2+h^2}$.

The quantity V can now by formulated as

$$V=\sqrt{(g^2+h^2)}\times length_{bb} \tag{4.56}$$

where $length_{bb}$ is the length of one side of the bounding box. Thus V can be seen either as a

measure of out of plane rotation, or the relative distance from the origin to the vanishing line. For the projective case $S(P_p^*)$ is the sum of Sc and V.

4.2 An Alternative Problem Formulation

Previous work on point matching has often formulated point matching as the problem of finding a pose or range of poses, which place a maximum number of model points within the error bounds for the corresponding data. This is very different from searching correspondence space for some optimal set of correspondences and algorithms designed for one formulation do not adapt easily to the other. More problematic, solutions found by algorithms working under one formulation may only approximate those found by algorithms working under the other.

Bounded error formulations of the problem evaluate a pose or range of pose space based either on the number of model points which are placed within the error bounds of some data points (11) or on the total number of pairings possible under the given pose (8). In both cases the relative penalty incurred by any given pairing is binary, either the pairing is feasible and contributes to the match score or it does not. A transformed model point falling at the edge of its corresponding data points error bound is no better and no worse than one that exactly matches the data point. Algorithms designed to deal with this formulation of the problem search for *maximum feasible match sets*, correspondences such that every transformed model point falling within the error bounds of a data point is paired with that data point.

This stands in stark contrast with the combinatorial formulation presented here, where every pairing has associated with it a real valued error. This error is the squared distance between the transformed model point and its paired data point, divided by the value σ^2. Consequently a pairing with distance σ has an associated penalty of 1, which is the same penalty incurred for an unmatched model point. Conceptually this can be thought of as establishing an implicit circular error bound with radius σ on each data point. Although pairings which violate this error bound are allowed, an algorithm such as local search will quickly discard them, leaving the model point unmatched rather than incurring the high penalty associated with a distant pairing.

Consider an algorithm operating on the bounded error model which uses circular error bounds of radius σ and enforces a one-to-one constraint on the correspondence by always choosing the pairing

with the least distance between the transformed model point and data point. A correspondence which is a local optima in a one-pairing neighborhood is guaranteed to be a maximum feasible match set for the corresponding pose. No additional pair may be added to a maximum feasible match set because the distance between any unmatched model point and the nearest data point must be larger than σ or else the current correspondence would not be a local optima. A point can not be dropped either, because every transformed model point in the correspondence is within σ of its paired data point.

Unfortunately the reverse is not true. For a given pose the maximum feasible match set may not be a local optima. Local search considers adding, removing, or changing all possible pairs. Dropping a pair from the match set will result in a new pose which reduces the error incurred by every other pair; and may potentially reduce the degeneracy penalty $S(P^*)$. In certain extreme cases it is possible that this reduction in penalty is more than the the penalty incurred for omitting another model point from the match, and thus the reduced match set may be a better solution under equation 4.1. Adding a pair to the match may also potentially result in a significant change in pose, without significantly increasing the fitting error associated with pairs already in the match set. This is particularly likely to happen under the projective transform when all matching pairs come from the same spatial region of the data set. The projective transform has sufficient degrees of freedom that it is possible to find poses which provide a good match for some pairs in the optimal correspondence but not others. However, for correspondences of sufficient size, and with pairings distributed evenly across the the plane, it is highly likely that a maximum feasible match set also represents a local optima.

Chapter 5

Algorithms

5.1 Introduction

Four algorithms are of primary interest in this study. Random starts local search provides a robust and general solution to the point matching problem. This algorithm can be improved by using key features as the starting points for local search, thus increasing the chance that any particular trial results in the global optima being found. Breuel's RAST algorithm (7) is of interest because it represents a polynomial time solution to some classes of the point matching problem. The RANSAC algorithm is not polynomial, but can solve the projective point matching problem for a limited number of problem domains. Pose equivalence by Cass (11) is also of theoretic interest.

5.2 Pose Equivalence by Cass

Pose equivalence divides transformation space into a series of hyper-planes, each of which represents a maximum feasible mapping between model and data. Cass puts forward several algorithms which explore this space (11), the most general of which enumerates all possible pose equivalence classes and their associated mapping between model and data. This is the algorithm considered here.

Pose equivalence requires that each data point be associate with an error bound, most often in the form of a convex polygon with k sides. Each pairing between a model and data point forms k constraints which must be satisfied in order for a mapping containing that pairing to be feasible. Each constraint is a hyper-plane in t dimensional transformation space. Each place where $t-1$ of these hyper-planes intersect is a point in transformation space where the associated maximum feasible match set changes and represents another pose equivalence class.

To evaluate each pose equivalence class each model point is paired with each data point to create a set of *kmd* hyper-planes. Let $\hat{N}$ be the unit-length normal vector for the t dimensional hyper-plane which arises from one of the two dimensional constraints, i.e. the hyper-plane describing the constraints on the transformation space which must be satisfied so that the transformed model point meets the bounds placed on it by one side of the k sided polygon forming the error bounds on a data point. Let u and v be the position of point (x,y) after transformation, and ρ be the distance from the origin at which (u,v) must fall in order to occur on the line described by $\hat{N}$. Then, the hyper-plane representing this constraint can be expressed as

$$\hat{N} \bullet \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - \rho = 0 \tag{5.1}$$

There are $C(kmd, t-1)$ possible combinations of these hyper-planes which intersect at a point. Each possible combination must be considered, and the corresponding intersection determined. This is facilitated by representing each hyper-plane in a fashion similar to equation 5.1.

$$\hat{k} \bullet [P] - \rho = 0 \tag{5.2}$$

Equation 5.2 gives the canonical form of a hyper-plane, where $\hat{k}$ is the unit length normal to the hyper-plane, and P is a point in t dimensional transformation space. The variable ρ is the distance from the origin to the plane.

Let K be the matrix where each row is one normal vector describing a hyper-plane, and ρ is a column vector composed of the corresponding ρ variables. Let P be the point where the hyper-planes composing K intersect. Then

$$K \times P - \rho = 0 \tag{5.3}$$

$$K \times P = \rho \tag{5.4}$$

$$P = K^{-1}\rho \tag{5.5}$$

To compute the set of all possible pose equivalence classes enumerate all hyper-plane constraints and all possible $t-1$ combinations. Equation 5.5 gives the point where each combination of hyper-planes intersects. Each resulting transformation can then be applied to the model, and a simple plane

sweep algorithm used to check which model points fall within the error bounds of which data point. It should be noted that not every intersection of $t-1$ hyper-planes will result in a point; when two or more of the normals to a hyper-plane lie in the same plane the combination of hyper-planes is degenerate and the corresponding matrix A is not invertible. These cases do not need to be checked for a correspondence between model and data as they do not represent a new pose equivalence class.

Deriving the normal to a constraint hyper-plane from a model point and the corresponding error bound constraint is dependent on the class of allowable transformations. In the discussion that follows each constraint on the data error bounds is assumed to already be in the form given by equation 5.1, where $\hat{N} = [\ \alpha \quad \beta \quad 0\ ]^T$.

5.2.1 Similarity Transforms

When the class of allowable transformations is similarity transforms equation 5.1 becomes

$$\hat{N} \bullet \begin{bmatrix} ax+by+c \\ -bx+ay+f \\ 1 \end{bmatrix} - \rho = 0 \tag{5.6}$$

which can be expanded and re-arranged as

$$a\alpha x + b\alpha y + \alpha c - b\beta x + a\beta y + f\beta - \rho = 0 \tag{5.7}$$

$$a(\alpha x + \beta y) + b(\alpha y - \beta x) + \alpha c + f\beta - \rho = 0 \tag{5.8}$$

From this it can be seen that

$$k = \begin{bmatrix} \alpha x + \beta y \\ \alpha y - \beta x \\ \alpha \\ \beta \end{bmatrix} \tag{5.9}$$

$$\hat{k} = \frac{k}{|k|} \tag{5.10}$$

satisfies equation 5.2.

5.2.2 Affine Transforms

Extending the class of allowable transformations to the affine transform produces similar results. In this case equation 5.1 becomes

$$\hat{N} \bullet \begin{bmatrix} ax+by+c \\ dx+ey+f \\ 1 \end{bmatrix} - \rho = 0 \tag{5.11}$$

which can be expanded as

$$a\alpha x + b\alpha y + \alpha c + d\beta x + e\beta y + f\beta - \rho = 0 \tag{5.12}$$

From this it can be seen that the normal to the hyper-plane has the following form.

$$k = \begin{bmatrix} \alpha x \\ \alpha y \\ \alpha \\ \beta x \\ \beta y \\ \beta \end{bmatrix} \tag{5.13}$$

5.2.3 Projective Transforms

In the projective case equation 5.1 becomes

$$\hat{N} \bullet \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix} - \rho = 0 \tag{5.14}$$

This expands to

$$\alpha\left(\frac{ax+by+c}{gx+hy+1}\right) + \beta\left(\frac{dx+ey+f}{gx+hy+1}\right) - \rho = 0 \tag{5.15}$$

Multiplying both sides by $gx+hy+1$, this can be rewritten as

$$a\alpha x + b\alpha y + \alpha c + d\beta x + e\beta y + f\beta - g\rho x - h\rho y - \rho = 0 \tag{5.16}$$

Which gives

$$k = \begin{bmatrix} \alpha x \\ \alpha y \\ \alpha \\ \beta x \\ \beta y \\ \beta \\ -\rho x \\ -\rho y \end{bmatrix} \tag{5.17}$$

5.2.4 Implementation Difficulties

The pose equivalence algorithm as outlined here requires $C(kmd, t-1)$ intersections be considered. For moderately large points sets, or high dimensionality transforms such as the projective transform, this is an intractable search space. Cass suggests the use of an algorithm from computational geometry (14) that can give the arrangement of hyper-planes in order $k^t m^t d^t$ time. Unfortunately, this is not enough of an improvement to make the algorithm practical for the projective case.

The algorithms which Cass implements to create a practical system build on the observation that many poses are equivalent with respect to the points matched, but operate in a significantly different manner from what is described here. Although Cass suggests that his system can solve point matching problems where the points are related by a similarity transform, the algorithms presented seemed to be geared toward those where only translation and rotation need be considered. It is not clear how these algorithms might be extended to solve problems where point sets are related by a projective transform.

Lacking a practical pose equivalence algorithm which can be applied to the case of the projective transform, I will instead focus on the RAST algorithm, for similarity problems, and RANSAC for projective matching problems.

5.3 The RAST Algorithm

The RAST algorithm detailed in (8) searches pose space to determine the sub-region where the largest number of points may be matched. RAST begins with a bounded area of transformation space which is large enough that every model point may be taken to every data point by some transformation in the space. This region of transformation space is given as a hyper-cube with minimum and maximum values for each parameter of the transformation. A list of possible matches between model and data points is constructed, and initially this list contains each of the md possible pairings.

At each stage in the search, transformation space is subdivided and evaluated. In (8) Breuel suggests splitting the transformation space in half along the largest dimension to produce two new search states. The sample implementation provided at the end of this paper takes a different approach and splits every dimension in half, combining each half of one dimension with each of half of the others to produce 2^D new search states.

Each region of transformation space, and the corresponding search step, is evaluated on the basis of those model points it may potentially match to data points. The center of the region is determined, and this transformation is applied to every model point. For each possible correspondence the distance between the transformed model point and the data point is determined. The value Q denotes the number of correspondences where the transformed model point falls within the error bounds of the data point. Those that fall outside of the error bounds may still be close enough that some transformation in the region under consideration will take them to within the error bounds. RAST determines this by projecting the region of transformation space onto the space in which the data lies in the form of a circular error bound, centered on the data point. Any model point falling within this bound is considered a potential match. The number of definite matches, plus potential matches, is denoted $\hat{Q}$.

The formulation of this circular bound relating transformation space to image space is key to Breuel's work, and he presents formulations for several transformation classes. Provided that the mapping between the two spaces can be determined, the RAST algorithm is capable of solving the point matching problem in polynomial time for any transformation class. Equation 5.18 gives the error bounds for the similarity case, equation 5.19 gives the bounds for the general affine case. The notation x^D denotes the magnitude of the region along a given dimension; alternatively the length of the hyper-cube along dimension x.

$$\sigma = \sqrt{(\alpha^D|x_i| - \beta^D|y_i| + t_x^D)^2 + (\beta^D|x_i| + \alpha^D|y_i| + t_y^D)^2} \tag{5.18}$$

$$\sigma = \sqrt{(a^D|x_i| - b^D|y_i| + c^D)^2 + (d^D|x_i| + e^D|y_i| + f^D)^2} \tag{5.19}$$

Rather than evaluate all md possible model to data pairings at each step, RAST maintains a list of those model to data points which may potentially match for a given region of transformation space. Each new sub region inherits this list of potential matches, only those pairings which could possibly match under a transformation in the larger parent region need to be evaluated. By maintaining this list of potential matches the evaluation of incorrect matches can be avoided, resulting in a significant savings in the time required to evaluate each region of the transformation space.

Search is prioritized based on $\hat{Q}$, with those regions having higher values of $\hat{Q}$ being searched first. Regions which have a value of $\hat{Q}$ lower than the best found Q can be pruned from the search

space. This heuristic is admissible, and the RAST technique is a good candidate for A* search. Unfortunately, the memory required for such a search is prohibitive, and in practice a depth first search is preferred.

5.3.1 Extending RAST to the projective transform

Breuel develops RAST for a number of transformation classes but not for the projective case. In theory extending RAST in this direction should be straightforward. The algorithm is very general and if an equation similar to equations 5.18 and 5.19 could be found for the projective transform the algorithm should operate in this space with little difficulty. The size of the search space, however, remains an issue.

The eight degrees of freedom provided by the projective transform allow for a large number of transforms to take any given model point to any given data point. Under the projective transform, transformation space must be subdivided more times than under the similarity transform before potential matches begin to drop out. Consequently, RAST may have to search a large portion of pose space before regions begin to differentiate on the basis of total possible matches. This would lead to a significant increase in run times. In the extreme case the algorithm degenerates into a Hough transform, evaluating a large number of very small partitions of pose space which differ only marginally.

The projective case is further complicated by the fact that with eight degrees of freedom it is possible to map a large number of model points to data points in a fashion which produces a highly unlikely or impossible pose. If there are a large number of missing or spurious points it is possible that RAST will favor a very poor transformation, pruning away the portion of the search space containing the desired match in favor of a portion of transformation space which causes more pairs to match. Even if this does not occur, the freedom allowed by the projective transform can cause RAST to spend significant time searching portions of pose space which contain poor solutions.

In Breuel's work RAST relies on orientation information attached to each point to significantly prune the search space early on and such an approach may be helpful when dealing with the projective transform. However, this information is not available in all problem domains.

5.4 Random Starts Local Search

Local search is a straightforward search technique which requires only the ability to assess differing solutions to determine their fitness relative to one another (25). Some definition for adjacency of solutions is given, defining a neighborhood of solutions similar to a given solution. Starting with an arbitrary initial solution the search then proceeds to check the neighborhood for an improvement on this solution. If one is found, then the search moves to the new solution and repeats, searching the neighborhood of this improved solution. When an improvement can not be found the search terminates. One iteration of random search can be completed quickly but is unlikely to yield an optimal solution. When multiple trials are run the probability that all trials will fail drops exponentially (25). If the chance of any individual trial failing is F_i, then the chance that all t trials will fail is $F_a = F_i^t$.

The natural representation for a solution in many-to-many correspondence space is a bit string of length 2^{md}, with each bit representing the presence or absence of a pairing between a point in the model and a point in the data. This lends itself to a neighborhood where two solutions are adjacent if they differ by only a one bit toggle; one pairing added or removed from the match. Using equation 4.1 to evaluate the fitness of a given solution correspondence space can be searched for a solution from an initial random solution. Greatest improvement local search is used, finding the best solution in a neighborhood and moving to that position to start the next round of search, rather than simply taking the first improvement found.

In many domains a many-to-many mapping is not required to obtain a correct or approximately correct mapping from model to data points. Often a one-to-one mapping is sufficient. For this reason it is often better to use an algorithm which only considers this restricted space. In such a space the neighborhood can be redefined to be all those solutions where one model point maps to a different, previously unmapped model point or the the null point. With such a definition local search can be applied to a one-to-one search space. If necessary, a solution found in this fashion can be used as the starting point for a round of many-to-many local search. In the experiments presented here the mapping used is straight one-to-one, without the optional many-to-many phase.

The utility of local search is based on its ability to quickly asses a large number of potential solutions. Solving equation 4.1 requires finding an optimal pose, and in the case of projective transforms

this requires a normalization of both the model and data sets; and a corresponding denormalization, of the computed pose to relate the pose back to the original point sets. This normalization, and de-normalization, of every pose considered is time consuming. Consequently, the model and data sets are normalized prior to the beginning of local search. Local search then searches for the best correspondence between these normalized point sets. Only after an optimal solution is found is the corresponding pose de-normalized to and reported as the final best pose found.

The normalization process involves a change of coordinate systems and a scale change. This process gives stability to estimates of the optimal pose, preventing the pose from changing radically as pairs are added and removed from a solution. However, the scale change does effect the parameters of the search. The parameter σ gives the maximum distance between model and data points in the space of the original data set. In order to preserve this meaning for the normalized point sets, σ must be scaled by the scale change on the data set. Evaluation of $S(P^*)$ also requires determining if the scale change of the transformation is within some predetermined bounds; again this bound is stated as the scale change between the original data sets. The solution is to apply the relative scale change between the two normalized point sets to the computed scale before determining if the scale change is within bounds. These changes can be done transparently by the system, mostly during algorithm initialization. Point matching is then done between the normalized point sets.

The effectiveness of local search is greatly influenced by the choice of starting points. A truly random solution is unlikely to lead to a global opt. As previously noted, for many point matching problems a one-to-one solution is a good approximation of the final correct answer. This approximation greatly reduces the number of possible initial starting points without greatly reducing the number of starting points which lead to a global opt. A one-to-one solution also avoids such pathologies as having a large number of points map to a single point, or leaving many model points unmapped. For this reason, random starts are determined by randomly selecting data points to map to model points in a one-to-one fashion; with each data point having an equal chance of being selected for pairing with a given model point.

Size of the initial match can also greatly effect the chance of a given trial succeeding. Consider a fully instantiated one-to-one match, where every possible model point is matched to one data point and all but two model points are paired to the correct data points. These two model points are paired

with each other's data points. In order for local search to find the optimal match it must take at least three steps, first dropping one bad pairing, switching the other model point to its correct data point, and then matching the first model point to the newly available, correct data point. At each step the corresponding pose of the model is incorrect making it unlikely that local search can make these three steps. Worse, as the number of pairs in the initial solution increases the chance that a bad pairing will enter into the solution increases.

Experimentally, it appears that an initial match size of five is a good choice for random starts local search. This is enough pairs that one potentially bad pairing can be dropped while preserving the ability to uniquely determine model pose.

5.5 Heuristic Starts Local Search

The efficiency of local search is determined by how often a starting point falls within the basin of attraction for the global optima. It seems reasonable to bias the selection of starting points in favor of those which might contain as least some of the pairings in the global optima. If only a partial solution is known, then local search might be reasonably expected to fill out this partial solution with the missing pairings at least most of the time. Unfortunately, it is not possible to generate a solution which is known to be a subset of the pairings in the global optima. However, it is possible to bias the starting points function to produce correspondences which are likely to be in the basin of attraction for the global optima.

The approach taken here is to construct key features in each point set, and then pair these key features in all possible combinations to create a ranked list of starting points likely to be partial solutions. Local search can then be started from these key features instead of from random points in the search space. The key feature algorithm proposed here is a modified version of the algorithm Beveridge, Graves, and Steinborn used for line matching (6).

Key features are defined as groups of model or data points, which have been paired based on the proximity of those points to a central or key point. This follows the intuition that points which are close to each other in the model can reasonably be expected to be close to each other in the data. Each possible pairing between key model and data points becomes the key pairing for a number of key features. A feature is composed of the key pairing and the z pairings formed by pairing the z

model points closest to the key model point with the z data points closest to the key point in that set. Every possible combination of model to data points is considered as a key pair, and every possible combination of the z closest points in each point set is considered to produce a list of key features of size $z!mn$. Figure 5.1 provides a graphical example.

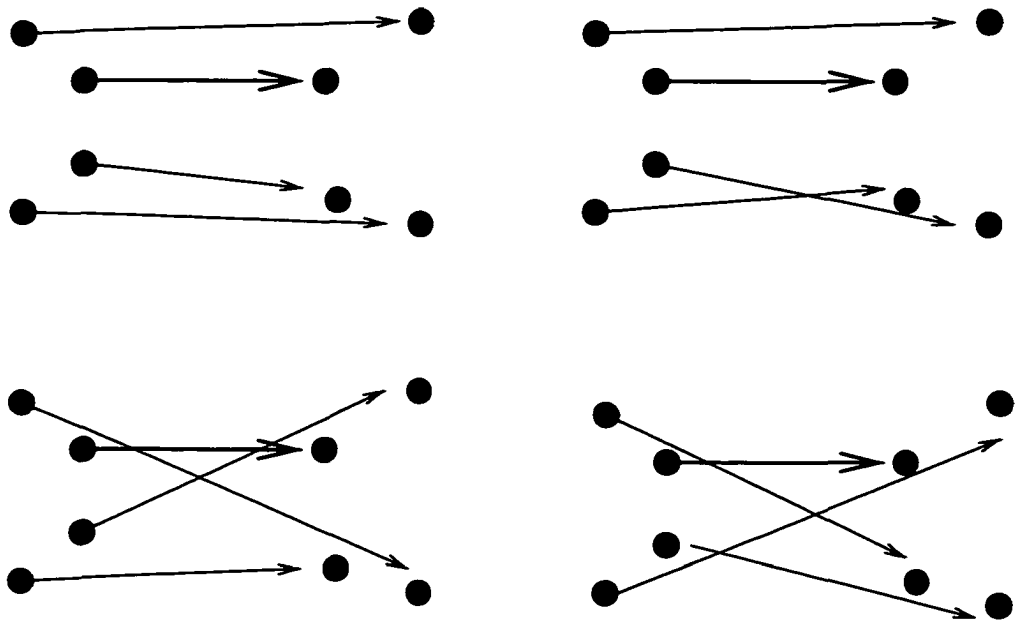


Figure 5.1: Example of key feature construction

Consider a second example. Suppose $z = 3$ and the model set contains points $m_0 - m_3$, such that the three closest points to m_0 are m_1,m_2, and m_3. The data set contains points $d_0 - d_3$, where d_1, d_2, and d_3 are the closest points to d_0. If m_0 and d_0 are paired to form the key pairing, then there are six key features containing this key pairing.

$\{(m_0,d_0),(m_1,d_1),(m_2,d_2),(m_3,d_3)\}$
$\{(m_0,d_0),(m_1,d_1),(m_2,d_3),(m_3,d_2)\}$
$\{(m_0,d_0),(m_1,d_2),(m_2,d_1),(m_3,d_3)\}$
$\{(m_0,d_0),(m_1,d_2),(m_2,d_3),(m_3,d_1)\}$
$\{(m_0,d_0),(m_1,d_3),(m_2,d_1),(m_3,d_2)\}$
$\{(m_0,d_0),(m_1,d_3),(m_2,d_2),(m_3,d_1)\}$

If the key feature algorithm is to succeed it must successfully locate a key feature with enough correct pairings to uniquely determine a pose which approximates the true pose of model in the data. In the case of the projective transform, this means a key feature with at least four correct pairings. If there are a large number of spurious points surrounding each point in the correct match this becomes difficult. In order to prevent this disruption by spurious points it is important that the size of the key

features be kept small, while still providing enough points so that the model pose can be uniquely determined when features are paired. In the case of the similarity transform, only two pairs of points are required, so $z = 1$. Affine transformations require three pairs of points, so $z = 2$. The projective transform requires four pairings to correctly determine the optimal model pose, so in this case $z = 3$.

Experiments suggest that it is helpful for local search to be able to drop a potentially bad pairing from a key feature immediately, and so in practice z is set to 4 for the projective case. This creates some tolerance for spurious or missing points, enabling such points to participate in key feature creation without greatly damaging the ability of the search algorithm to overcome such problems.

When $z = 4$ the key feature algorithm constructs $24md$ initial key features, to many to consider when dealing with large point sets. All of these key features incur the same omission penalty from equation 4.1, but because they contain one more point than is necessary to exactly determine the optimal pose, they also contain some fitting error. Many key features may also be degenerate in some fashion, incurring a large penalty in the $S(T)$ term of equation 4.1. Such features can be immediately discarded. The remaining features are ranked according to fit, and some portion of this list is used as the starting points for local search. In practice only a small portion of this key feature list needs to be search. If the algorithm is working, then this should be sufficient to find the global optima, if it is not there is unlikely to be a successful key feature on the list.

5.6 RANSAC

The RANSAC algorithm presented by Hartley and Zisserman (19) is capable of solving the projective point matching problem for a limited class of problems, and previously has been the best available method for solving projective point matching problems. RANSAC begins with four correspondences chosen at random, or on the basis of some heuristic. The implementation discussed here uses the same key feature algorithm as heuristic starts local search to create key features which contain a total of four pairings.

The four initial correspondences are used to produce an estimate of the pose, using the method outlined in chapter 4. The model is transformed by this pose, and a new set of correspondences is constructed by taking all those pairs where the model is less than some σ from a data point. Although RANSAC is generally formulated for many-to-many matching, this is often not required and can

mislead the algorithm. Consequently, the new set of correspondences is constrained to be one-to-one. This is accomplished by adding the pairs in the order of the distance between the transformed model and data points; skipping those pairings where one point is already matched. The resulting set of correspondences is used to recompute the pose, and the process repeats until the correspondences stabilize. Note that it is possible for pairings to be dropped during this processes, if later estimates of the pose take the distance between a paired model and data point beyond σ, although this is unlikely.

Chapter 6

Data Sets Used

6.1 General Methodology

A great many point matching algorithms have been validated on synthetic data that has been created based on some statistical model. This provides a good way to quickly test new algorithms and evaluate their performance under very specific conditions, but rarely provides an accurate indication of how such algorithms will work on real data. To avoid this problem the experiments here use points extracted from real imagery.

Most of the images used in this study, with two exceptions detailed below, where taken specifically for this study using a digital camera. These images where shrunk to half their original size to create an anti-aliasing effect and then converted from color to 256 value gray-scale. Originally, the SUSAN point extractor (27) was used extract corners from these images, but it was found that the feature extractor from the Intel OpenCV library (23) produced more accurate results. The SUSAN program is also capable of a smoothing operation, and this was applied to the images before the OpenCV library was used to extract points.

6.2 Data Sets for Image Registration

Six data sets where used to test the ability of local search to solve the projective image registration problem. Each data set consisted of a number of images of the same object or scene, taken from a variety of camera angles. Table 6.1 shows the size of each data set, figures 6.1-6.6 show the actual data sets used for image registration.

Table 6.1: Size of Image Registration Data Sets

Imagery	Image no.	# of Points	Imagery	Image no.	# of Points
Polygon	1	9	Picture	1	16
	2	9		2	15
	3	12		3	15
	4	10		4	14
	5	10	Java Book	1	51
Jacob's	1	89		2	50
	2	87		3	53
	3	69		4	47
	4	149		5	39
Poster	1	119		6	45
	2	128	Ft. Hood	1	65
	3	97		2	67

6.2.1 The Polygon Data Set

The polygon data set consists of five images of an irregular hexagon printed in black on white paper. The extracted points were the six points on the hexagon, and the four corners of the sheet of paper; although three data sets actually contained more or fewer points for various reasons. The points extracted from this imagery are remarkable free of clutter and noise, and consistent from all viewing angels. Consequently, this data presents point matching problems which are extremely easy. The major difficulty in matching points between images in this set comes from the potentially large scale changes between the different images.

6.2.2 The Picture Data Set

The picture imagery shows a framed, abstract, painting hanging on the wall. The exterior frame provides four points, with another four from the border between the actual painting and the matting. These eight points form an eightfold symmetry, which can potentially confuse a point matching algorithm. The interior of the painting provides the remaining points in all the images, with most of the points towards the top of the painting. This breaks the symmetry, providing a group of points with which the point matching algorithm can distinguish between correct matches and those representing poses corresponding to one of the symmetries. Like the polygon data set, there is very little noise in the location of each individual point.

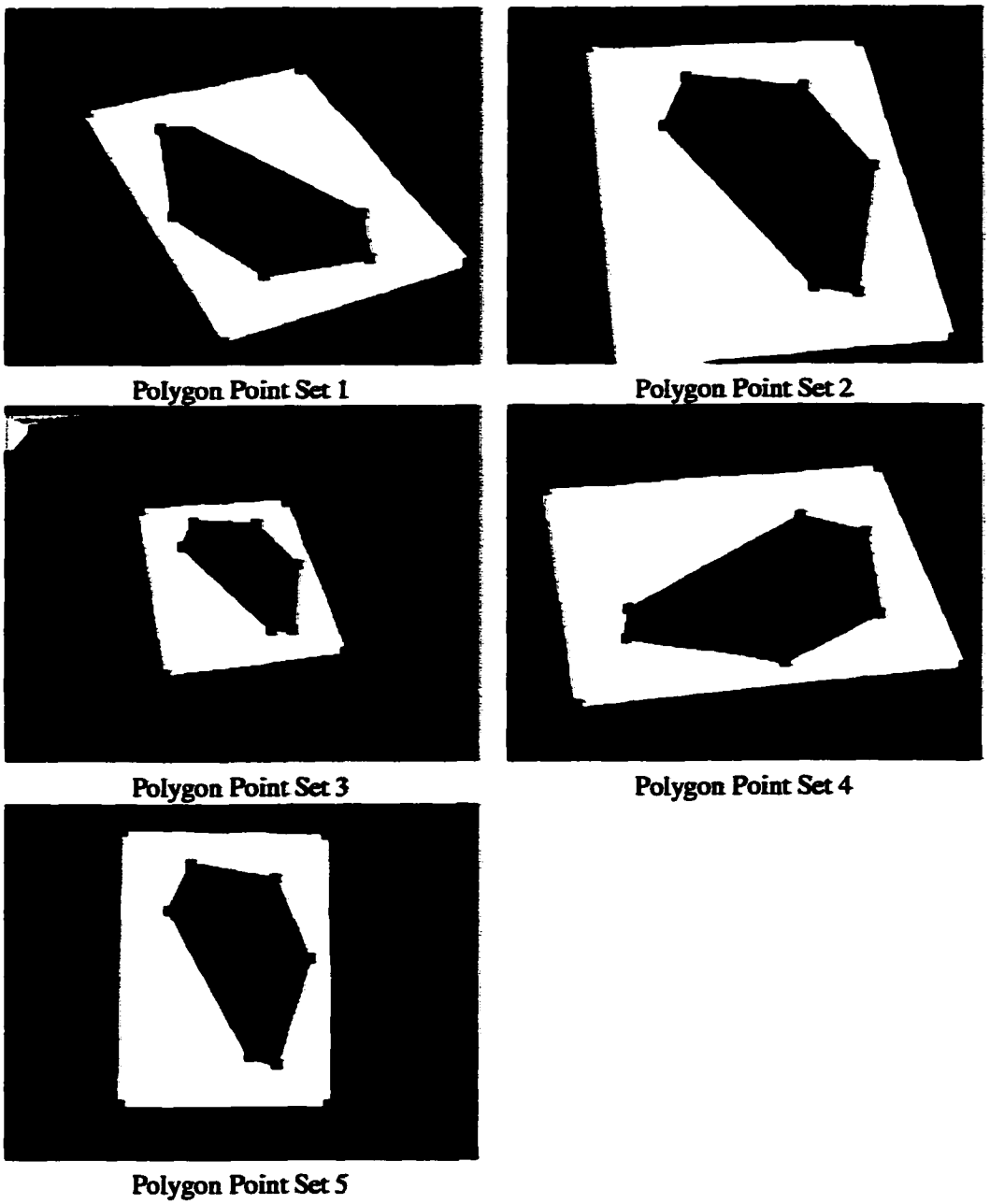


Figure 6.1: The Polygon Data Set

It is important to note that every image of the picture produced the same 14 points, out of no more than sixteen points total for the largest data set. This has an important simplification of the point matching problem. When matching from one picture data set to another there are never more than two missing or spurious points, far fewer than the fourteen correct matches. Even when there are missing or spurious points they are a good distance from the next closest point in the point set, resulting in data which contains, for practical purposes, no clutter. The lack of clutter makes this problem substantially easier.

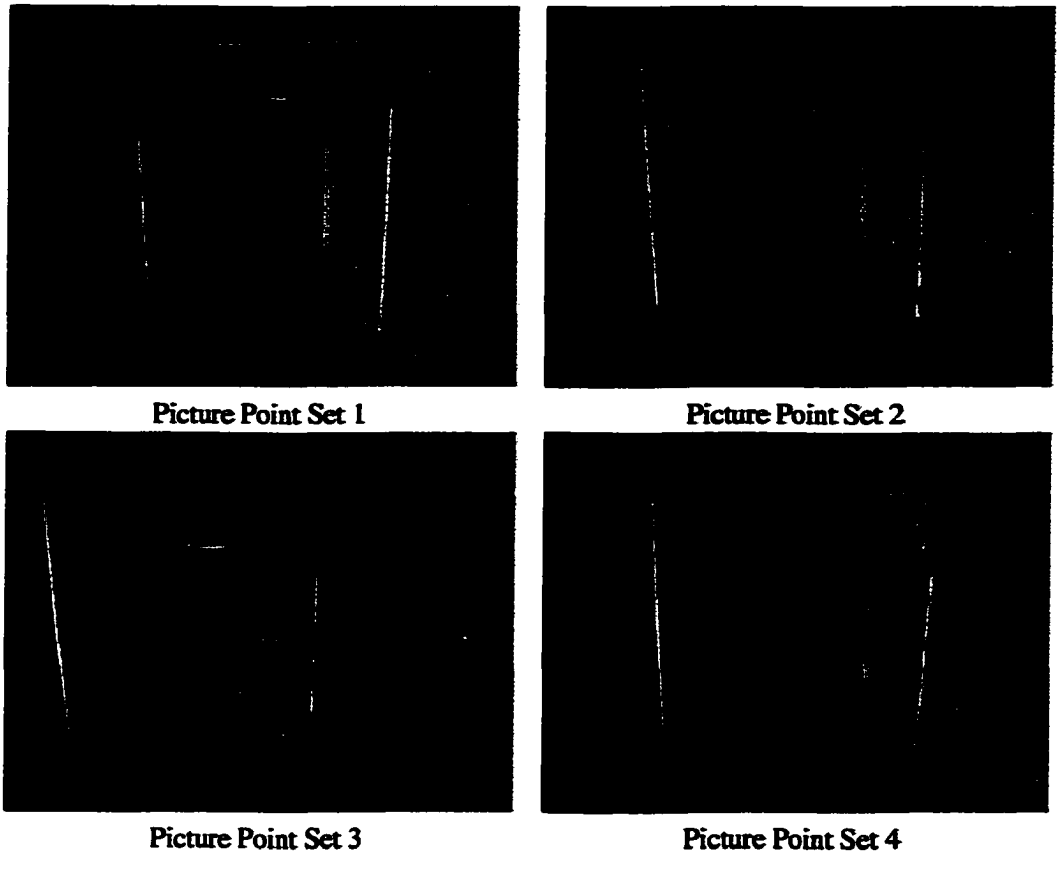


Figure 6.2: The Picture Data Set

6.2.3 The Java Book Data Set

The images taken of the Java book provide the first practical test of a point matching algorithm's ability to deal with missing and spurious points. These images are of a rather thick text on Java programming, with the points of interest to the point matching algorithm coming from the front cover. Although these images where produced with the same process as the others, there are qualitative

differences. The corners in this imagery are significantly less stable, and as the angle changes some corners are lost and others appear. The exact location of each corner in the image is also susceptible to change, resulting in a great deal of noise in each point location. The size of the data set presents further problems, with many points packed into a relatively small space. The result is that data taken from this imagery has a great deal of noise, and a large number of spurious and missing points. This makes problems from the Java book data set some of the most difficult to solve.

6.2.4 The Poster Data Set

The poster imagery provides the largest data sets in this study. Like the book data, there is noise in the location of each individual point. This is a more significant problem in this data set because the large number of points means that most points are close to other points, increasing the potential for point matching algorithms to be confused. This data set represents the hardest image registration problems studied here.

6.2.5 Fort Hood Aerial Photographs

One of the potential applications for point matching is registration of ariel images, and the Fort Hood data is a well known data set of this type. These images where originally taken as part of the government sponsored RADIUS project (16) and have been made widely available to researchers. Most of the images are related, or appear to be related, by only translation and scale. However, some are taken from different angles. Because the images are extremely large, a sub-tile was selected from each of two images, showing the same general area from different angles. Points where extracted from these images using the Intel OpenCV library.

The Ft. Hood data sets contain a cluster of points from a group of buildings in one part of the image, and a few distinct points in the other. The location of each point is fairly accurate, but there are a number of missing and spurious points between the two images, making exact matches between points in the clusters problematic.

6.2.6 Data Provided by Jacobs

Basri and Jacobs recently published a paper dealing with projective alignment based on regions (4), and generously made the imagery used in that paper available. These images consist of four different

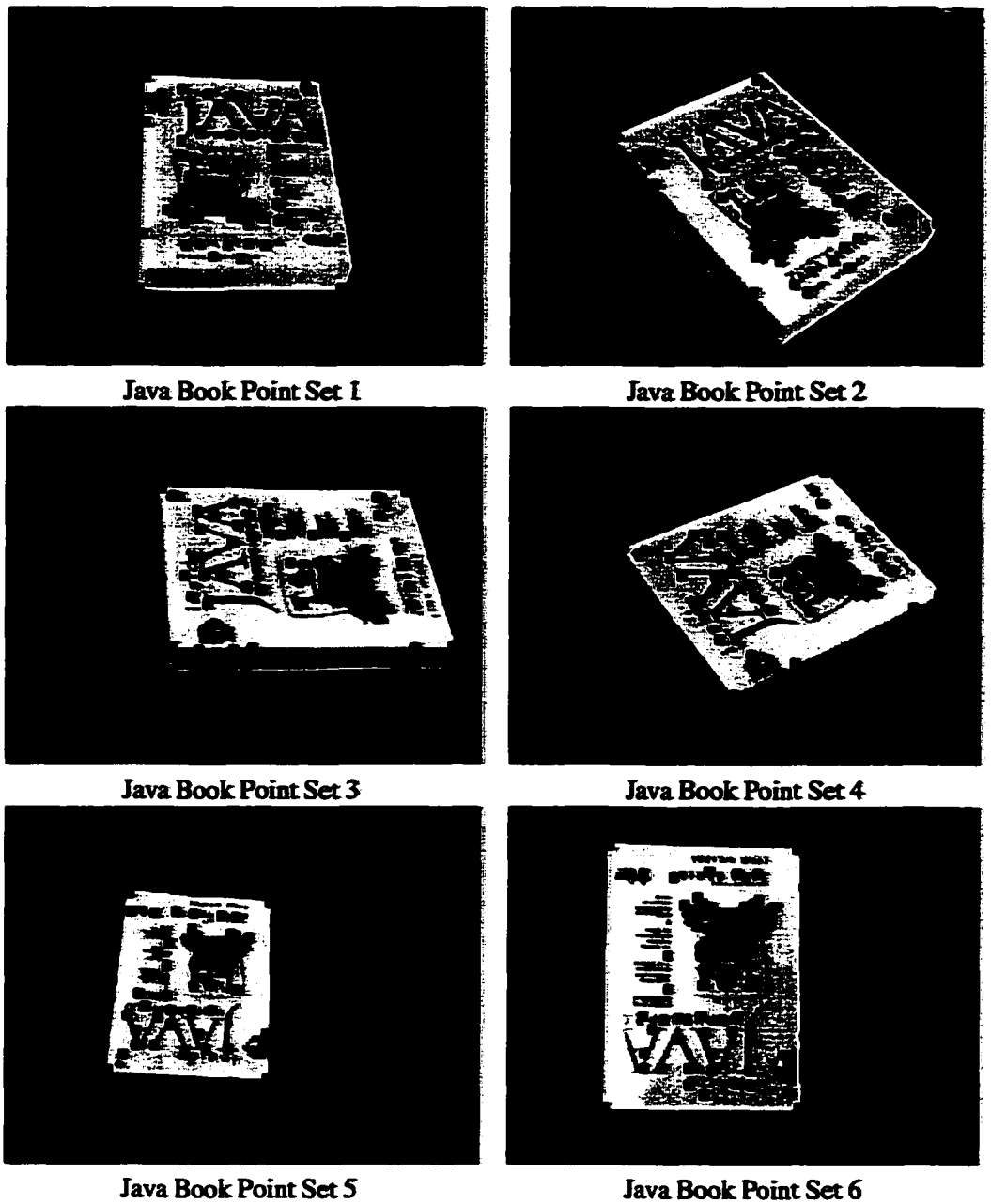
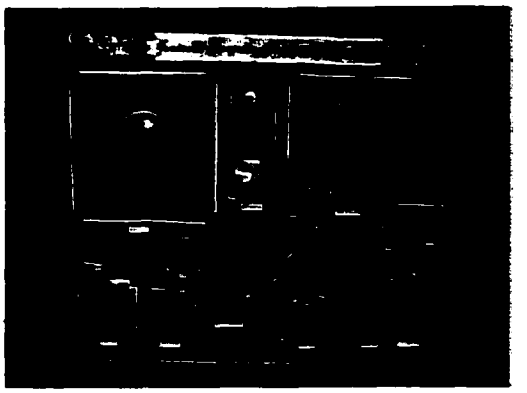
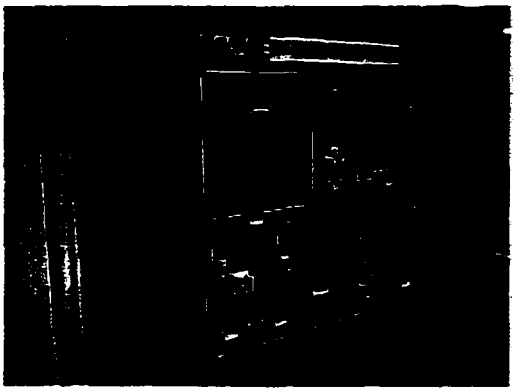


Figure 6.3: The Java Book Data Set

Poster Point Set 1



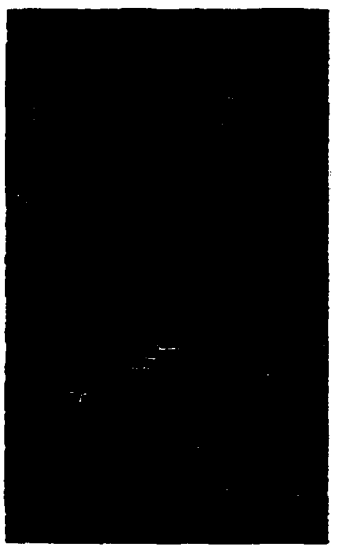
Poster Point Set 2



Poster Point Set 3

Figure 6.4: The Poster Data Set

Ft. Hood Point Set 1



Ft. Hood Point Set 2

Figure 6.5: The Ft. Hood Data Set

views of another abstract painting, one of which is obstructed by the presence of other objects.

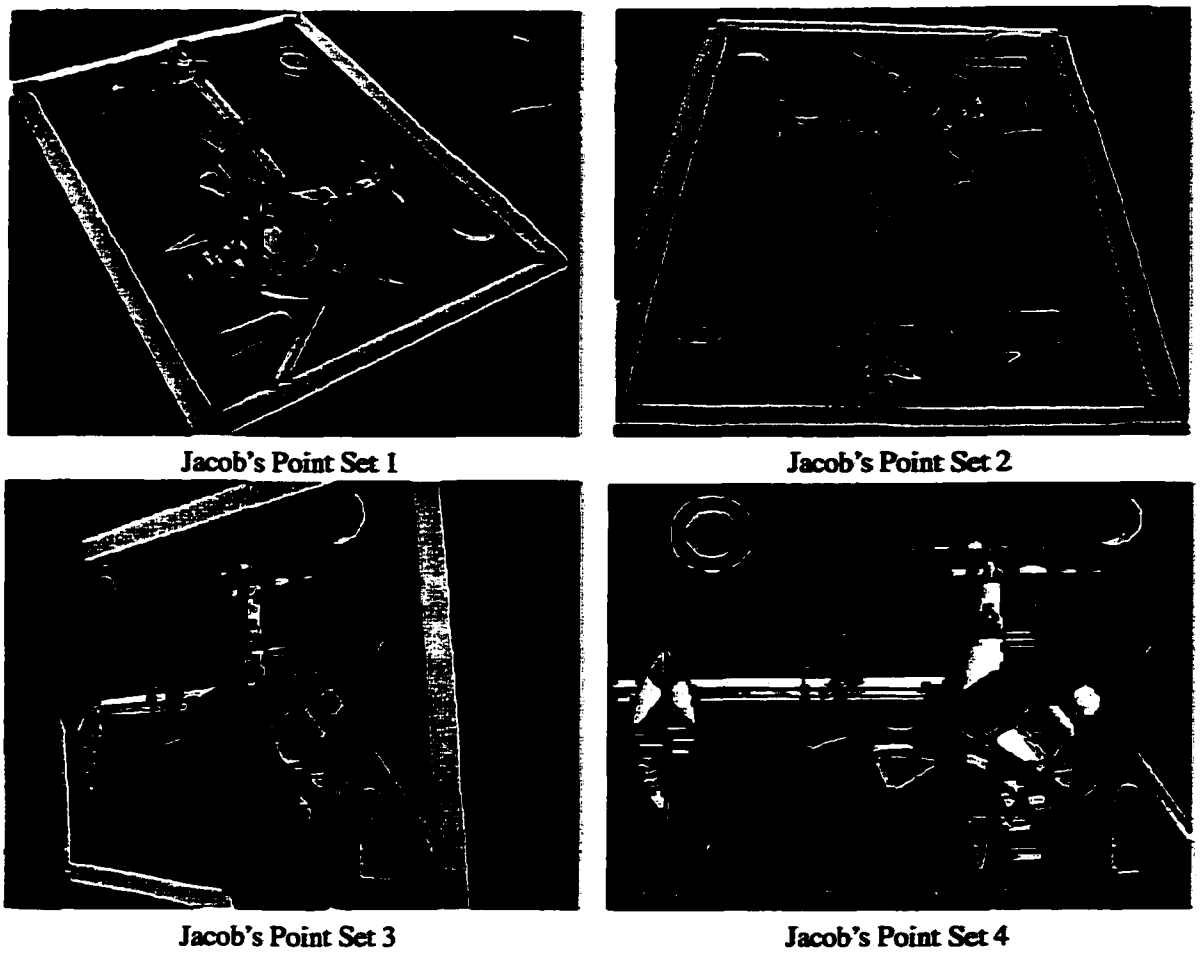


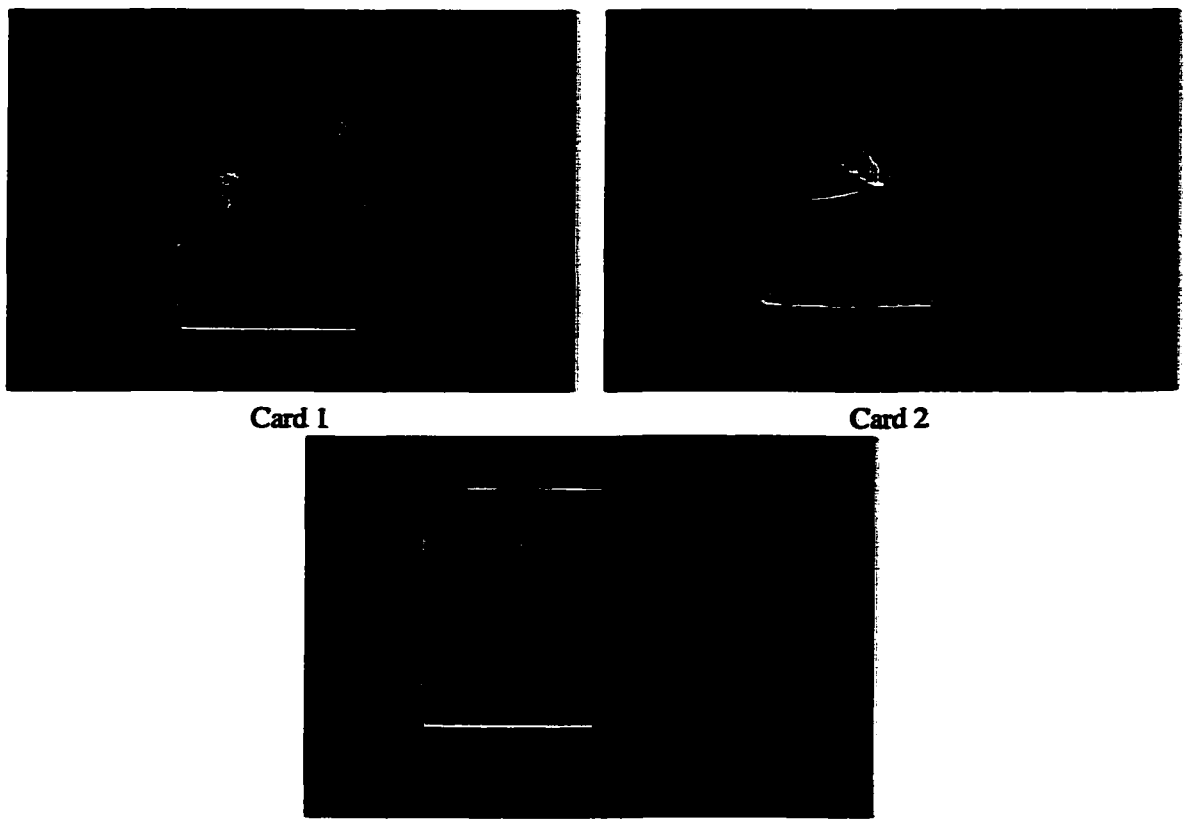
Figure 6.6: The Jacobs Data Set

6.3 Data Sets for Model Recognition

Unlike image registration, model recognition requires two distinct point sets. One set for the object model, and one for the data in which the object model instance is to be found. Data for model recognition problems comes from images of Christmas cards. Images of single cards provided the object models, and collections of cards provided the data sets. That nature of cards two and three made point extraction difficult, the images on these cards had many soft or blurry edges. As a result, they often produced many points where points would not normally be expected when imaged from up close. When imaged from further away these points could not be detected.

6.3.1 Object Models

Three Christmas cards, shown in figure 6.7, where used as object models. These where photographed straight on, looking down. Points where extracted from each image using the methodology already described.



Card 1 Card 2

Card 3

Figure 6.7: Object Models for Model Recognition

6.3.2 Data Sets

Data sets to be matched where obtained by taking pictures of collections of Christmas cards from various angles. Each collection included images from three different angles. Collection one, shown in figure, contains three instances of card one and one instance of card two to test the ability of local search to find multiple model instances. Collection two contains two images of card two. Collection three contains one instance of each card. The cards in collection four lie on top of each other, making two of the model instances partially occluded. Collection five contains one instance of each card, with a very small amount of occlusion on card one. Collections one and two are shown in figure 6.8,

figure 6.9 shows collections three and four, and collection five appears in figure 6.10.

6.4 Data Sets for Similarity Transform Problems

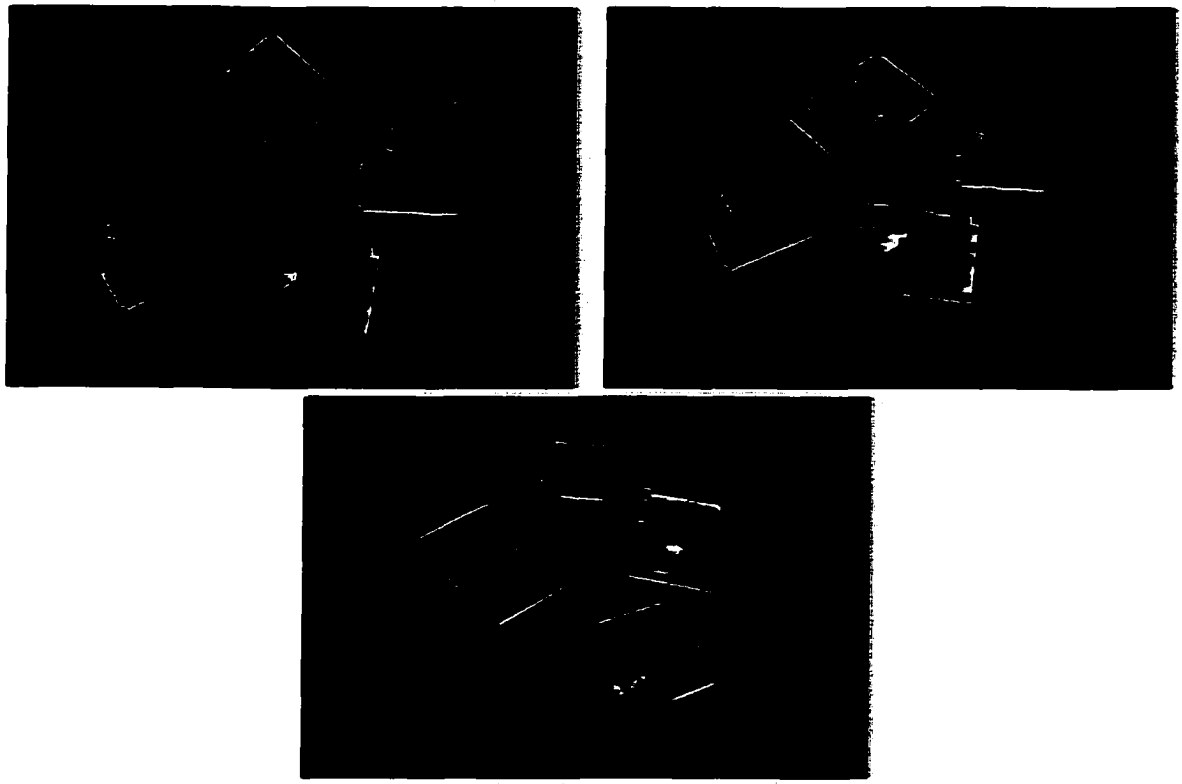
Previous work on solving the point matching problem has often focused the class of similarity transforms, and comparison between local search and this previous work requires that some data be restricted to this case. Fortunately, some of the previously collected datasets are related by poses which closely approximate similarity transforms. Using these data sets to test the ability of local search to solve similarity problems provides the added benefit of giving some idea how solutions for more restricted transformation classes may be used to approximate and guide local search towards the correct solution in higher dimensional cases.

A similarity transformation closely approximates the relationship between data sets three and four from the book imagery, the first two images from the first two Christmas card collections, and the last two images from the third Christmas card collection. These data sets were used for similarity experiments.

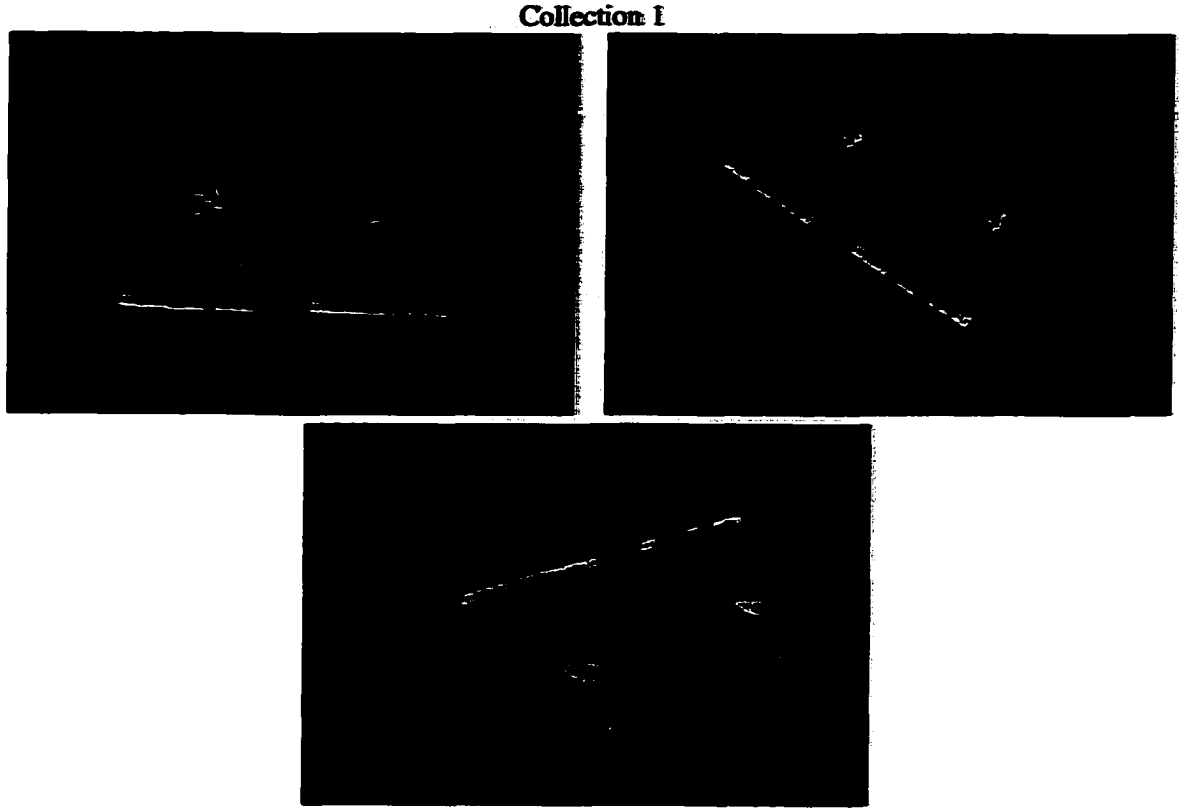
6.4.1 Rigid Body Data Sets

In (8) Breuel provides an implementation of RAST for the rigid body problem written in Java. In order to facilitate a direct comparison between local search and RAST, data for experiments using this limit class of transforms was collected. This data came from four images of a book on UML, with points extracted in the usual manner.

Because the performance of RAST can be expected to suffer less in the presence of clutter than the key feature algorithm, four additional data sets where created by adding clutter to each of the four original data sets. This clutter consisted of a number of new points equal to the number of original points, randomly spread through the area of the bounding box on the original data, plus 10% on each side. Figure 6.11 shows both the original data sets and the cluttered versions.

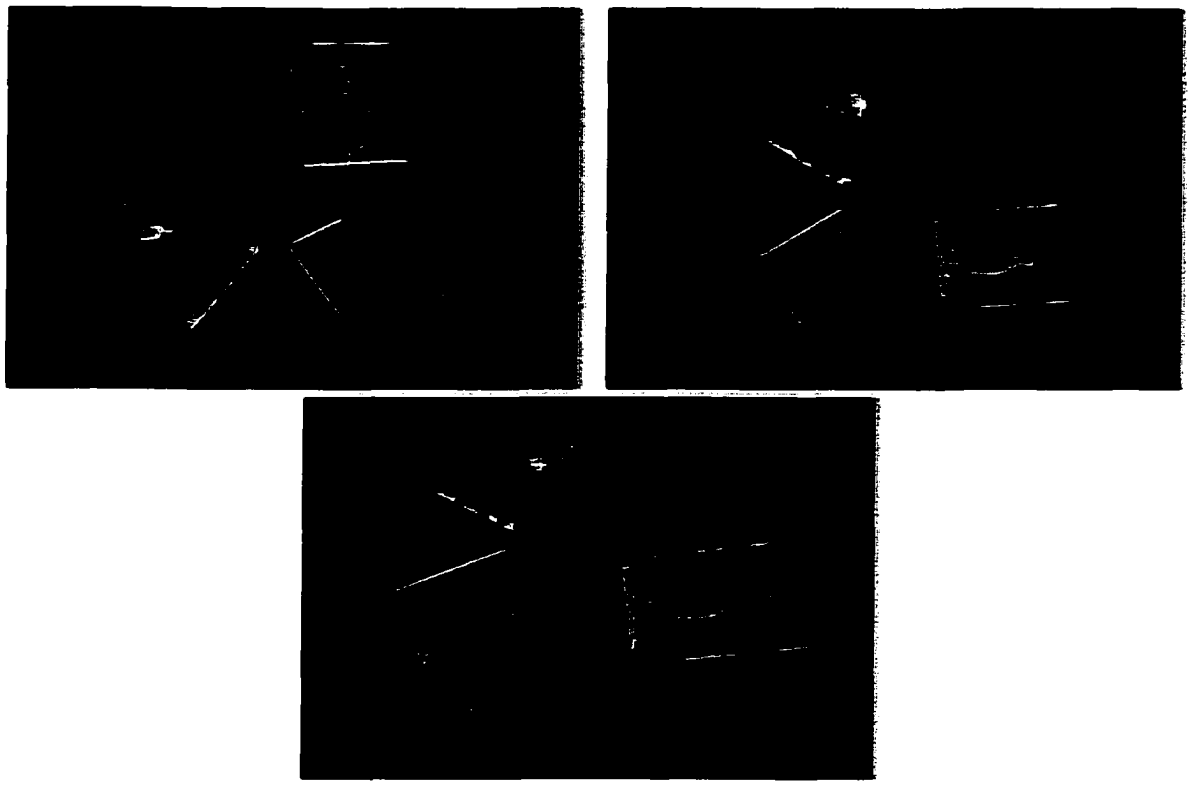


Collection 1

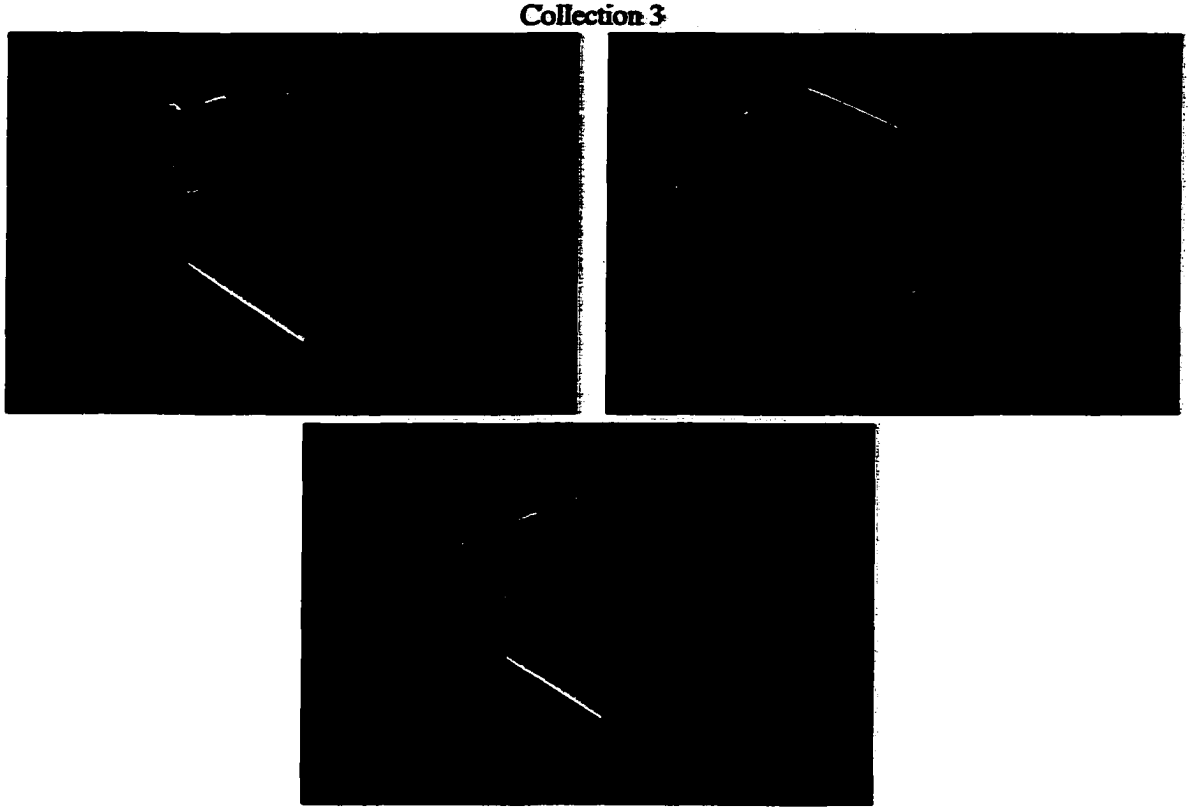


Collection 2

Figure 6.8: Data Sets for Model Recognition, Part 1

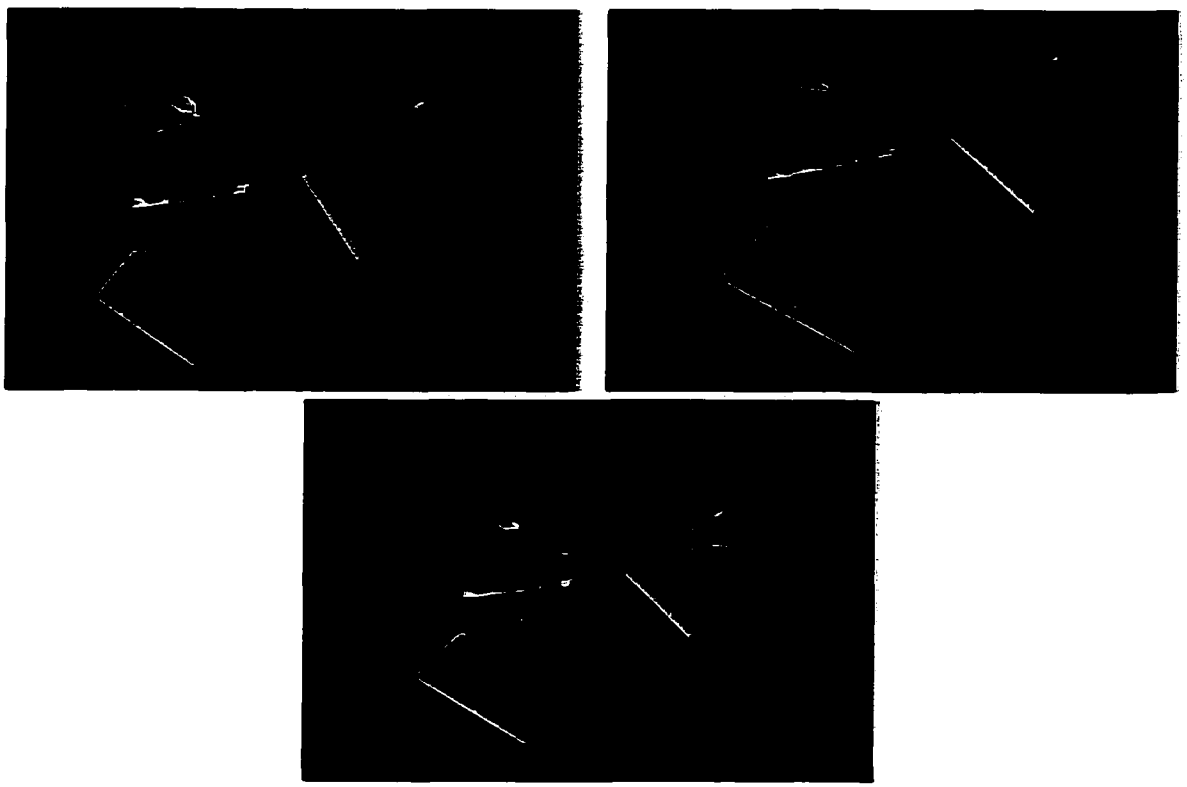


Collection 3



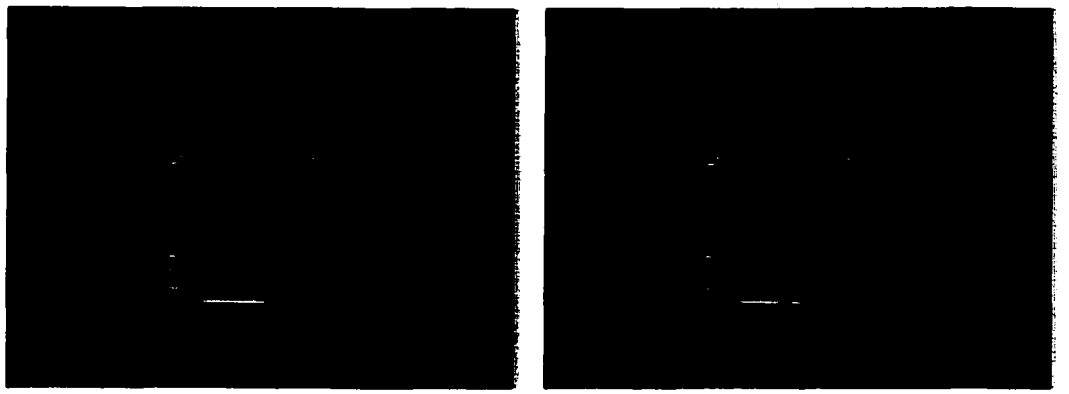
Collection 4

Figure 6.9: Data Sets for Model Recognition, Part 2



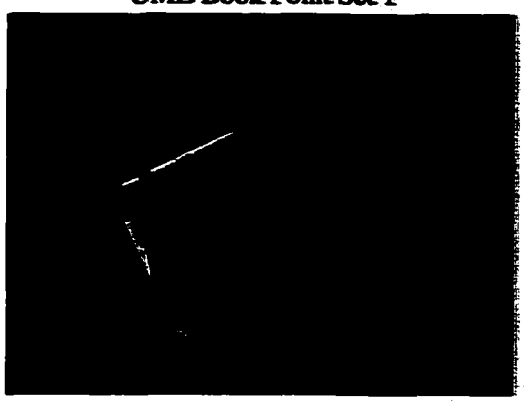
Collection 5

Figure 6.10: Data Sets for Model Recognition, Part 3



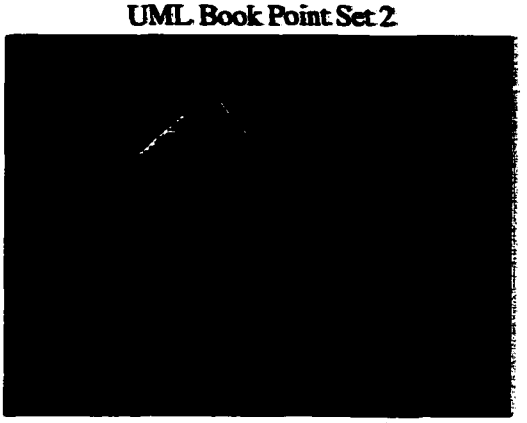
UML Book Point Set 1

UML Book Point Set 1 with Clutter

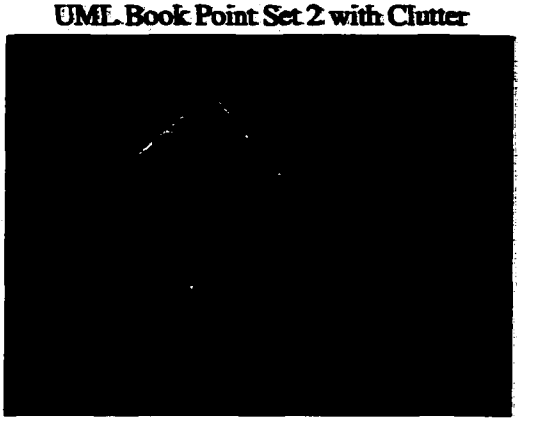


UML Book Point Set 2

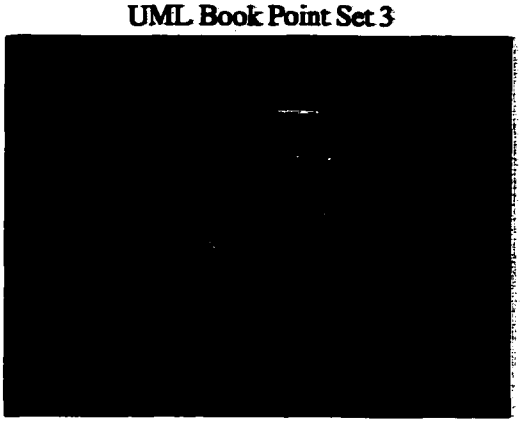
UML Book Point Set 2 with Clutter



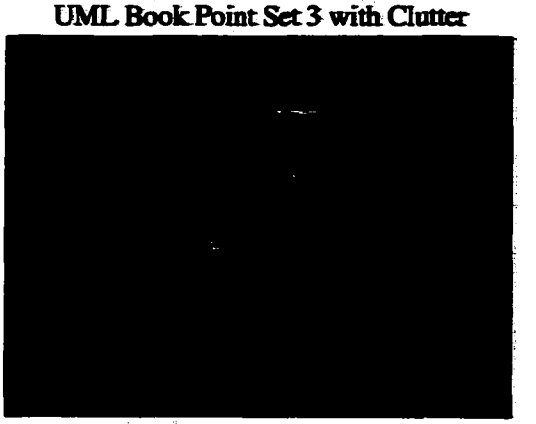
UML Book Point Set 3



UML Book Point Set 3 with Clutter



UML Book Point Set 4



UML Book Point Set 4 with Clutter

Figure 6.11: Data for the Rigid Body Problem

Chapter 7

Experiments with Local Search

7.1 Experiment Methodology

A critical question is how often, and after how much work, does local search find a globally optimal solution, or at least an apparently optimal solution? The answer to this question is that local search can often find a solution with relatively little work. This prompts the next question, when does local search fail, and why? Then, how many trials does it take to find a solution, and how long does it take to run the required number of trials? How does this compare with other algorithms? Presented here are experiments which answer these questions and others.

All experiments where run on 1 GHZ Pentium III Xeon machines, running the Linux operating system. The experimental software was compiled with the Intel C/C++ compiler. The standard compiler for this platform is the GNU C compiler GCC, but this compiler is known to be poor at optimizing floating point operations. The Intel compiler provides approximately a 30% boost in run times for this code on these machines. Because of the long run times required to run large numbers of trials, most experiments where run in a distributed fashion. When random starts are used, running local search in a distributed fashion simply required multiple independent program instances be run. When the key feature algorithm was used to generate initial starts, each program instance generated the entire key feature list and then searched only a limited portion of it based on its given search parameters. In both cases the results of all runs where collected and merged into a single log file, with the appropriate statistics re-computed to gain a complete picture of the experiment.

7.2 Image Registration Experiments

Experiments related to image registration where conducted on all six image registration imagery sets. The effects of local search where evaluated by pairing each data set with every data set taken from the same imagery, including itself. This resulted in 106 image registration problems, including 24 identity problems when a data set was matched to itself.

Tables 7.1 through 7.5 summarize these problems. Each individual experiment is identified by the imagery involved, and the model and data sets which make up the experiment. The size of the model and data sets are denoted m and d, with the total number of possible pairs givens as $n = md$. The "Pairs" entry is the number of these pairs in the optimal match. For all identity problems and problems in the Polygon and Picture images sets this number is known exactly. For most problems in the Java Book image sets, this number can be determined by inspecting the best match found, and checking that all possible pairings have been found correctly. For a few problems in this set, local search failed to find a reasonable match and the number of optimal pairs is not known. The same holds true for the Ft. Hood and Poster problems sets; the number of optimal pairs given is based on inspection of the best match found. In cases where inspection clearly shows that the resulting match is not optimal no number is given; the number of optimal pairs is listed as NS, for Not Solved. Local search failed to find solutions for all of the problems derived from the imagery provided by Jacobs, except for the identity problems. The final column gives the size of the optimal match relative to the total number of possible pairings. That is, the ratio of the number of pairs in the optimal match to n.

It should be noted that the number of pairs in an optimal solution is not always symmetric when the roles of model and data are reversed. The evaluation function is not symmetrical, and this can sometimes result in pairings being dropped or added. The matter is further complicated by the presence of noise in point locations, which can effect the optimal pose and thus the ability of the search to add or drop pairs. However, the set of optimal pairings between model and data are always very similar, and often the same for many problem instances.

Although local search searches through correspondence space, there is a pose associated with each search state and the optimal pose is often of as much interest as the optimal match set. Transforming the original model imagery by the pose associated with the global optima provides a quick

Table 7.1: Summary of Problems from the Polygon Imagery

Model	Data	m	d	$n = md$	Pairs	Rel. Size
1	1	9	9	81	9	0.111
1	2	9	9	81	8	0.988
1	3	9	12	118	9	0.076
1	4	9	10	90	9	0.100
1	5	9	10	90	9	0.100
2	1	9	9	81	8	0.988
2	2	9	9	81	9	0.111
2	3	9	12	118	9	0.076
2	4	9	10	90	9	0.100
2	5	9	10	90	9	0.100
3	1	12	9	118	9	0.076
3	2	12	9	118	9	0.076
3	3	12	12	144	12	0.083
3	4	12	10	120	10	0.083
3	5	12	10	120	10	0.083
4	1	10	9	90	9	0.100
4	2	10	9	90	9	0.100
4	3	10	12	120	10	0.083
4	4	10	10	100	10	0.100
4	5	10	10	100	10	0.100
5	1	10	9	90	9	0.100
5	2	10	9	90	9	0.100
5	3	10	12	120	10	0.083
5	4	10	10	100	10	0.100
5	5	10	10	100	10	0.100

Table 7.2: Summary of Problems from the Picture Imagery

Model	Data	m	d	$n = md$	Pairs	Rel. Size
1	1	16	16	256	16	0.0625
1	2	16	15	240	15	0.0625
1	3	16	15	240	14	0.0583
1	4	16	14	224	14	0.0625
2	1	15	16	240	15	0.0625
2	2	15	15	225	15	0.0667
2	3	15	15	225	14	0.0622
2	4	15	14	210	14	0.0667
3	1	15	16	240	14	0.0583
3	2	15	15	225	14	0.0622
3	3	15	15	225	15	0.0667
3	4	15	14	210	14	0.0667
4	1	14	16	224	14	0.0625
4	2	14	15	210	14	0.0667
4	3	14	15	210	14	0.0667
4	4	14	14	196	14	0.0714

visual check of the accuracy of the match. Figure 7.1 shows typical results for the image registration experiments, showing the original model image, the transformed model image, and the data image it was matched to.

7.2.1 How many random starts does local search require?

Random starts local search was run on three of the six image registration imagery sets; the polygon data, the picture data, and the Java book data. Although it could be easily run on other data sets, the results from these three experiments suggest that it would require substantial amounts of CPU time to achieve good solutions to any of these problems using this algorithm. The results of these experiments are summarized in tables 7.6, 7.7, and 7.8. The model and data columns give the points sets used for each experiment. The trials column gives the number of trials run, and the column labeled "Sec/Trial" gives the average time required to run a single trial on this combination of model and data sets. The column labeled "Opts" gives the number of times that local search succeeded in finding the global optima. The percentage failure column gives the ratio of success to failures, giving the expected chance that any given random trial will fail; rounded to three decimal places. Note that in some cases the percentage chance of failure is very close to one, and is given as 1.000 in the table.

Table 7.3: Summary of Problems from the Java Book Imagery

Model	Data	m	d	$n = md$	Pairs	Rel. Size
1	1	51	51	2601	51	0.0196
1	2	51	50	2550	36	0.0141
1	3	51	53	2703	NS	
1	4	51	47	2397	NS	
1	5	51	39	1989	NS	
1	6	51	45	2295	NS	
2	1	50	51	2550	NS	
2	2	50	50	2500	50	0.0200
2	3	50	53	2650	NS	
2	4	50	47	2350	NS	
2	5	50	39	1950	31	0.0159
2	6	50	45	2250	NS	
3	1	53	51	2703	NS	
3	2	53	50	2650	NS	
3	3	53	53	2809	53	0.0189
3	4	53	47	2491	39	0.0157
3	5	53	39	2067	37	0.0179
3	6	53	45	2385	35	0.0147
4	1	47	51	2397	NS	
4	2	47	50	2350	31	0.0132
4	3	47	53	2491	39	0.0157
4	4	47	47	2209	47	0.0213
4	5	47	39	1833	33	0.0180
4	6	47	45	2115	33	0.0156
5	1	39	51	1989	NS	
5	2	39	50	1950	28	0.0144
5	3	39	53	2067	35	0.0169
5	4	39	47	1833	30	0.0164
5	5	39	39	1521	39	0.0256
5	6	39	45	1755	28	0.0160
6	1	45	51	2295	NS	
6	2	45	50	2250	NS	
6	3	45	53	2385	36	0.0151
6	4	45	47	2115	33	0.0156
6	5	45	39	1755	33	0.0188
6	6	45	45	2025	45	0.0222

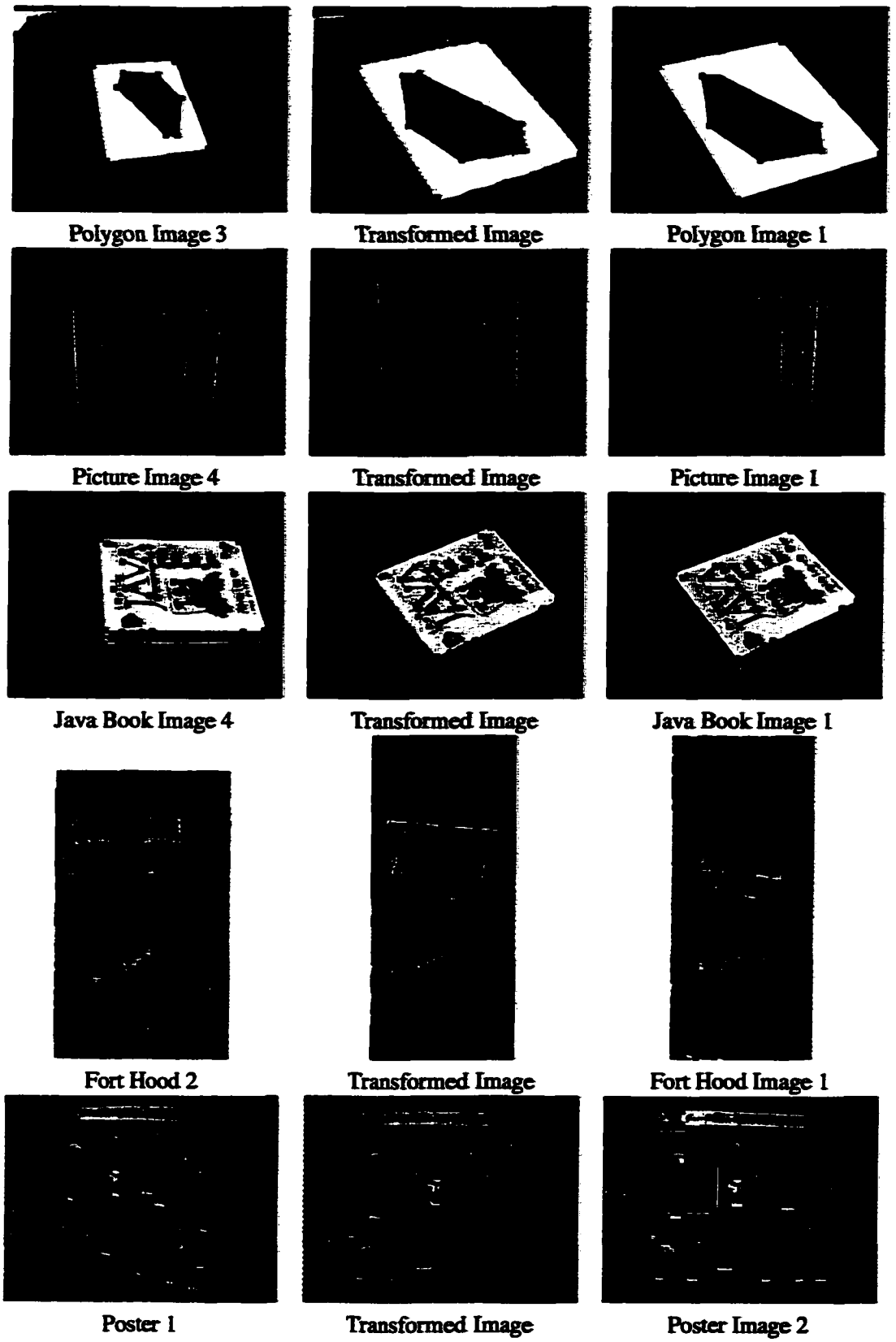


Figure 7.1: Typical Results for Image Registration

Table 7.4: Summary of Problems from Fort Hood Imagery

Model	Data	m	d	$n = md$	Pairs	Rel. Size
1	1	65	65	4225	65	0.0154
1	2	65	67	4355	28	0.0064
2	1	67	65	4355	27	0.0062
2	2	67	67	4489	67	0.1492

Table 7.5: Summary of Problems from Poster Imagery

Model	Data	m	d	$n = md$	Pairs	Rel. Size
1	1	119	119	14161	119	0.0084
1	2	119	128	15232	106	0.0070
1	3	119	97	11543	NS	
2	1	128	119	15232	107	0.0070
2	2	128	128	16384	128	0.0078
2	3	128	97	12416	NS	
3	1	97	119	11543	NS	
3	2	97	128	12416	NS	
3	3	97	97	9409	97	0.0103

The value of t^* is the number of trials that must be run before we can be 99% confident that the global optima has been found. Based on this number the average required run time can be computed, and it is given in the last column.

Examination of tables 7.6 and 7.7 shows that problems from the polygon and picture image sets are generally easy for local search to solve, with a few exceptions. Matching point set one from the polygon imagery to point sets two and three appears to be marginally more difficult than other problems from this imagery, and matching set three to set one is extremely difficult for random starts local search. These are not anomalous results, running the experiment again produced similar numbers. Figure 7.2 shows the number of trials which must be run to achieve 99% confidence that the global optima has been found, as a function of n, the total number of possible pairings. With the exception of a few outliers this appears to be a linear relationship, with the number of trials required growing linearly with the product of the size of point sets being matched. However, the high variability in run time for different problems of similar size make it difficult to draw firm conclusions about the nature of the run-time growth

Table 7.6: Random Starts Local Search for the Polygon Problem Set

Model	Data	Trials	Sec/Trial	Opts	% Fail.	$t*$	Time
1	1	10000	0.000942	277	0.972	164	0.15
1	2	10000	0.000928	21	0.998	2191	2.03
1	3	10000	0.001555	23	0.998	2000	3.11
1	4	10000	0.001160	179	0.982	255	0.30
1	5	10000	0.001148	153	0.985	299	0.34
2	1	10000	0.000937	110	0.989	416	0.39
2	2	10000	0.000944	261	0.974	174	0.16
2	3	10000	0.001544	105	0.989	436	0.67
2	4	10000	0.001149	198	0.980	230	0.26
2	5	10000	0.001147	202	0.980	226	0.26
3	1	10000	0.001273	4	1.000	11511	14.65
3	2	10000	0.001273	68	0.993	675	0.86
3	3	10000	0.002164	156	0.984	293	0.63
3	4	10000	0.001584	74	0.993	620	0.98
3	5	10000	0.001545	64	0.994	717	1.11
4	1	10000	0.001081	191	0.981	239	0.26
4	2	10000	0.001059	125	0.987	366	0.39
4	3	10000	0.001758	80	0.992	573	1.01
4	4	10000	0.001315	271	0.973	168	0.22
4	5	10000	0.001311	215	0.978	212	0.28
5	1	10000	0.001096	182	0.982	251	0.27
5	2	10000	0.001098	169	0.983	270	0.30
5	3	10000	0.001795	109	0.989	420	0.75
5	4	10000	0.001351	222	0.978	205	0.28
5	5	10000	0.001345	255	0.974	178	0.24

Table 7.7: Random Starts Local Search for the Picture Problem Set

Model	Data	Trials	Sec/Trial	Opts	% Fail.	t*	Time
1	1	10000	0.004738	54	0.995	851	4.03
1	2	10000	0.004273	41	0.996	1121	4.79
1	3	10000	0.004256	26	0.997	1769	7.53
1	4	10000	0.003816	29	0.997	1586	6.05
2	1	10000	0.004503	53	0.995	867	3.90
2	2	10000	0.004089	67	0.993	685	2.80
2	3	10000	0.004033	34	0.997	1352	5.45
2	4	10000	0.003657	55	0.994	835	3.05
3	1	10000	0.004232	26	0.997	1769	7.49
3	2	10000	0.003768	37	0.996	1242	4.68
3	3	10000	0.003753	69	0.993	665	2.50
3	4	10000	0.003408	64	0.994	717	2.44
4	1	10000	0.004177	43	0.996	1069	4.46
4	2	10000	0.003799	66	0.993	695	2.64
4	3	10000	0.003784	49	0.995	938	3.55
4	4	10000	0.003433	81	0.992	566	1.94

Table 7.8: Random Starts Local Search for the Java Book Problem Set

Model	Data	Trials	Sec/Trial	Opts	% Fail.	t*	Time
1	1	100000	0.137273	8	1.000	57562	7902
1	2	100000	0.089448	1	1.000	460515	41193
2	2	100000	0.115563	7	1.000	65786	7602
3	3	100000	0.143233	7	1.000	65786	9423
3	4	100000	0.123809	1	1.000	460515	57016
3	5	100000	0.098579	1	1.000	460515	45397
4	2	100000	0.102209	1	1.000	460515	47069
4	3	100000	0.110810	2	1.000	230256	25515
4	4	100000	0.096504	11	1.000	41863	4040
4	5	100000	0.077937	1	1.000	460515	35891
5	2	100000	0.082603	4	1.000	115127	9510
5	3	100000	0.088862	2	1.000	230256	20461
5	4	100000	0.077535	1	1.000	460515	35706
5	5	100000	0.064464	15	1.000	30699	1979
6	3	100000	0.107566	1	1.000	460515	49536
6	4	100000	0.093638	1	1.000	460515	43122
6	5	100000	0.074645	2	1.000	230256	17187
6	6	100000	0.088789	9	1.000	51166	4543

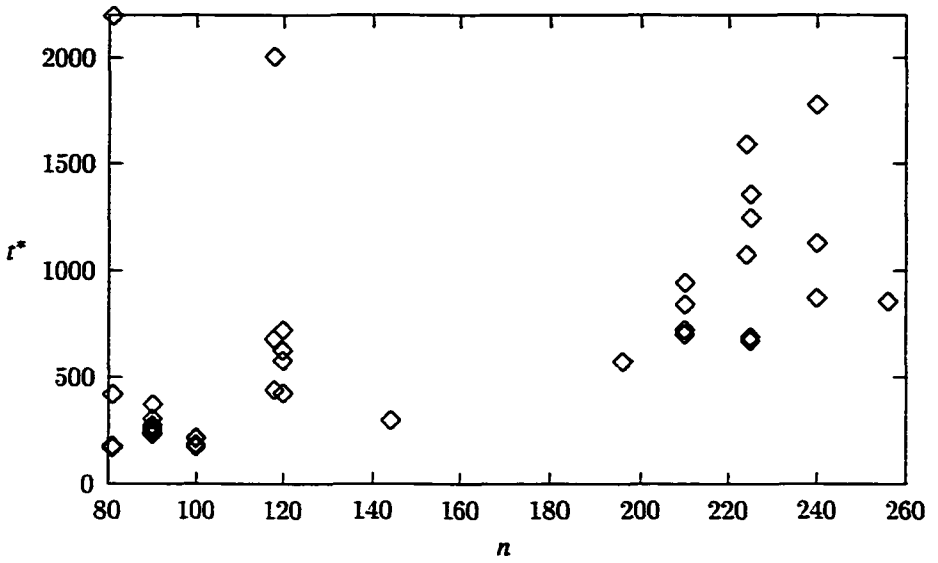


Figure 7.2: N vs. Trials Need for 99% of Success

Examination of table 7.8 suggests that 100,000 trials may be insufficient to solve problems from the Java book imagery on a consistent basis. It should be noted that point sets extracted from this imagery are not directly comparable to sets extracted from the polygon or picture imagery because they contain substantially more clutter and noise in the location of each point, as discussed previously in chapter 6.

7.2.2 How Effective is Heuristic Starts Local Search?

Tables 7.9 through 7.13 present the results of running heuristic starts local search on the image registration experiments. The first, second, and last entries in the table give the indices of the key features that allowed local search to find the global optima. The successes entry is the total number of key features which allowed local search to find the global optima, out of the total number of non-degenerate key features for a given problem. The density entry is the ratio of successful key features to the total number of key features, and can be seen as the probability that any entry in the key feature list will result in a successful trial. Comparing this number to the failure rates for random starts local search shows that heuristic starts is far more likely to result in a successful trial.

However, the density listing is misleading. For many of the problems the top ranked key feature results in a successful trial, and in many of these cases the second key features does as well. Looking

at the last entry shows that even low ranked key features can result in successful trials. The distribution of successful key features is weighted in favor of the early part of the list, but it appears that when the key feature algorithm is working any non-degenerate key feature is more likely to result in a successful trial than a random starting point.

Table 7.13 is somewhat abbreviated, these problems are large enough that running local search trials from all non-degenerate key features requires a prohibitive amount of time. Key feature local search successfully solved some problems in these problem sets, searching a sub set of the entire list and finding only one successful key feature.

Table 7.9: Heuristic Starts Local Search for the Polygon Problem Set

Model	Data	1st	2nd	Last	Successes	Total KF	Density
1	1	1	2	1921	65	1943	0.0335
1	2	1	2	1858	8	1944	0.0041
1	3	3	4	2086	14	2592	0.0054
1	4	1	118	2153	43	2160	0.0199
1	5	1	2	2135	43	2158	0.0199
2	1	1	2	1907	51	1944	0.0262
2	2	1	2	1891	62	1944	0.0319
2	3	1	2	2589	63	2592	0.0243
2	4	1	2	2141	33	2160	0.0153
2	5	1	2	2113	45	2160	0.0208
3	1	2	3	2073	5	2591	0.0019
3	2	1	2	2433	26	2592	0.0100
3	3	1	2	3363	59	3456	0.0171
3	4	1	113	2751	32	2879	0.0111
3	5	1	2	2647	26	2879	0.0090
4	1	1	74	2077	55	2160	0.0255
4	2	1	2	2144	32	2160	0.0148
1	2	1	491	2654	26	2880	0.0090
4	4	1	2	2344	64	2400	0.0267
4	5	1	2	2319	43	2400	0.0179
5	1	1	2	2128	27	2160	0.0125
5	2	1	2	2113	33	2160	0.0153
5	3	1	2	2764	37	2879	0.0129
5	4	1	2	2399	56	2400	0.0233
5	5	1	2	2395	56	2400	0.0233

It is worth noting that random starts local search was able to solve some problems in the Java book imagery that heuristic starts local search was not able too, even though it did not solve all problems that heuristic starts did. This is to be expected. For problems where no good key feature exists

Table 7.10: Heuristic Starts Local Search for the Picture Problem Set

Model	Data	1st	2nd	Last	Found	Total KF	Density
1	1	1	2	5709	45	6144	0.0073
1	2	1	2	5660	23	5760	0.0040
1	3	5	6	5373	18	5760	0.0031
1	4	1	6	5024	16	5376	0.0030
2	1	1	2	5162	27	5758	0.0047
2	2	1	2	5234	57	5400	0.0106
2	3	1	2	5260	18	5400	0.0033
2	4	1	2	4982	28	5040	0.0056
3	1	1	2	5330	29	5757	0.0050
3	2	1	2	5151	44	5399	0.0081
3	3	1	2	5372	89	5398	0.0165
3	4	1	2	4970	42	5039	0.0083
4	1	1	4	4659	28	5376	0.0052
4	2	1	2	4977	39	5039	0.0077
4	3	1	2	4666	34	5040	0.0067
4	4	1	2	4497	57	4704	0.0121

Table 7.11: Heuristic Starts Local Search for the Java Book Problem Set

Model	Data	1st	2nd	Last	Found	Total KF	Density
1	1	1	2	46728	61	62085	0.0010
1	2	14807	None	14807	1	60922	0.0000
2	2	1	2	56485	72	59777	0.0012
3	3	1	2	66674	77	67311	0.0011
3	4	2	106	18894	3	59710	0.0001
3	5	3298	14654	44333	3	49532	0.0001
3	6	1	4	56142	7	57142	0.0001
4	3	1	164	164	2	59496	0.0000
4	4	1	2	52576	73	52772	0.0014
4	5	10119	None	10119	1	43741	0.0000
4	6	11	44955	44955	2	50521	0.0000
5	3	57	305	29967	4	49586	0.0001
5	4	24513	None	24513	1	43974	0.0000
5	5	1	2	27432	48	36489	0.0013
5	6	1	None	1	1	42104	0.0000
6	3	1	2	56351	7	57170	0.0001
6	4	1	17474	22736	3	50702	0.0001
6	5	16	5098	5098	2	42058	0.0000
6	6	1	2	46974	67	48517	0.0014

Table 7.12: Heuristic Starts Local Search for the Fort Hood Problem Set

Model	Data	1st	2nd	Last	Found	Total KF	Density
1	1	1	2	93201	116	94912	0.0012
1	2	4511	13027	13027	2	97399	0.0000
2	1	720	0	720	1	100459	0.0000
2	2	1	2	95751	95	103260	0.0009

Table 7.13: Heuristic Starts Local Search for the Poster Problem Set

Model	Data	1st	Total KF
1	1	1	323629
1	2	88	349241
2	1	2222	351366
2	2	1	379511
3	3	1	219376

because of clutter or noise in the data, random starts may still be able to find a successful starting point by chance. In theory random starts local search is capable of solving any problem that heuristic starts local search can by correctly picking an initial solution which corresponds to a successful key feature, but this is unlikely to happen unless a huge number of trials are run or there are lots of successful key features.

7.2.3 How should local search be employed?

When heuristic starts local search works, it generally does so very quickly. For small problems with little noise and clutter the top ranked key feature will almost always result in a successful trial. Even more difficult problems will often be solved by starting local search from the first key feature, and for those problems where this is not the case searching from only a small portion of the key feature list frequently results in the global optima being found. However, several problems from the book imagery are only solved by only one key feature that occurs very late in the list. In these cases, it seems probably that key features resulting in a successful trial do so by chance rather than some desirable property of the key feature algorithm. A small number of other problems were only solved by random starts local search, none of the key features generated lead local search to the global optima.

It seems that if the first part of the key feature list does not solve the problem, it is better to employ random starts local search to solve a particular problem. This raises the question of how to determine

if a given solution is optimal or not; an important question given that large problems require a large number of random trials before there can be any confidence that the global optima has indeed been found. The nature of the problem makes this a difficult question to answer, at least in the general case and without human intervention. Some progress may be made if the something is known about the point sets to be matched. The final value of of the objective function, equation 4.1, is often dominated the omission term. If the number of missing points can be estimated, then a bound may be placed on the error term and search terminated when the error term meets this criterion.

Generally, $2 \times max(m,d)$ key features seems sufficient to solve point matching problems with little noise and low clutter, if heuristic starts local search can find the global optima. If the solution found is not satisfactory, then random start local search should be employed. For problems with more noise and clutter it may be appropriate to try more key features, perhaps as many as $1/2(md)$, before resorting to random starts local search.

7.2.4 Solving Identity Problems

Local search solved all identity problems, matching a point set to itself, even for the data extracted from the Jacob's imagery where it could not solve any other problems. These problems are substantially simpler than ordinary point matching problems for a number of reasons. First, identity problems effectively contain no noise. When any subset of the correct pairings is used to determine an optimal pose the result will be the identity transformation, which will cause every model point to exactly match the appropriate data point. The absence of missing or spurious points also makes the problem significantly easier because it greatly reduces the chance of local search becoming confused by erroneous pairings. This allows random starts local search to locate a correct solution in relatively few trials.

The use of the key feature algorithm further simplifies the solving of identity problems. Because there are no missing or spurious points, the initial key feature list is guaranteed to contain md key features which are proper subsets of the optimal match. Because each of these key features is the same size all key features incur the same omission penalty, and the formulation of the degeneracy function ensures that $S(P^*) = 0$ when P^* is the identity transformation. The fitting error for those features, which represent subsets of the true match, will also be zero due to the lack of noise. Conse-

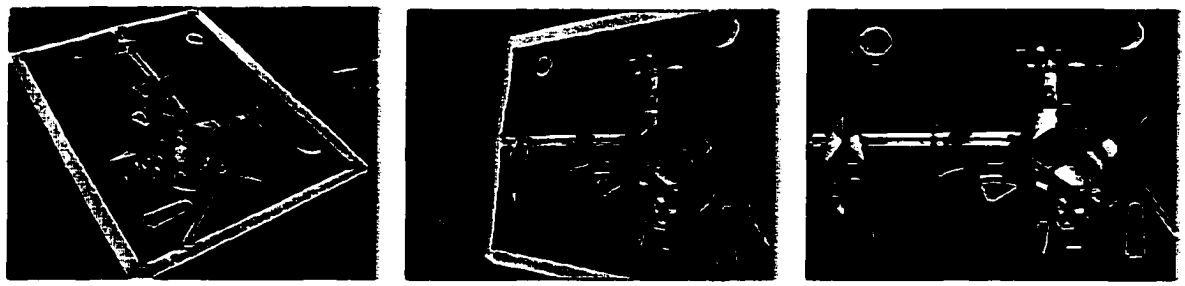
quently, a ranked list of key features will always place one key feature representing a proper subset at the top of the list. This key feature will always be in the basin of attraction for the global optima, since each omitted point, and only the points omitted from the optimal match, may be added by local search without incurring an increase in the fitting portion of the evaluation function.

7.2.5 Failure Modes for Local Search

Local search fails to find optimal or even good solutions for some of the image registration problems from the Java book and poster imagery, and completely fails to solve problems derived from the imagery provided by Jacobs.

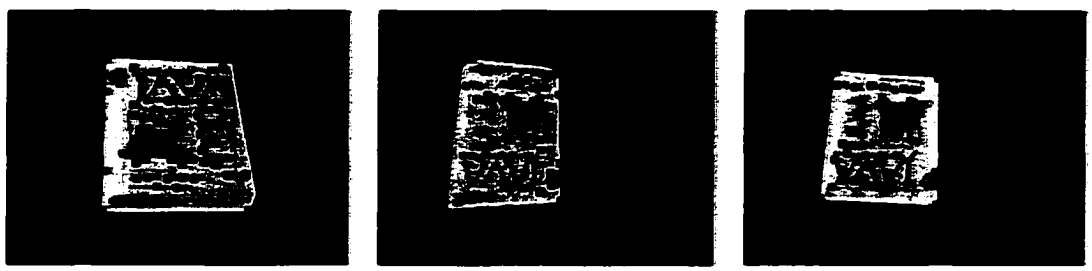
In some cases local search finds a set of correct correspondences which are spatially proximate, but fails to find correct matches for points in other parts of the model set. Figure 7.3 shows an example of this from the Jacob's imagery; local search has correctly matched points along what is the the right side of data set, but failed to find correct matches for points on the left hand side. With five of more pairs in the correspondence the optimal pose is the least squares estimate which minimizes the distance between model and data points. The presence of noise in the location of each point causes this estimate to be inaccurate, and because the projective transformation has eight degrees of freedom the estimate found can match the points in the correspondence very closely while placing unmatched model points far from their true location. When the correctly paired model points occur in the same spatial region this can result in a model pose which is locally optimal for some region but becomes progressively worse for points farther from those in the correspondence. When a sufficient number of correct pairs are added from the same region it can be difficult for local search to find a correct pairing in another region because doing so will increase the fitting error on each of the previously matched points, while the error on the newly located pairing will remain high until other other pairs in the region are added to balance the pose estimate. Consequently local search needs to find correspondences from several areas of the model set early in the search in order to correctly locate the globally optimal solution.

Local search can also be lead astray by a special combination of missing and spurious points, here termed "misleading points". A misleading point is a spurious point in the data which occurs close to a missing point. The result is that there exists a point in the model for which there is no correct



Model Image Transformed Model Image Data Image

Figure 7.3: Local Search finding a subset of the correct correspondences



Model Image Found Optima Data Image

Figure 7.4: Example of Misleading Points

pairing, but for which there is a data point which, when the model is properly transformed, lies close to that model point. For instance, in the point set derived from the first image in the Java book set the bottom two corners of the front cover are not detected by the corner extractor. However, the corner extractor does find the bottom two corners of the back cover in this image. Point sets extracted from other images in this data set omitted the points on the bottom cover but do contain the corners of the front cover. Because the points on the bottom cover are spatial proximity to the location of the missing front corner points, local search will often try to pair them to the front cover points. Figure 7.4 illustrates this. The problem of missing points becomes more acute when the point sets to be matched are extracted from imagery where there are clusters of closely spaced features.

Under certain circumstances it may be reasonable to treat misleading points as cases where the model and data points match with a high degree of fitting error. If there are a large number of correct correspondences already found then this is not a significant error and may not visually effect the resulting optimal pose. However, misleading points are distinct from the case where there is a great deal of noise in the point location because they arise from different features in the original imagery.

Card 1 Composite Collection 1

Figure 7.5: Results of Local Search for Model Recognition

A close examination of problems drawn from the Java book and poster data shows that even when the initial starting point for local search is a proper subset of the global optima, the global optima may not be found. In fact, most key features which where proper subsets of the global optima did not result in a successful trial, although a few did. This behavior can be explained by the presence of misleading points and the high level of noise and clutter present in both model and data sets. Problems derived from the Ft. Hood, picture, and polygon imagery do not exhibit this behavior, and for these problems key features which are subsets of the global optima always result in the optima being found.

7.3 Model Recognition Experiments

One of the primary problem domains for point matching is model recognition, the task of finding one or more model instances within a data set. Experiments using local search to solve this kind of problem where done using the data drawn from the Christmas card imagery. Results in this problem domain where generally not as encouraging as those for image registration, but this appears to be partially due to issues arising from the quality of the data sets available, rather than from limitations of local search.

Key feature local search correctly found all three instances of the first Christmas card in the first image of the first collection. Figure 7.5 shows the first card transformed by the three correspondences found in the first data set, and composited into a single image. In the other two data sets drawn from this imagery local search was only able to find two instances of the first card. Images from this collection also contain a single instance of the second card in the set, which local search could not find.

The second collection of images contained two instances of the second card, and local search correctly located these in the first data set, and correctly located the first card in the remaining two. In the second data set from this imagery, local search nearly located the second instance correctly,

but a missing corner point results in one corner of the second instance being off.

Key feature local search consistently located the first card in all versions of the third, fourth, and fifth collections. Local search proved unable to locate instances of the second of third problem due to problems with the data set.

7.3.1 Difficulties with Model Recognition

The model recognition experiments raised several issues with point matching in general that did not arise in earlier experiments. First, local search tended to find multiple correspondences for the same physical model instance. These correspondences contain a majority of pairs in common, but differ by only a few pairings. Pairings between two sets of spatially proximit points maybe swapped, or one pairing would be missing allowing another model point to map to the unmatched data point. All but one of these correspondences is suboptimal, with respect to the objective function, for any given instance of the model in the data, but each represents a plausible mapping from model to data. The pose associated with these nearly optimal solutions is generally very close to the optimal pose. When the original image is transformed by the optimal pose, and by a nearly optimal pose, a difference can be seen, although the difference is only obvious when the images are superimposed or studied side by side. These nearly optimal solutions all have match errors which are extremely close in value. Consequently, the top four or five matches often represent the same model instance, followed by a group of nearly optimal matches for the next model instance. In theory this situation could occur with any data set, although it was rarely seen in the image registration experiments. Here it is important only because the goal is to find multiple instances, not the single best model instance. In order to discriminate between correspondences for the same model instance and those for different model instances, a simple heuristic of counting the number of shared and unshared pairs was used. Correspondences with more than half of their pairs differing where considered to represent different model instances.

Local search was unable to match the second card in the first collection, and generally unable to match most cards other than the first in most of the collections. Examination of the data sets shows that it is unlikely that any algorithm could solve these problems. The model images for all three cards contain a large number of points spread across the area of the card. The images for these models

where taken at roughly twice the scale as the images used to generate the collections in which the cards where matched. The greater distance between the camera and the objects of interest resulted in images from which the available corner extractors where unable to extract as many points. The large number of points missing in the data set makes it impossible to find an accurate match between model and data. It is possible that a more accurate match could be found by deleting points from the model. However, when there are only a small number of points to match, the projective transformation has sufficient degrees of freedom that any number of plausible mappings and poses can be found. Many of these have little or no relation to the desired match. It is even possible that one of these mappings will have a lower fitting error than the true match, once the effects of noise in the point locations is taken into account.

The ability of local search to solve problems from the polygon data set, and to find all three instances of the first card in collection one, demonstrate that the algorithm can deal with large changes in scale between point sets. However, such large changes bring with them the secondary problem of points going undetected. In many real world problems this may be a significant factor. Under such circumstances it seems unlikely that any point matching algorithm could find an appropriate match with any confidence.

7.3.2 How Long Does it Take to Run Local Search?

The effectiveness of random starts local search depends on the ability to run lots of trials in a relatively short period of time. Even when enhanced with the key feature algorithm, local search benefits from the ability to quickly conduct a single trial. Figure 7.6 shows a plot of n versus the time required for a single trial of local search. This data comes from the average time required to run each trial in all experiments previously discussed in this chapter. The value $n = md$ is the product of the sizes of the point sets, and an indicator of the size of the search space. As more points are added to either point set, the size of the search space grows exponentially. Time is measured in milliseconds. In order to place data from experiments with a wide range of values for n on the same plot, values on both axes are given as square roots.

Examining the data shows that the run time for local search is very nearly linear with n, even given the various amounts of clutter in the different problem instances. There does appear to a slight

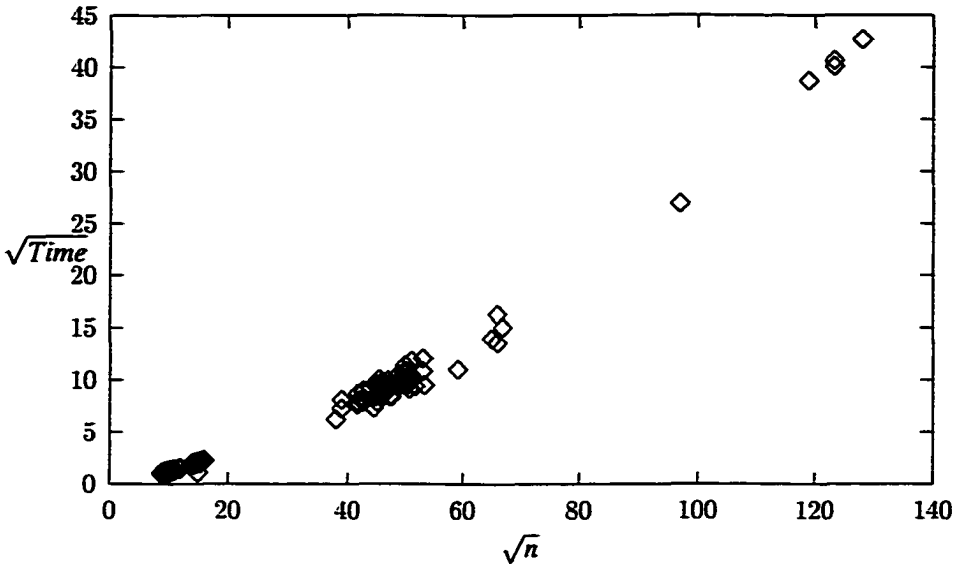


Figure 7.6: N vs. Run Time of Local Search

upward curve in the data, but clearly the growth in run time is modest compared to growth in n. This is an encouraging result, suggesting that local search scales well with the problem size and can be expected to maintain its efficiency when applied to problems larger than those examined here. It appears that the trials using key features for model recognition ran slightly faster than those for image registration problems, which ran slightly faster than random starts local search on the same problems. Although these trials are not differentiated here, the difference is very slight and does not seem to be significant.

7.4 Similarity Experiments

Although any algorithm capable of solving projective point matching problems is, in theory, also capable of solving problems from more limited transformation classes, problems requiring a more limited transformation class are often best solved by algorithms which consider only the actual space of candidate transformations. Considering a larger transformation space than necessary can make an algorithm more susceptible to being mislead by clutter and noise, and the additional degrees of freedom increase the number of matches which are plausible or somewhat plausible. A version of local search restricted to similarity transformations has the added benefit of simpler fitting and degeneracy calculations, reducing the time required for each trial.

This raises the possibility of using a match found under a low order search transformation to approximate a correct solution, when the low order transform is likely to approximate the correct, final, transformation. Such an approximation could be found quickly, and with less likelihood that the point matching algorithm will be mislead by incorrect poses which match a large number of points.

A few of the images taken for previous experiments where taken at similar camera angles, so that there is a similarity transform which will approximate the true projective transformation. A version of local search which was restricted to considering similarity transforms was used to search for matches between these point sets; as discussed in chapters 3 and 4 this version differs from the previously discussed algorithm only in the fitting and degeneracy routines used. Table 7.14 summarizes the results of these experiments. The correspondences found generally included a subset of the correct correspondences, with about an equal number of correspondences which where incorrect but are reasonably close under the found transformation. The resulting poses are not correct, but a good approximation. Figure 7.7 shows how well this technique may sometimes work.

Model Image Transformed Image Data Image

Figure 7.7: Similarity Approximation to Projective Transform

Table 7.14: Key Feature Local Search for Similarity Approximation

Model	Data	Correct	Missing	Incorrect	Secs/Trial
Java Book 3	Java Book 4	14	25	19	0.0763
Java Book 4	Java Book 3	16	23	10	0.0690
Xmas 1b	Xmas 1c	33	38	17	0.5017
Xmas 1c	Xmas 1b	38	13	9	0.3964
Xmas 3a	Xmas 3b	15	25	9	0.0908
Xmas 3b	Xmas 3a	15	25	4	0.1142

These experiments suggest that, in certain circumstances, it may be worth while to begin with an approximate solution under a low order transformation class, and then use that as the basis for finding

a proper solution under a higher order transform. Clearly this is not practical when the additional parameters of the higher order transformation dominate the correct pose, but when there effects are small such a technique may be effective. Running local search in a restricted transformation domain is somewhat faster. First, determination of the pose for similarity transforms is much quicker than determination of the optimal projective pose. Second, the reduced degrees of freedom make it easier to identify those pairings which are obviously incorrect, and this can be used to effectively prune the search space.

7.5 Comparisons with RAST

For point matching problems where the transformation class is limited to transformations such as similarity or general affine transforms, the RAST algorithm by Breuel provides an efficient polynomial time solution (7; 8). An implementation of RAST, written in Java, which solves point matching problems when the class of allowed transformations is translation and rotation, is provided at the end of (8). This implementation was compared with a version of local search restricted to similarity transforms, the same version used in section 7.4.

Previous experiments with local search suggest that the presence of clutter is the driving factor in the performance of local search. RAST is also effected by clutter, but perhaps less so. In particular, the key feature algorithm often allows local search to find the optimal solution in the first trial when there is little clutter. It is unlikely another algorithm can improve significantly on this result, but most problems of interest have significant clutter. To make a meaningful comparison, clutter was added to the rigid body data set in order to produce a second set of problems as discussed in chapter 6, which provide a better test of the relative abilities of these algorithms under real world circumstances.

Several things should be noted about the relative performance of these algorithms. The Java implementation of RAST provided by Breuel is meant to demonstrate how easily an implementation of RAST may be written, it is not optimized at the algorithm level. Second, the implementation discussed here was compiled and run using the Java SDK provided by Sun, version 1.3.1 for Linux. Performance of such an implementation is greatly affected by the choice of virtual machine, but even with a highly efficient virtual machine a program written in Java can not be expected to perform as well as a similar algorithm implemented in C, or otherwise compiled to run natively. Breuel reports

Table 7.15: Comparison between RAST and Random Starts Local Search

Model	Data	RAST Time	1/3 RAST Time	RSLS Time
1	1	0.4700	0.1567	16.6676
1	2	11.9300	3.9767	25.4250
1	3	19.6000	6.5333	16.8935
1	4	15.7100	5.2367	16.5450
2	1	29.4200	9.8067	21.0840
2	2	1.9700	0.6567	30.9814
2	3	NS	NS	21.6881
2	4	32.7200	10.9067	22.3353
3	1	24.7100	8.2367	17.5096
3	2	19.5100	6.5033	23.0911
3	3	0.4500	0.1500	14.9533
3	4	9.3900	3.1300	17.4925
4	1	22.9700	7.6567	17.7042
4	2	35.9200	11.9733	26.5666
4	3	27.0100	9.0033	17.4873
4	4	0.5800	0.1933	17.1184

that the C++ implementation of RAST runs about three times faster than the Java version (8), and this seems to be a reasonable claim. Consequently, timings between the version of RAST studied here and local search are not directly comparable.

Table 7.15 compares the results of running RAST on these problems with running random starts local search. The time required to run RAST is given for each problem, as is the an entry giving one-third this value for ease of comparison. The time given for random starts local search is the time required to reach 99% confidence that the global optima has been found, computed based on the number of times the optimal solution was found out of ten thousand trials, as explain previously. Table 7.16 gives the result of running key feature local search on these problems.

Random starts local search is roughly competitive with the Java based version of RAST on most problems, but clearly could not compete with a native code implementation. The use of the key feature algorithm changes this. In every case but one, the top ranked key feature leads local search to the optimal solution in the first trial. In the one case where this is not true, the eleventh key feature is successful. Running a hundred key features on these problems is more than sufficient for every problem and fast enough that it can be expected to out perform even an optimized version of RAST compiled

Table 7.16: Key Feature Local Search for Cluttered Rigid Body Problems

Model	Data	1st	2nd	Last	Successes	Total KF	Density
1	1	1	2	32000	154	32447	0.0047
1	2	1	2	38668	165	38688	0.0043
1	3	1	2	32223	147	32448	0.0045
1	4	1	2	32421	177	32448	0.0055
2	1	1	2	38385	97	38688	0.0025
2	2	1	2	46088	238	46128	0.0052
2	3	1	2	38630	160	38688	0.0041
2	4	1	2	38425	121	38688	0.0031
3	1	11	124	32113	109	32447	0.0034
3	2	1	8	38566	177	38688	0.0046
3	3	1	2	32424	237	32448	0.0073
3	4	1	2	32423	129	32448	0.0040
4	1	1	2	32409	114	32448	0.0035
4	2	1	3	38646	172	38688	0.0044
4	3	1	2	32409	133	32447	0.0041
4	4	1	2	32440	163	32448	0.0050

to native code. The ability of the key feature algorithm to solve these problems is not surprising. Under translation and rotation the relative distances of points from each other does not change, key features will not be disrupted except by the presence of clutter. Even then, a significant number of key feature clusters must be disrupted before the key feature algorithm fails to find one that lies in the basin of attraction for local search.

It is also interesting to note that RAST failed to solve one problem in the problem set, when allowed to run over night. When run on the uncluttered version of the data, RAST required nearly twenty minutes to solve the problem, more than twenty times what it required to solve the others. Why RAST has this difficulty is not clear, but clearly the presence of clutter makes it worse. For this problem RAST explored far more states than it did for other, similar problems. It seems likely that in certain cases RAST can become confused by similar or otherwise promising sets of correspondences, and must search a greatly increased set of states. If this is the case, then, RAST will encounter this sort of difficulty more often when the transformation class includes greater degrees of freedom and thus more ways to produce large sets of feasible correspondences.

7.6 When does RANSAC fail?

Prior to this work the best known algorithm for explicitly dealing with projective point matching was the RANSAC algorithm. This algorithm was originally formulated to deal with problems of image or model registration (15), and so its ability to solve the image registration problems was tested. It was found that for certain problems RANSAC can quickly find a globally optimal solution. However, real world conditions often render data unsuitable for use with this algorithm.

As shown by Hartley and Zisserman (19) small errors in the location of a point can quickly cause the estimated pose to diverge from the true pose. Even when the location of the points is relatively noise free, this problem can quickly become acute. Experiments with the picture data set show that for most of these problems the RANSAC implementation described in chapter 5 quickly finds a good key feature and from this is able to find about 70-75% of the correct pairings, spread across a large portion of the image plane. The resulting pose is visually close to the optimal. RANSAC fails to find the remaining pairs because the optimal pose for the pairings it has found places the unpaired model points five to ten times further away from the data points they should pair with, than the specified error tolerance σ would ordinarily allow. This is a problem with estimating a projective pose in general. With eight degrees of freedom it is possible to find a pose which matches a number of points quite well, while leaving other points far from their correct matches.

For the problems in the picture data set it is sufficient to simply increase σ to about ten times the ordinary value; under these conditions RANSAC easily and quickly solves all the picture problems. Unfortunately, the presence of clutter will prevent this approach from being effective. In problems from the picture data set there are very few missing or spurious points. If the pose is approximately correct and a transformed model point is unmatched then the closest data point is always the correct match. The presence of significant numbers of missing or spurious points will quickly destroy this assumption and lead RANSAC to incorrect pairings. These will lead the search further from the correct pose.

Even when a large number of trials are run RANSAC fails to find plausible solutions for all non-identity problems taken from the Java book or Fort Hood imagery. Given the amount of clutter and noise in these problems it seems likely that RANSAC is completely incapable of solving them, even

given unlimited run times.

When the available data is free of significant clutter and the location of the points is very precise, RANSAC may be a very efficient algorithm for point matching. Under real world conditions, the algorithm quickly fails and becomes unable to find good solutions.

7.7 What Makes Point Matching Hard?

The most significant factor in determining the difficulty of a point matching problem is the number of missing and spurious points present in the data sets. This is the primary reason that problems derived from the Java book imagery are significantly harder than the larger problems taken from the Fort Hood imagery. It is also the primary difference between the polygon and picture data, and other data sets used in this study. The polygon and picture data contain very little noise in the location of each individual point, and very few missing and spurious points. The result is data sets which are free of clutter for all practical purposes. By comparison, the other data sets contain significant amounts of clutter and noise. The problems drawn from the Jacob's imagery contain significant amount of spurious and missing points, and this prevents local search for solving these problems.

Consider the effects of clutter on random starts local search. In order for an initial random start to lead to the global optima it must, with very few exceptions, be composed of enough correct pairings to determine a good approximation to the optimal pose. The presence of clutter in point sets increases the chance that one of the initially chosen pairings will be incorrect, since if a missing or spurious point is chosen as part of one pairing there is no chance it will be paired correctly. Even if other pairings are correct, it is possible that this incorrect pairing will lead the search astray.

The key feature algorithm is also prone to disruption by the presence of clutter. A spurious or missing point that occurs close to any point which is matched in an optimal solution will prevent any key feature based on points close to it to contain fewer correct pairings. Allowing key features to contain one more point than is necessary allows local search to overcome this in some cases. A missing or spurious point occurring in such a cluster of points may be close enough to the true point that its disruption of the resulting pose may not be significant, but local search must still drop the bad pairing before the correct one can be found.

Even when local search can overcome missing and spurious points to locate a subset of the cor-

rect pairings, it must then find other correct pairings. Additional clutter and noise make this difficult as well. When there is even a little noise in the location of the initial subset points, the resulting pose can leave transformed model points in the vicinity of spurious points, or even within the error bounds on data point which should match to different model points. This can lead the search astray. Experiments with the Java book problems show that in many cases where the key feature algorithm correctly found a subset of the optimal pairings from which to start the search, it failed to find the global optima.

Chapter 8

Conclusion

8.1 Overview of Results

Point matching under the projective transform is a difficult problem, and prior work has rarely addressed this problem directly. Existing algorithms such as pose equivalence (11) and RAST (8) have difficulties scaling to address this class of transformations. The RANSAC algorithm (15) is capable of solving simple projective point matching problems, but has difficulties dealing with noise and clutter. Beveridge has previously formulated local search to address line matching under a similarity transform (5), and here it has been shown an effective algorithm for point matching. Combined with a key feature algorithm similar to that described by Beveridge and Whitley (32), this algorithm is capable of solving many point matching problems with little computational effort.

Both random starts local search and key feature based local search derive much of their effectiveness from the explicit evaluation of each pose in the search process. Previous point matching algorithms (3; 7; 11) have evaluated a pose only on the number model-data pairs which the given pose makes possible. Under rigid body transforms this is sufficient, as all poses are equally likely; algorithms for matching under the similarity transform need only guard against extreme scale changes. In the case of such low order transforms, any pose which matches a large number of model points to data points is likely to be correct. This is not the case for the projective transform. Many possible projective transforms are either physically impossible, or represent unlikely model positions under real world constraints. More importantly, for algorithms which evaluate a pose based only on the number of matching pairs, the projective transform has enough degrees of freedom that many poses can produce a large number of pairs while having no relation to the correct match. Prior work (3;

20) has not provided a means of evaluating a pose separate from the number of pairings which it satisfies. While the formulation of the degeneracy function presented here is not a definitive means of evaluating a projective pose, it does represent an important step towards search techniques which incorporate information about the likelihood of a given pose to guide the search.

Key feature local search has been shown to be effective at solving projective point matching problems from two separate computer vision domains; random starts local search can also solve these problems, provided sufficient trials are alloted. In the case of image registration, both algorithms find matches between large point sets, even in the presence of significant amounts of clutter. Local search algorithms can also be used to find multiple model instances within a data set, although issues of model resolution and feature extraction prevent practical problems for any algorithm operating within this domain. The practical details of solving these problems are widely different, and the algorithms presented here can reasonably be expected to perform well in novel problem domains.

The combination of local search and the key feature algorithm is competitive with existing algorithms (8) to solve point matching problems in the similarity or rigid body domains. Under the rigid body or similarity transforms the relative distance between points is fixed, and thus the key feature algorithm is less susceptible to disruption. In cases where there is not enough clutter to disrupt the key feature algorithm, local search can be expected to out perform tree search based algorithms. For problem instances where there is a great deal of deal of clutter, the key feature algorithm is less effective, but algorithms such as RAST can also be expected to require increased computational time to reach a solution in these situations.

8.1.1 Importance of the Key Feature Algorithm

For many problems, the overriding factor in determining the efficiency of local search is the performance of the key feature algorithm. Often, the key feature algorithm can chose partial solutions which are in the basin of attraction for the global optima as the top ranked key feature. Usually, the runner up will also be in the basin of attraction. For identity problems the top ranked solution is guaranteed to be in the basin of attraction for local search, and for simple problems it is nearly certain that this condition will continue to hold. Although the presence of missing and spurious points makes point matching more difficult, the key feature algorithm can often overcome this and still chose a

starting point for the first trial of local search which will lead to the global optima, as shown by the experiments using the imagery drawn from the Java Book. In those cases where it does not, there is often a solution leading to the optima on the list of non-degenerate key features, and it is often ranked highly enough to make running local search from each key feature on the first part of the list an effective search strategy.

The key feature list is of size $O(n)$, where n is the product of the sizes of each point set. This list can be ranked using a standard quick sort in time $O(n \log n)$. Problems from the Java book set provide a good example of typical run times. These point sets generally have forty or more points and a moderate amount of clutter. When the top ranked key feature leads local search to the global optima, the complete algorithm can be run in about 1 second on a 1 GHZ machine. When the class of transformations is limited to similarity transforms, problems of a similar size can be solved in about half that time; substantially faster than the RAST algorithm (8). Even when points sets are quite large, the combination of key feature creation and local search can run in under ten seconds provided the top ranked key feature leads to the global optima. Large problems with substantially clutter, such as those drawn from the poster imagery, require only about three minutes to run 100 trials of key feature based local search, which is often enough to find the correct solution.

8.1.2 Importance of Local Search

The key feature algorithm, when successful, is the main driving factor in the performance of local search. But, the local search algorithm itself is also important. As discussed in chapter 7, RANSAC proved unable to solve certain problems when started from the same initial solutions as local search. A greedy search which adds all promising pairs at each step can easily be lead astray by clutter, and confused by the presence of noise in point locations. Local search overcomes these problems to provide an effective means of building a complete solution from an initial solution which may or may not be a proper subset of the optimal solution.

The iterative version of RANSAC proposed by Hartley and Zisserman (19) can be seen as a form of local search; one which attempts to optimize an evaluation function based solely on the number of model points which fall within the error bounds of the data points. The search neighborhood is then all those pairs where a currently unmatched model point falls within the error bounds on

some data point, under the current transformation. The local search algorithms described here use a more sophisticated evaluation function, incorporating information about the current pose to guide the search. This allows local search to drop or avoid pairings which might be an attractive match but which would lead to unlikely transformations. This ability is particularly important for the projective case, where the greater degrees of freedom allow for poses which match a large number of points by stretching the model into unlikely configurations. Also important in the success of local search is that it considers only solutions which differ by a single pairing when looking for an improvement. RANSAC will add all those points which appear to be likely matches to a correspondence before is recomputes a pose, and never explicitly considers dropping a pairing. This results in longer trials for local search, but makes local search much less susceptible to clutter; it will not accidently pick up a bad pairing when adding a good pairing.

When the key feature algorithm fails local search may still be successful at finding a solution using random trials. Generally, for those cases where the key feature algorithm fails, a large number of random trials will be required. However, there is still some hope for success.

8.2 Open Questions

Local search can require large amounts of time to solve difficult point matching problems. Massively parallel implementations can partially address this issue, but a more efficient algorithm for projective point matching is still desirable. As previously discussed, the iterative version of RANSAC proposed by Hartley and Zisserman (19) can be considered a form of local search, where the local neighborhood is defined by the current pose rather than the correspondences currently in the match. This allows RANSAC to operate extremely fast, because a large number of pairs are added in between pose calculations. This suggests that a version of local search which adapts the neighborhood of the search based on the current pose may be substantially faster than the algorithm proposed here. A conservative approach would be to add all those pairs where a transformed model point is within some fraction of σ from a data point. This could be used to fill in most of the match, before the algorithm reverts to the single change neighborhood described here to find any remaining pairs.

When key features that lead local search to find the global optima appear late in the list of ranked features, there are often other successful key features at a similar position in the list. This occurs be-

cause key features pairing the same model and data points are likely to have a similar associated error. This suggests that the key feature list may be further pruned to remove similar key features. Alternatively, the pose of each key feature could be used in a pose clustering algorithm (30) to determine which areas of pose space, and which groups of key features, are likely to be lead to a globally optimal solution. It is also possible that an entirely different and more efficient heuristic algorithm may be found to determine initial starting points, but it is difficult at this point to suggest what form it might take.

Genetic algorithms hold promise for a more efficient solution to the point matching problem as well. Whitely and Beveridge (32) applied a messy genetic algorithm to the problem of line matching under a similarity transform. They found that the messy genetic algorithm solved difficult line matching problems in a fraction of the time required by the previously applied local search algorithm (5). As previously noted the local search algorithm presented here was developed based on this work, as is the key feature algorithm. This suggests that a similar messy genetic algorithm could be applied to solve the point matching problem.

The experiments presented in section 7.4 suggest that, where possible, it may be productive to begin the matching process by finding an approximate correspondence under a lower order transformation, such as a similarity transform, and then using that approximate solution to seed trials of local search which consider the full transformation space. Matching done under lower order transformations is faster, due to simplified pose and degeneracy calculations. The space of allowable transformations is also greatly constrained, and although the search process can consider only approximations to the correct pose, it does not spend time considering many invalid poses.

8.3 Conclusion

Local search is an effective strategy for point matching, particularly when paired with the key feature algorithm. For problems where the class of allowed transformations is of a low order, it is highly competitive with existing solutions. In the projective case it is capable of solving problems that the existing algorithm, RANSAC, can not. The presence of clutter in a model or data set greatly increases the difficulty of point matching for all algorithms, but local search is capable of dealing with this at the cost of increased computational time. For some large problems, clutter can make the run times of

local search excessive, but the nature of the algorithm allows for nearly linear improvements in run times on parallel processing systems. Successive refinements may potentially yield more efficient local search algorithms, increasing the ability of the general technique to solve difficult problems quickly. Several promising avenues for this have already been identified.

REFERENCES

[1] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete and Computational Geometry*, 3:237–256, 1988.

[2] E.M. Arkin, K. Kedem, J.S.B. Mitchell, J. Sprinzak, and M. Werman. Matching points into noise regions: combinatorial bounds and algorithms. In *Proceedings 2 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 42–51, 1991.

[3] Henry Baird. *Model-based Image Matching Using Location*. PhD thesis, Princeton University, oct 1984.

[4] R. Basri and D.W. Jacobs. Projective alignment with regions. *T-PAMI*, 23(5):519–527, may 2001.

[5] R. J. Beveridge. *Local Search Algorithms for Geometric Object Recognition: Optimal Correspondence and Pose*. PhD thesis, University of Massachusetts, 1993.

[6] R. J. Beveridge, C. R. Graves, and J. Steinborn. Comparing random starts local search with key feature matching. Technical Report CS-97-117, Colorado State University, http://www.cs.colostate.edu/, 1997.

[7] Thomas M. Breuel. Fast recognition using adaptive subdivisions of transformation space. In *CVPR*, pages 445–451. IEEE, 1992.

[8] Thomas M. Breuel. Branch-and-bound algorithms for geometric matching problems. Draft from the author, aug 2001.

[9] T.A. Cass. Feature matching for object localization in the presence of uncertainty. In *International Conference on Computer Vision*, pages 360–64. IEEE, 1990.

[10] T.A. Cass. Feature matching for object localization in the presence of uncertainty. Technical Report 1133, MIT AI Laboratory, May 1990.

[11] T.A. Cass. Polynomial-time object recognition in the presence of clutter, occlusion, and uncertainty. *ECCV*, 92:834–842, 1992.

[12] S.H. Chang, F.H. Cheng, W.H. Hsu, and G.Z. Wu. Fast algorithm for point pattern-matching: Invariant to translations, rotations and scale changes. *Pattern Recognition*, 30(2):311–320, February 1997.

[13] F.H. Cheng. Point pattern-matching algorithm invariant to geometrical transformation and distortion. *Pattern Recognition Letters*, 17(14):1429–1435, December 1996.

[14] H. Edlesbrunner. *Algorithms in Combinatorial Geometry*, chapter 7. Springer-Verlag, 1987.

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

[16] D. Gerson and S. Wood. Radius phase ii - the radius testbed system. In *Arpa Image Understanding Workshop*, pages 231–237, Monterey,CA, nov 1994.

[17] A. Goshtasby and G. C. Stockman. Point pattern matching using convex hull edges. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(5):631–637, 1985.

[18] W. E. L. Grimson and D. P. Huttenlocher. On the sensitivity of the hough transform for object recognition. *T-PAMI*, 1990.

[19] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[20] J. Hong and X. Tan. A new approach to point pattern matching. In *Proc. 9th International Conference on Pattern Recognition*, pages 82–84, 1988.

[21] X.P. Hu and N. Ahuja. Matching point features with ordered geometric, rigidity, and disparity constraints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1041–1049, October 1994.

[22] D.P. Huttenlocher and Todd A. Cass. Measuring the quality of hypotheses in model-based recognition. In *ECCV*, pages 773–777, 1992.

[23] Intel. Intel open source computer vision library. Software, 2000. http://www.intel.com/research/mrl/research/opencv/.

[24] D.J. Kahl, Azriel Rosenfeld, and Alan Danker. Some experiments in point pattern matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(2):105–115, feb 1980.

[25] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*, chapter Local Search, pages 454–480. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[26] D. Skea, Barrodale, R. Kuwahara, and R. Poeckert. A control matching algorithm. *Pattern Recognition*, 26(2):269–276, 1993.

[27] S.M. Smith and J.M. Brady. Susan - a new approach to low level image processin. *International Journal of Computer Vision*, 23(1):45–78, may 1997.

[28] J. Sprinzak and M. Werman. Affine point matching. *Pattern Recognition Letters*, 15(4):337–339, April 1994.

[29] J.P.P. Starink and E. Backer. Finding point correspondences using simulated annealing. *Pattern Recognition*, 28(2):231–240, February 1995.

[30] G. Stockman. Object recognition and localization via pose clustering. *Computer Vision, Graphics, and Image Processing*, 40(3):361–87, 1987.

[31] V. Vinod and S. Ghose. Point matching using asymmetric neural networks. *Pattern Recognition*, 26(8):1207–1214, 1993.

[32] D. Whitley, J. R. Beveridge, C. Graves, and C. Guerra-Salcedo. Messy genetic algorithms for subset feature selection. In *Proceedings of the International Conference on Genetic Algorithms*, pages 586–575, jul 1997.