Two Methods for Recognition of 3D Surfaces in the Presence of Obstruction and their Applications to Robotic Localisation

A thesis submitted in fulfilment
of the requirements for the degree of
Master of Science
in Computer Science
University of Waikato
by

MALCOLM J. LETT

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2006

Abstract

Recognition of three-dimensional surfaces is becoming increasingly important. One particularly interesting task is for a mobile robot to perform localisation using surface recognition on the three-dimensional structure of its immediate surroundings. Obstructions in such a setting makes recognition very difficult. Some research has been done on this topic but little work has achieved real-time efficiency while maintaining the success rates of recognition necessary for robotic localisation.

The most successful methods perform recognition by aligning a representation of the surface of the immediate surroundings with each of many surface representations taken from previous locations. The surface of a previous location with which the most successful alignment is found determines the robot's current location. The registration step is performed by finding matches between local features. Often feature points are chosen randomly, however this is too simplistic. Dividing the surfaces into planar regions using a noise tolerant decomposition algorithm is much more effective for feature selection.

This thesis describes two new methods, DMARA and DPSA, for recognition of surfaces using decomposition for feature extraction. Error handling methods are used for aligning two surfaces. DMARA uses an algorithm whereby many random searches are performed to align the surfaces. DPSA uses a generic point set alignment algorithm. When applied to the task of aligning two different decompositions of the same surface, DMARA achieved a 95% likelihood of success using 271 random searches. For the same task, DPSA succeeded for every set of parameters attempted. The results suggest the potential effectiveness of these new surface recognition methods for real world scenarios.

Acknowledgements

Credit goes to a number of people for helping me during this project and for making the creation of this thesis possible. Particular credit goes to my supervisors, Margaret Jefferies, Michael Cree and Mike Mayo.

First and foremost, many thanks to Michael Cree from the Waikato University School of Physics and Engineering, who took me on as his own student when Margaret, my given supervisor, was no longer able to help. Michael met with me for meetings which never kept to their allotted hour while juggling a high teaching workload of his own. He had a tremendous ability to understand my strange ideas and shared his own during our numerous discussions when I wasn't sure how next to proceed. Throughout, Michael was an immense help. Thank you, Michael.

Much credit also goes to my original supervisor, Margaret Jefferies, from the School of Computing and Mathematical Sciences, who helped me while I tried to figure out what my masters project would be about. She led me into the world of cognitive robotic mapping and guided me through the process of coming to grips with the existing research in the field and of where to find that research. Sadly, Margaret passed away before I could complete the thesis. Rest in peace Margaret Jefferies.

I thank Yann Michel for helping me a number of times to decide how to lay out figures, and for his tips on the use of LaTeX. I also thank David Milne for giving me a short article explaining how to write good introduction chapters.

I am deeply thankful to the Department of Computer Science, School of Computing and Mathematical Sciences, Waikato University, which gave me a departmental scholarship, without which this project would not have been possible.

Lastly, but most definitely not least, I want to thank my girl, Nithya Chandrasekharan, for being similarly mad enough to think that doing a masters degree was worthwhile, for supporting me during the whole of the year and during those times when I didn't feel I was making progress, and for generally keeping me at a sane level of insanity.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Mobile robots operate and navigate within a three-dimensional (3D) world with 3D objects of interest and 3D obstacles, yet most robots are required to locate themselves using only a 2D map of their environment. With increasing performance of 3D capturing methods, increasingly we will see robots relying on 3D information for reasoning about their environment. Solutions to tasks such as recognition of 3D objects and self localisation within their environment using 3D information are becoming more important.

Self localisation, often referred to as *ego*-localisation, has seen a lot of research using 2D representations of the *map* of the robot's environment. A lot of this research has focussed on Simultaneous Localisation and Mapping (SLAM) using 2D bird's-eye-view maps. SLAM involves keeping track of the robot's position while mapping, and poses a number of difficult problems. Of particular importance is the ability to detect *correspondences* between the robot's immediate current surroundings and its immediate surroundings at a previous time, indicating that the robot has returned to the same position.

Using only 2D information for detecting these correspondences is not effective enough. 3D surface information is much more description of the environment and this thesis presents two new methods for detecting such correspondences using 3D surface information.

The robot's record of its surroundings at all previous points in time can be thought of as a database, built up as the robot navigates. At each new position, the robot captures some representation of its immediate surroundings and stores this as an entry within the database, recording the map position along with the database entry. The task of ego-localisation is then performed by the robot recognising its current surroundings as being the same as its surroundings for one of the database entries. Recognising the correct database entry gives the location of the robot. Perhaps after some further processing, the robot can accurately calculate its position relative to the representation stored within the database entry, and thus determine its position in the world.

The majority of such systems use a laser positioned so that it scans horizontally, measuring the distance to points surrounding the robot on a single horizontal plane. However, using only this 2D information for the recognition task is problematic because it discards too much of the potentially available information and makes different parts of the environment look too similar for accurate recognition to be possible. Recognition mistakes caused by this can lead the robot into making the wrong decisions about how the map should be constructed and can further lead to other miscalculations and mistakes.

Capturing a 3D surface representation of the surrounding environment helps to alleviate this problem by providing the robot with more descriptive information. Systems are available for capturing 3D surface information using laser and sonar. These produce single range scans taken from a single viewing angle and methods exist for combining many scans to produce accurate 3D surface representations of, say, the robot's surroundings. A further capture method, which is now beginning to become feasible, is to reconstruct the 3D surface information from 2D images acquired from one or more cameras. Although different arrangements exist, the simplest is to use two cameras positioned side by side and synchronised to acquire frames at the same time. A single pair of frames is used to reconstruct the 3D surface of the visible part of the surroundings.

A captured 3D surface of the same set of objects will look very different when captured from different angles. Objects visible when viewed from one angle may be only partially visible when viewed from another angle or may even be totally hidden because another object is now in front. This is called *obstruction*. This is apparent when using both 2D capturing and 3D capturing methods, and it is one of the most significant reasons why recognition of the robot's immediate surroundings is difficult.

Robotic localisation using 3D surface information is performed by aligning the 3D surface representation of the current immediate surroundings with the surface representation within each entry of the robot's database of known locations. The database entries are annotated with the real-world position from which they were captured, and so the database entry to which the current surface is best aligned indicate the robot's current location. If the surface representation of the current surroundings cannot be satisfactorily aligned to any database entry, then the robot considers itself to be in a new location. Assuming a static environment, the alignment process produces a rigid transformation of the coordinates of the current surface onto the database entry's surface. That is, a rotation and a translation representing the relative *orientation* of the two surfaces, and indicating the *pose* of the robot relative to the database entry's surface. Finding the alignment between the current immediate surroundings' surface and the surfaces within the database is the most effective means of recognising the current surroundings. Only poor results have been achieved otherwise.

The task of finding the best alignment between two 3D surfaces, known as *registration*, is a very hard problem because of noise inherent within the capture process, occlusions, and the large amount of data stored within a 3D surface representation. The majority of methods available for recognising 3D representations focus on complete 3D models where occlusions, or *partial matching*, are not permitted, or they only handle statistically small amounts of occlusion. Typically, the best solutions randomly select a predefined number of points within the 3D surface and extract some locally measured feature vector for each of those points. This is done for all surfaces, creating a list of feature points for each surface within the database and for the immediate surroundings. An algorithm is then employed to find the best matches between feature points in the current surface to feature points in a surface from the database, known as the *correspondences* between the feature points, and this is used to determine how closely the two surfaces can be registered (note that "correspondence" here is not used in the same sense as in the context of the Correspondence Problem of robotic localisation). But randomly selecting points is not very effective because it requires a large number of points in order to find a suitably sized set of correspondences which are consistent in that they agree on the relative orientation of the two surfaces, and most existing systems are too computationally expensive for real-time use, or are too inaccurate.

Methods exist for better selecting locations for extraction of features. One such method is to divide the surfaces into regions of approximately planar surfaces in a process known as *decomposition*. These regions can be used to better control the relative importance that regions are given for feature extraction. Furthermore, naturally occurring features such as edges can guide the decomposition process giving a much better chance of feature points lining up to their corresponding feature points within the database entry's surface. For example, randomly selecting points on two copies of the same surface will yield different point positions, but decomposing each copy of the original surface and then selecting the centre points of the decomposed regions will result in the same positions being chosen for both copies.

This research project attempted to address the problem of efficiently registering two surfaces for the purpose of 3D surface recognition within robotic localisation. Two methods were attempted. The first method used an efficient decomposition method from existing literature and combined this with methods for feature matching to aid the search for the registration solution using the popular RANSAC algorithm. The second method treated the centres of the decomposed regions as unlabelled points in space, forming a 3D point cloud. Any two surfaces were thus represented as two sets of unlabelled point clouds and an algorithm was applied which simultaneously finds point correspondence and point set transformation information.

This thesis covers the background of research done within the field of robotic localisation, explaining the reasoning behind the choice of solutions employed, and

gives a detailed description of those solutions. Experiments analysing the effectiveness and the efficiency are given. An indication of the actual execution times on a standard PC are also given.

1.1 Research Goals

In presenting this work, three important research questions are examined.

The first question considers how decomposition can be used to extract features. Decomposition can lead to many different feature representations. One method is to directly use the decomposed region as the feature representation. The registration process requires finding matching features. The decomposed regions can be matched using some form of similarity measure. Alternatively, the boundary of the region can be used, rather than storing the whole surface structure of a region. Another representation is simply to use the coordinates of the centres of the decomposed regions, discarding information about the shape of the regions and relying instead on the distances between regions, for example.

The second question asks what the likelihood, or success rate, is that the correct registration is found using these features and the associated registration methods. Some methods can only register surfaces which have very low noise, others tend to find sub-optimal solutions. The consequence is that no method can perform perfectly 100% of the time and the likelihood of decomposition based registration methods to succeed must be measured.

The final question seeks to determine how efficient registration is when using decomposition based methods. The ultimate task is for the registration approach to enable a robot to navigate and self-localise. This requires that the registration method can be executed quickly. The robot should not be expected to wait a long time to determine its location.

The research questions are summarised below:

1. How can decomposition be used to extract features for the purpose of registration and recognition?

2. What success rate can be achieved for registration using features extracted from decomposed regions?

3. How computationally efficient is decomposition based feature extraction and registration?

1.2 Thesis Organisation

This thesis is organised as follows.

Chapter 2 describes the current popular approaches for SLAM and existing work on using 3D recognition for localisation. A detailed coverage of concepts used within

the thesis are presented and some existing work for 3D surface recognition, decomposition and other related topics is highlighted. It also presents the rationale behind the approach taken within this thesis, examining the problem of surface recognition and leading into a number of specific algorithms which are combined to form the final solution. Chapter 3 presents a review of some existing literature within the research areas covered by this thesis. Four important algorithms from existing literature are presented within Chapter 4. These algorithms form the basis of the approaches used here.

Chapter 5 describes the implementation of two new registration methods which can be used for the purposes of 3D surface recognition and robotic localisation. Tests for these methods are presented within Chapter 6. The data set used is described and a set of goals are established, followed by the description of the experiments and their results.

The results of Chapter 6 are examined within Chapter 7 and conclusions drawn on the effectiveness of the presented methods. Finally, areas for improvement are highlighted at the end of Chapter 7.

Chapter 2

Background

Recognition of 3D surfaces is increasingly becoming an important task. Increasing numbers of 3D models exist within publicly accessible databases within the internet. Designers using Computer Aided Design (CAD) software need the facility to search their designs and perform other high level tasks on 3D models, such as including a 3D model from another source and merging it with their existing model but without having to perform the alignment phase themselves. Autonomous robots must operate in a 3D world and thus should be capable of reasoning about the 3D objects within its environment, and this requires recognising those objects.

The approach presented here can be used for any of these tasks, however the focus is on the task of recognising 3D surfaces for Robotic Localisation.

2.1 Robotic Localisation

Robotic localisation is important in many processes used within an autonomous or semi-autonomous robot. A robot must be able to detect its current position in order that it may know where it should move next, and it must keep aware of its location as it moves. Some designs use GPS or other tracking technology external to the robot, others require that the robot capable of learning its position using sensors and a pre-programmed map. Still other designs require that the robot detect its position using sensors and a map which it has learnt itself.

By far the most interesting design is one where the robot is required to learn a map of its environment and to self localise whenever needed, including during the process of learning the map, without input from external sources. The map usually represents a bird's-eye-view of the boundaries and obstacles forming the environment in which the robot moves; such as corridors, rubbish bins, chairs and tables, within an office environment. The process of autonomously learning the map while navigating is known as Simultaneous Localisation and Mapping (SLAM) [43] and is considered a hard problem in robotics.

Most robots are built using some form of sensor equipment, such as laser, camera,

sonar, and pressure plates, and usually use wheels to navigate. Often a combination of sensor equipment is used, such as using laser or camera to capture information for the map, using sonar to detect obstacles and avoid collisions, and using pressure plates to detect when the robot has had a collision. Most designs are unable to detect holes in the ground and thus the motion of robots is usually restricted in some way to avoid such things as stairs.

There are two types of robotic localisation: *local localisation* (or *tracking localisation*), and *global localisation*. Measurements from the robot's drive mechanism (such as wheels) are used for odometry readings to give an approximate position of the robot, however these odometry readings are prone to error. Local localisation is concerned with correcting the odometry readings. This technique requires approximately knowing the initial position of the robot. The robot uses some form of sensor data, such as a horizontally scanning laser, to measure the world surrounding it and to compare to sensor data from the previous known position. This comparison, using the approximate odometry readings, locates corresponding features in both sets of readings and thus computes the robot's new position. However, should the robot lose track of its position, it is unable to recover.

Global localisation is more difficult than local localisation and is used when the robot's position is not known, perhaps because it has lost track of its position from insufficient data. There are two particularly interesting problems where global localisation becomes important:

- The Kidnapped Robot Problem
- The Correspondence Problem

A robot is *kidnapped* when it is moved while switched off, or its sensors malfunction momentarily, leaving the robot unaware of its new location. In order to recover, and assuming it has a map of its current surroundings, the robot must recognise the new environment using its sensors. This can be a hard problem or a very hard problem, depending on the type of sensors the robot has.

The *Correspondence Problem* [86, 9] is more subtle. During SLAM, the robot must detect when it returns to a point previously observed. It may arrive back at this point because it has turned around and gone back the same way, or it may have returned through a root previously unobserved. An example of the second reason is when the robot is mapping an office block or apartment block with corridors which circumnavigate some central region. Due to errors in odometry and the mapping process, the robot may think that it is traversing a spiral, such as shown in Figure 2.1. Here, the top plot indicates the desired map, however a robot may produce something akin to the bottom plot as its odometry consistently indicates a motion with a small error in one direction away from its actual path. The robot needs to detect that the section of map which it is currently capturing *corresponds* to a section of the map which it has already observed and stored. This task is complicated

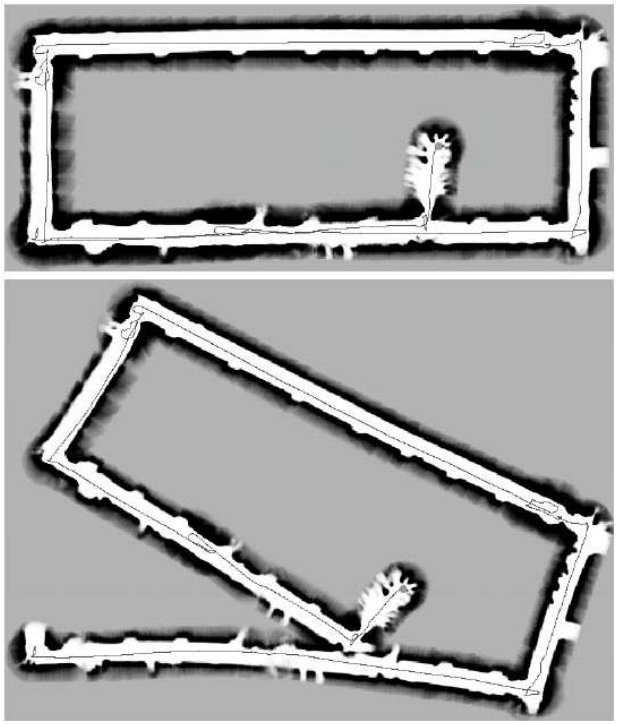


Figure 2.1: Correspondence problem in SLAM
Top: the desired map after starting within the side room, navigating around the main corridor and going past the room.
Bottom: an example outcome when odometry readings have a small error and correspondence is not detected. The thin line indicates the path of the robot.
(Source: S. Thrun, 1998 [86])

by the fact that the measurement error accumulated over the course of the mapping so far can become unboundedly large. A further problem is that the presence of people cause the environment to be constantly changing, making localisation even harder. However this problem is often ignored and will be likewise ignored here.

2.1.1 Map Representation

The representation used for the map can have an effect on how difficult it is to build a map during SLAM, how accurate the map needs to be, how difficult it is to detect correspondences, and how difficult it is to perform global localisation. There are two popular types of representations in use today: *global absolute metric maps* and *topological maps*. A global absolute metric method [89, 28] represents the entire world encountered by the robot using a single, self consistent, metric map in a single reference frame. Figure 2.1 is an example of a global absolute metric map. A common form of metric map is an *occupancy grid map*, developed by Elfes and Moravec [30, 63].

Topological maps [13, 98, 47, 90] attempt to represent the map in a less rigid manner by partitioning it into separate regions, or *local spaces* [43], and storing them as connected partitions. Each separate partition is often represented by an absolute

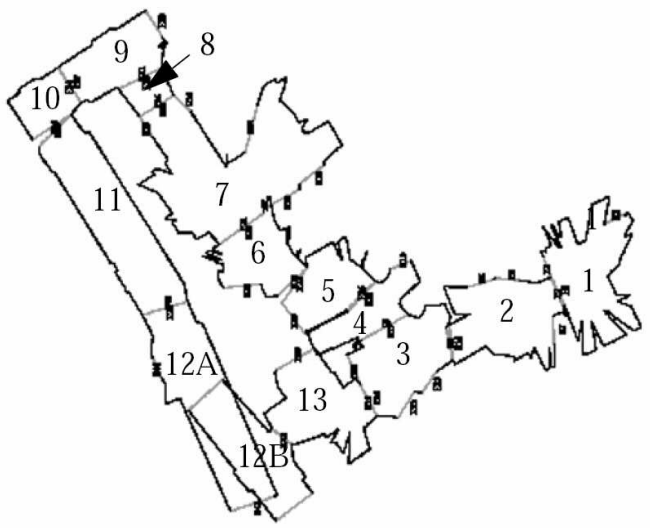


Figure 2.2: Correspondence problem in SLAM
Illustration of a topological map. Each labelled region corresponds to a single ASR (partition). The grey lines represent the exits forming the connections between ASRs. ASR 12A is visited from ASR 11, ASR 12B is visited from ASR 13. (Source: Yeap and Jefferies, 1999 [98])

metric method and the connections are found as the robot navigates from partition to partition. The advantage of topological maps is that global consistency between partitions, as they are now represented, is not required and local measurement errors are constrained to within the partitions. The partitions usually represent such things as parts of corridors and rooms within structured indoor environments.

The correspondence problem must be handled in both types of representation. Robots using global absolute metric maps usually use such methods as Kalman Filters [60, 45, 81, 80] or Particle Filters to localise themselves within the map and have to perform computationally expensive processes to detect correspondence due to the large amount of sampled data present within metric maps. And when a correspondence has been found, all points within the map must be updated. Topological maps have the advantage that the discovery of correspondences requires only updating the connections between partitions, and perhaps the two corresponding partitions, but does not require updating the data representing the unaffected partitions. Detecting correspondences can be easier too as the robot needs only to match the current partition with the correct corresponding partition.

2.1.2 Global Absolute Metric Map based Solutions to the Correspondence Problem

The correspondence problem is one which was initially ignored by robotic mapping research, being considered too hard by most practitioners, however a lot of research has emerged within the last nine years. The most popular approach used for global

absolute metric maps is Kalman filtering [45, 60].

Kalman filters are Bayes filters which represent posterior probabilities of the robot's *pose* (position and orientation) and points within the map. The posterior probabilities are updated in a linear fashion each time the robot moves. Kalman filters, however, suffer from the assumptions necessary to make it feasible to calculate. The measurement noise, both of map coordinates, and of robot pose, is assumed to have a gaussian (normal) distribution, which is rarely the case. Secondly, robot motion is usually curved, not linear. Non-linear motion is usually handled by expanding the motion into several smaller linear steps, and the non-linear sensor noise is sometimes handled in a similar fashion. A further problem arises because the update phase requires $O(K^2)$ time, where K is the number of map features. Some newer versions of Kalman filters are capable of reducing this complexity, such as the FastSLAM algorithm [62, 40] which is capable of reducing the complexity to $O(\log K)$ in some situations.

More recent work extending Kalman filters is based on the Lu/Milios algorithm [39, 38, 55]. A recent alternative to Kalman filters is based on the *Expectation Maximisation* (EM) algorithm [26]. It applies two steps iteratively. The first, the E-step, updates the posterior over robot poses for the current map. The second, the M-step, calculates the most likely map given the previous known map, the new map information, and the posterior of pose. Much successful work has been achieved using EM algorithms in situations where Kalman filters fail. EM algorithms recalculate the pose and all points within the map at each stage and consequently are capable of handling the correspondence problem in much more complicated environments than Kalman filters. However, the EM algorithm's main drawback is its computational complexity. It must be run as an off-line process and may take many hours to calculate the resultant map on a standard PC.

Thrun, 2002 [87], provides a good survey of common mapping approaches to date, including a clear description of Kalman filtering and the EM algorithm, and how they are applied to localisation, particularly when using occupancy grids.

More recently, Lowe, Little, and Se [52, 51, 53] developed a system called the *Scale-Invariant Feature Transform* (SIFT) [54], which uses features extracted from visual input acquired from three cameras. These features, called *visual landmarks* when extracted from visual inputs, are processed in such a way as to be largely invariant to scale, so that a feature found in one image will also be found in another image taken at a different distance and the two will match. Lowe *et al.* show how SIFT features can be used to perform global robot localisation. Correspondences between features from each of three cameras are found, enabling calculation of the 3D real-world coordinates for each SIFT feature through well understood homographies [69]. The map is represented as a 3D point cloud, where each point is a single SIFT feature. Localisation is performed by matching the current 3D point cloud map to a database of other point cloud sets observed at previous locations, incor-

porating an approach known as the RANSAC algorithm. The RANSAC algorithm is described in detail in Chapter 4.

2.1.3 Topological Map based Solutions to the Correspondence Problem

As already discussed, topological maps help somewhat in addressing the problems encountered when using global absolute metric maps. By representing the map using two distinctive and separate concepts – connections and local spaces – the computational complexity can be reduced by concentrating on one part of the representation.

Yeap and Jefferies [98] describe how a single *Absolute Space Representation* (ASR), representing a single partition within a topological map, can be computed using data from a horizontal laser. The ASRs are simple representations of enclosed regions, usually constructed as sets of lines forming the perimeter of the ASR from a bird's-eye-view. Potential "exits" are marked by detecting obstructions and other discrepancies within the laser data. Consequently, some "exits" may simply lead to areas behind obstructions and which are otherwise part of the previously observed ASRs but will be formed into separate ASRs.

The correspondence problem is addressed by recognising single ASRs. Figure 2.2 illustrates how the same region can be encountered from two different directions. In this example, region 12 is encountered from both ASR 11 and 13 and some means is required for recognising that the two variants of region 12 correspond to the same region. Jefferies *et al.* give a solution to this in [43]. They use two techniques simultaneously for recognising ASRs. In the first technique, a global metric map is computed alongside the topological map by placing the ASRs together (as in Figure 2.2). If two ASRs overlap significantly then they are deemed to be the same. The second technique uses single-output neural networks, one for each ASR, to learn the signatures for the ASRs. A vote is then performed to select the most likely matching ASR to one just captured and a threshold applied to detect new ASRs. The networks' inputs are based on the lengths of lines forming the perimeter of the ASR and the angles between those lines.

The global metric map based technique suffers from occasionally producing false positives (falsely recognising ASRs as the same when they are in fact not the same). This can occur when measurement errors in the robot's pose cause the current ASR to *seem* to be overlapping an ASR that it should not. For example, if the measurement error was sufficiently bad that ASR 12B overlapped ASR 11 in Figure 2.2. This technique can also suffer badly from false negatives (not recognising ASRs) when the measurement error is so bad that ASRs don't overlap when they are really the same region. The neural network based approach attempts to resolve this by enabling the robot to consider more ASRs already observed for recognising the current ASR. However, recognising such simple 2D representations of environments

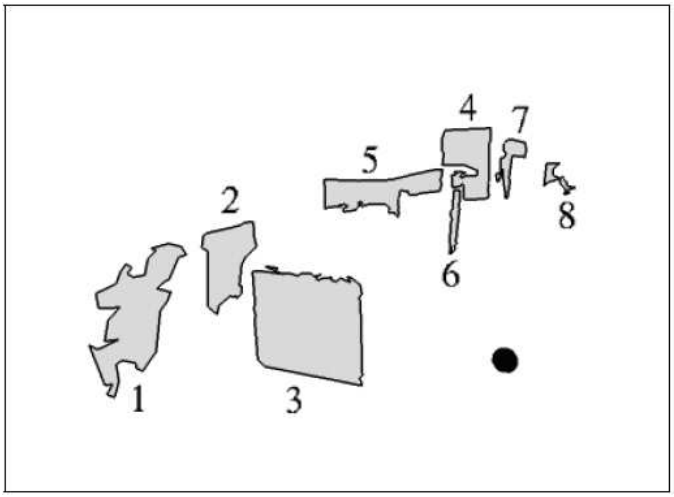


Figure 2.3: Baker's 3D ASR representation
Extracted planar surfaces projected into 3D. The black circle represents the position of the robot, which is approximately facing towards region 2. Regions 5, 4, 7 and 8 are to the robot's right, regions 1 and 3 are to robot's left.
(Source: Baker, 2004 [9])

can be very prone to error, both false positive and false negative.

Baker [9] attempts to resolve these issues by incorporating 3D scene data into the ASR representations. The combination of a horizontal laser and a camera are used to extract rudimentary visual scene features and project these into 3D coordinates. A Harris [41] corner detector is used to detect features within the camera image and correspondences are found between corners in the previous view and the current view. Segmentation is applied to the image and correspondences found between segmented regions, or "blobs", within the same two views. If a blob contains sufficient corresponding corners to project it, then those corners are used to calculate the blob's coordinates in 3D space. If, however, insufficient corners are found within some blobs, then those blobs which intersect the plane in which the horizontal laser scans are assumed to be vertical and are projected into 3D space using the distance measurements from the laser scan data. Those segmented regions neither containing sufficient corners nor intersecting the laser scan are rejected. The final 3D representation contains a number of irregularly shaped planes in 3D space, as illustrated in Figure 2.3.

3D ASR representations are compared using a *histogram correlation* approach by Rofer [74]. The sum of lengths of line segments forming the perimeter of the planar regions are binned into a histogram based on the line segments' orientations. When comparing two regions, their histograms are aligned to match the relative rotation of the two regions, and finally a measure of similarity is computed. This measure is used to compute an measure of similarity between two 3D ASRs. Other measures, including comparing the colour of the regions, are also tried.

Baker's method attempts to extract 3D spacial information and represent this

with low computational overhead. However this approach leads to a number of problems. Firstly, only planar surfaces are kept as part of the representation and regions which have high changes in gradient across their surface are removed as part of the segmentation algorithm. Those regions remaining are then assumed to be planar – often a valid assumption within structured office environments. Secondly, it is hard to correctly find corresponding corners between two consecutive views because corners alone provide little feature information. Thirdly, Baker's method relies on regions extracted during a segmentation process. Segmentation is strongly affected by noise in images, making it very difficult to achieve consistent segmentations between consecutive image captures of the same physical environment. This makes the region matching process very unreliable. Lastly, the information retained after eliminating non-planar regions, *et cetera*, is only a small portion of the information available from the images. Baker found that his method was not discriminating enough, producing many false positives, while it also performed poorly at recognising ASRs when they were the same.

2.2 Surface Recognition for Robotic Localisation

As hardware technology has improved the solutions available to the problems of Robotic Localisation have increased. The original methods relied on 2D data acquired from laser or sonar systems, and later 3D data has been used, usually acquired from sonar or laser. The recent trend is increasingly to rely on vision techniques instead of laser. Vision techniques have two important advantages. Firstly, cameras are usually cheaper than laser hardware. Cheap cameras and video capture boards are readily available as off-the-shelf the components. Secondly, both laser and sonar are *active* methods, while vision systems can be *passive*. Active capturing techniques, such as laser and sonar, interact with the environment. In some cases this interaction is unacceptable. Laser systems emit light rays (sometimes at frequencies invisible to the human eye) and rely on the reflected rays to measure depth. Laser systems are ineffective in large open areas because the reflected rays have very weak signal strength and are thus difficult to detect.

Baker [9] attempted to address the localisation problem by introducing partial 3D information into an existing 2D map representation. The methods presents in this thesis extend that system so that full 3D surface information is used. Various possible approaches can be taken which use 3D information, each having their own inherent difficulties. For example, what is a suitable representation to use? The storage space required, the computational efficiency for processing the data stored in the representation, and the ease of performing such operations as merging, registration, and recognition, must all be considered. Once that representation is chosen, how is recognition performed? If a feature based method is used, how are features extracted? How are those features compared against each other?

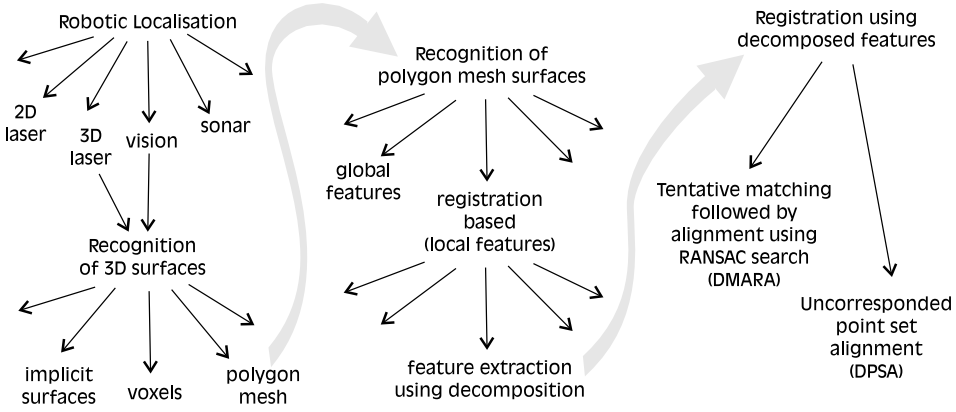


Figure 2.4: The robotic localisation solution space
Many possible solutions exist. This thesis takes one path which seems a sensible approach. The result is two methods using decomposition of surfaces.

While no one set of correct answers exists for these questions, one possible set of answers is now given. This section serves as a justification for the methods presented in Chapter 5. The relation of the methods presented here with respect to other possible solutions can be seen in Figure 2.4.

Any capture method, such as laser, sonar, or vision techniques, can only determine information about the surfaces of the environment surrounding the robot. Full volumetric information about the objects which make up the environment is unavailable and can only be extracted by means of an object recognition system. Thus it seems evident that a 3D model that represents surfaces as an inherent part of its structure is desirable over an approach that represents the environment using volumetric information, such as voxels (described briefly in Section 2.3.2). Furthermore, the surface representation scheme should be flexible enough to model complex surfaces with any degree of complexity.

Polygon meshes are chosen as a suitable representation. They are a common choice for many tasks because they are well understand and many years of research has resulted in very efficient algorithms for the various tasks performed on polygon meshes. The versatility of polygon meshes has resulted in their extensive use within Computer Graphics. Most modern personal computers now come with Graphics Processing Units (GPUs) which use a high degree of parallelisation to perform operations on polygon meshes in a fraction of the time necessary for those operations using the Central Processing Unit (CPU). It stands to reason that, if a polygon mesh representation is used, it may be possible to take advantage of the GPU to perform some of the computation tasks necessary for surface recognition. For example, some of the initial processing required for the Watershed Decomposition algorithm described in Chapter 4 can be performed as matrix operations within the GPU.

2.2.1 Recognition of 3D Models

Once the robot has captured its immediate surroundings as a surface representation such as polygon mesh, the surface representation must be recognised in order that the robot can know its location. Surface recognition approaches can be grouped into two broad categories: "global feature based" and "local feature based". Both of these methods extract *features* which further represent the surface representations, but using a much reduced set of data. Recognition can be performed more efficiently by reducing the size of the representation used.

Global feature based approaches often calculate statistics over the entire model, such as distributions of surface-to-surface angles and point-point distances [66, 94], and others use curvature measurements [78]. Some approaches use transforms such as converting to the frequency domain or to a 2D representation [97]. Some methods further process these statistics and produce 1D or 2D histograms [66]. These histograms can be matched very efficiently. Global feature based approaches are best suited to the task of finding matches to a 3D model within a database of many CAD generated models, where the reference model and the database models are single objects without the presence of clutter due to other objects and when the models are complete. A complete model has no holes in its surface.

When a robot enters a room, it will observe many partially observable objects. Parts of those objects are *occluded* by other objects or because the robot cannot see all sides of the object. Recognition of such objects requires the ability to perform *partial matching*. Global feature based approaches give poor results when partial matches are required. Of the research found by the author, 20% missing/extraneous data is the most any global recognition system could handle.

Alternatively, local feature based approaches attempt to individually match many features extracted from local regions. Local features can measure similar things to the global features, except that only points within a local region are considered [21, 44]. An alternative approach is to convert the surface into a graph representation such as Reeb graphs [91], however these require surfaces encasing obvious volumes and only some methods can cope with surfaces which are not "water-tight" (having no holes).

The final recognition result is determined based on the success of the individual feature matches. When partial matching is not required, a sufficient measure is to measure the percentage of local features that are matched between the observed scene and a database model. However, in the presence of occlusion, any process comparing against the whole can lead to mistakes, particularly if caution is not taken. Let us consider an example in order to illustrate a potential problem and its solution.

Consider attempting to recognise an observed object, m, against two models, d_1 and d_2, within a database. In this case, d_1 is considerably larger than both m and d_2

and as such has a much larger number of local features than the other two models. Models m and d_2 have similar sizes and a similar number of features. Now, let us say that 90% of model m features have a successful match with features within model d_1, but since d_1 has many features, those matched features only comprise 20% of features from model d_1. When comparing against model d_2, however, 50% of features from m matched to 45% of features within model d_2.

Now, if we use the percentage of features successfully matched within the database models, we would choose model d_2 as the winner, having 45% of features matched, instead of the 20% for the other database model. But we would likely be drawing an incorrect conclusion simply because of scales – we had ignored that d_1 has more features than d_2. In this case the solution is obvious, we should use the percentage of features matched within model m instead, and thus would choose d_1 as the winner. Analysis of this simple example leads to an important rule of thumb:

> *When determining the best overall match, do not calculate match ratios or percentages by dividing by parameters which are different for each of the potential matches, and if necessary use absolute values rather than ratios.*

In our example we divided by a large number for the measure against the first database model, but divided by a small number for the other. We resolved our mistake by dividing by a parameter which was the same for both database models: the number of local features within model m.

So far we have looked at a naïve local feature based recognition approach. Unfortunately, this approach cannot be relied upon for accurate results and is best used as an initial step to eliminate the most unlikely matches. A further requirement on the correspondences between the local features of two models is that the correspondences must lead to a transformation from points in one model to points in the other which is composed purely of rotation and translation and the addition of a small amount of noise. This is called a *geometric constraint*. The consequence is that the correspondences should not "criss-cross", but this can easily arise because matches between features are not always correct.

It may be noted that the problem caused by incorrect matches between features is dependent on the level of detail used to describe each feature. Low levels of detail lead to features which are more likely to have many similar matches, whereas using high levels of detail can reduce this effect substantially, and thus improve the chances of resulting in a correct conclusions based only on the number of well matched features. However, it is almost impossible to define unique identifiers for features and thus geometric consistency must be ensured through some method.

With geometrically inconsistent correspondences, determining whether two surfaces are the "same" (or at least partially overlap), must be done by first *registering* the two surfaces. This registration process finds the rotation and translation

transformation which represents how the two models are positioned relative to one another. The accuracy with which the surfaces are aligned is then used for recognition, rather than directly using feature matching. This form of transformation is called a *rigid body transformation*; because the models themselves are not deformed in order that they can be aligned better.

2.2.2 Surface Registration

Registration of complex 3D surfaces can be difficult to do both efficiently and accurately, and so the process is usually broken into two phases: *course registration* followed by *fine registration*. Course registration is quick, but inaccurate. It produces a registration which is (hopefully) close to the desired registration using approximations and heuristics, often using a guided search technique. Fine registration is capable of registering two surfaces very accurately, and relatively efficiently, provided that the surfaces are already registered closely. Thus the result from the course registration is used as an input for the fine registration.

The common approach used for fine registration of polygon meshes is the Iterative Closest Points (ICP) algorithm. ICP was originally devised by Besl and McKay [10] and was followed by a lot of research improving the algorithm to make it more robust against noise and partial overlaps. For example, the original ICP algorithm does not handle partial overlaps well and Chetverikov [20] extended ICP to handle partial overlaps and noise in a better manner. Fitzgibbon [34] presented an alternative approach to ICP, based on non-linear minimisation using the Levenberg-Marquardt algorithm. Fitzgibbon claims that it is more robust and effective than ICP based methods. He showed that his approach could register two copies of the "stanford bunny" [1, 2] from an initial offset angle within a range of about 140 degrees whereas traditional ICP would only succeed over a range of about 55 degrees.

Unlike fine registration, course registration is much harder, and more dependent on the representation scheme, the types of objects being modelled, and the application. Most commercial systems to date rely on human interaction to obtain a course registration. The reason is that registration of 3D surfaces can be very computationally expensive. Surfaces can often contain many thousands of control points and the computational cost of registration can increase exponentially with the number of points used.

The simplest approach is to randomly select a number of local features in each of the surfaces and to perform some sort of search method to find the rigid transformation between the surfaces. Most approaches attempt to reduce the search space by first finding an initial correspondence between local features in one surface to local features in the other, as illustrated in Figure 2.5. But this brings us to the same problem encountered when attempting to perform surface recognition.

The problem is of outlier removal. Some matches between local features are

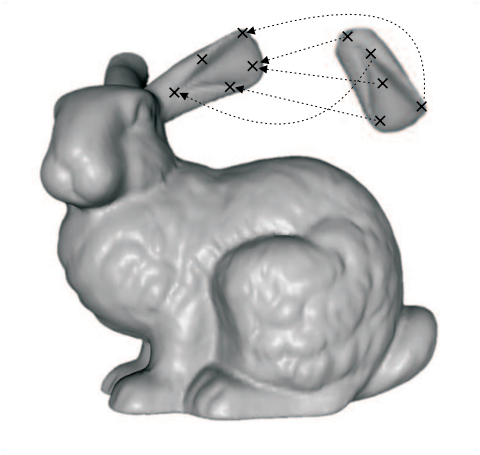


Figure 2.5: Feature correspondences for surface registration
Local features on each of the bunny and the ear are marked with crosses. Matches from ear features to bunny features are indicated by arrows. Some matches are incorrect.

incorrect. These are the outliers which must be removed in order to find the geometrically consistent set of correspondences and calculate the rigid transformation. A number of different approaches exist for this. The features used by Johnson and Hebert [44] enabled them to perform a geometric consistency search on the feature correspondences. Features were randomly selected and incrementally added, rejecting those feature correspondences which caused greatest deviation from the previously calculated transformation.

The RANSAC algorithm of Fischler and Bolles [33] is a very effective and generic outlier removal algorithm. RANSAC has been successfully used by Lowe *et al.* [51] to perform registration using two labelled point clouds. The points represented features extracted using their Scale-Invariant Feature Transform (SIFT) and were labelled with the feature values. The RANSAC algorithm was used to randomly search the list of feature correspondences and to find a rigid transformation which accepted the majority of correspondences.

Gold *et al.* [35] developed a method for fast registration of unlabelled point clouds without known correspondences. Instead, the correspondences are simultaneously calculated with the transformation calculation, using an iterative method.

In this thesis, two registration methods are attempted, based on the two approaches just mentioned. In the first method, a RANSAC based search is used to find the consistent set of feature-to-feature correspondences. In the second method, Gold *et al.*'s method is used to find the rigid transformation while simultaneously finding the correspondences. The main difference to the approaches described above is in how the features are extracted. The problem of feature extraction is now investigated. In doing so, an alternative method for feature extraction will be arrived at.

2.2.3 Feature Extraction

By reviewing the literature on 3D model recognition, two things are apparent. Firstly, to recognise surfaces with only partial matches, a large set of local features are required, and recognition is best performed by attempting to maximise the number of recognised local features. Secondly, no standard means of computing local features exist for 3D surface models. Most research is done using *ad hoc* feature extraction techniques and features are often selected at random points over the surfaces.

Random feature selection can lead to poor selection of features if only a small number of features are chosen. For example, with only a very small number of features, the distance between each feature point becomes very large. Consequently, the points in either surface will not align well except by chance. In order to reduce the average distance between points and make more accurate alignments possible, many feature extraction methods attempt to cover some percentage of the total surface area by the cumulative area of the regions covered by the sampled features. However, many matching algorithms scale exponentially with the number of points, and none to date perform better than scaling with the square of the number of points (assuming roughly the same number of feature points from either surface).

A better approach is to be more selective about where local features are extracted from. Locations should be chosen so that they are located at salient features within the surface, such as edges or distinctive curve regions. One simple method for selecting such features is to divide the surface into separate distinct regions. In an environment with a number of large planar surfaces, such as is found in structured office environments, a very suitable feature is a planar surface region, rather than a single point. If the surface is divided at regions of high curvature, the resulting set of regions will be predominantly planar. The process of dividing into separate regions is called *decomposition*.

2.2.4 Decomposition

Decomposition is the 3D analogue to *segmentation* in 2D images [27, 50]. In fact, one well known 2D image segmentation algorithm, the Watershed algorithm [77], has been extended to the task of decomposition of polygon surfaces by Mangan *et al.* [57], and Papaioannou *et al.* [68] appear to have independently devised a decomposition algorithm very similar to Mangan *et al.*'s work.

Decomposition is usually performed by detecting the curvature of surfaces and breaking into separate regions when deep concavities are found [46, 57, 99], or by finding convex regions [19]. Zuckerberger *et al.*'s [99] method, based on Mangan *et al.*'s work, performs watershed decomposition on the surface using angles between adjacent faces to determine curvature. Their method is very efficient, and with care can be implemented in such a way as to have $O(n)$ computation efficiency

in the number of faces, n. A detailed description of Zuckerberger *et al.*'s method is given in Chapter 4.

The result of a decomposition process is a set of arbitrarily shaped regions called *patches*. The patches cover the whole of the original surface.

2.2.5 Matching Decomposed Regions

For decomposition to be successful, some method for matching patches in one surface to patches in another surface must be used. This requires a method for calculating the similarity of patches. One method is to directly compare patches based on their boundaries. The "Turning Angle Functions" [6] and the "Earth Movers Distance" [75] are two possibilities.

An alternative is to encode the general shape of the patches by some low dimensional *patch descriptor*. Osada *et al.*'s "Shape Distributions" [66] represent the shape as the probability distribution of some geometric characteristic of the patch. One geometric characteristic is to measure the distance between points and so the shape distribution then represents the probability distribution of distances between points on the patch. Shape distributions were shown to be effective at recognition of 3D models in the presence of noise. Despite being intended for 3D models, shape distributions are easy to apply to patches by measuring the distances between points on the boundary of the patch.

2.2.6 Section Summary

We are now in a position to describe the whole of the process used within this thesis for the task of surface recognition.

- Polygon meshes are used to represent the surfaces because they are easy to use, are well understood, and there is the possibility of using the GPU for computations.
- Recognition is performed by taking each surface from a database of previous locations and registering the surface of the immediate surroundings to that database surface. The database surface to which the current scene's surface is registered most accurately is chosen as being representative of the robot's location.
- Features are extracted by performing decomposition using Zuckerberger *et al.*'s version of the Watershed decomposition algorithm.
- The decomposed surfaces form patches which are matched using Osada *et al.*'s Shape Distributions.
- Shape distribution based matching is used to create a set of tentative matches (the patch correspondences) which contain any number of false matches.

- The first registration method then applies the RANSAC search algorithm to find the geometrically consistent rigid transformation, giving the relative orientation of the two surfaces.
- The second registration method uses the patch features directly without the need of the tentative patch correspondences.

2.3 Concepts Covering Areas of Research within Thesis

This thesis covers a number of separate areas of research within Computer Science. The work presented focusses on the solution for Robotic Localisation and the Correspondence Problem, and some aspects of these research areas have already been touched upon. It is important to briefly cover the subject of data capture. There are a number of different means being used for capturing 2D and 3D data for the purposes of Localisation, each with their advantages and disadvantages. It is also important to explain briefly some of the 3D surface and volume representations in use today. These topics are discussed within sections 2.3.1 and 2.3.2.

2.3.1 Data Capture for Localisation

The most effective research done to date on Robotic Localisation has relied on laser data [9, 43, 98, 89, 88, 56, 23]. Laser has tended to provide the most accurate data and means exist for modelling the noise [37]. Laser is also the easiest to work with, as it immediately provides depth information. Predominantly only 2D lasers have been used in initial localisation work [9, 43, 98], producing the sort of results as seen in Figure 2.2.

More recently, 3D laser data has been used successfully to perform SLAM [88]. Some methods use full 3D lasers, while others [89, 88, 56] use two 2D lasers: one on the horizontal axis used to perform the usual 2D laser based SLAM, the other oriented on the vertical plane. Data from the vertical laser is registered and combined automatically as the robot moves. Since 3D laser produces a vast amount of dense 3D information, systems using 3D laser data usually simplify the data to sets of planar surfaces [89, 56] to reduce the amount of data to be processed for localisation and mapping purposes.

Laser based approaches, however, are expensive, and cheaper systems are beginning to be developed based on visual approaches. Some systems incorporate laser with visual information [9, 23], using the visual information for feature extraction and the laser information to aid in calculation of depth.

A lot of research has been done in one vision based approach called *appearance based localisation*. The most basic appearance based localisation is done by storing many 2D images of the observed scenes as the robot moves. Features, such as corners and colour statistics, are extracted from the images to form a *signature*.

When localisation is performed, the signature from the current view is compared to previous views, stored in a database, until a match is found. The location is determined from the recorded position which the robot was in when the database image was taken [25, 73]. More advanced techniques use such methods as extracting 3D lines and projecting these onto 2D images in order to accurately detect the current pose [23].

Full 3D capture and reconstruction from images – creating a full 3D model from images – is far more difficult than just using a few lines or other features and projecting into 3D space. In the late 1970s, Marr and Poggio produced their seminal work on a "Computational Theory of Human Stereo Vision" [58, 59]. They were the first to develop a computational model of the human visual system. Their work was based on the analysis done by Julesz [32] in the previous decade on the constraints within the human visual system.

Capturing of 3D information entirely from camera is usually done using a stereoscopic camera configuration [42, 49], or occasionally a triscopic configuration [3, 51]. In stereoscopic camera configuration, two cameras are placed side by side with known separation and relative angle. Images from both cameras are captured simultaneously and processed frame-pair by frame-pair. Features, such as corners or edges, are extracted from each image and correspondence between features in the two images are computed using various means. Some vision based localisation systems use these features, projected into 3D space, for localisation. Each new location of the robot produces more 3D feature points, which can be used to recognise the location when the robot needs to perform either local or global localisation. A triscopic configuration uses three cameras to improve the accuracy of depth calculations.

Computer vision is very much an active research topic. 3D data capture from 2D image cameras is considered very difficult. For feature based approaches, extracting reliable and consistent features is difficult, and finding correspondences between these features in image pairs is also difficult. Pollefeys [69] provides a detailed description of many of the issues involved with finding correspondences between image features and in performing full 3D reconstruction.

Automatic full 3D reconstruction from sets of 2D images is the ultimate goal of many researchers within Computer Vision. Using cameras to capture the environment in such a way is a passive process, and thus does not affect the environment it observes. Ideally 3D reconstruction does does not require the environment to be initially set up with placemarks or beacons to aid it. Some systems, including that of Pollefeys *et al.* [71] are now appearing which are capable of using a video camera, without any other registration source, to compute a full 3D reconstruction.

2.3.2 3D Model Representation

Many different representations exist for 3D models. Some are based on geometric mathematical formula, while others are based on 3D extensions to "pixels" in 2D images, or use 3D point clouds, possibly with some concept of connections between points. Another method often used, stemming from CAD systems, is to represent 3D objects as composed of multiple simple 3D shapes, such as cubes, spheres, planes and cylinders.

Geometric models, such as *parametric forms* and *algebraic forms* [84] are often used for CAD systems or because a mathematical representation is required. Popular parametric forms include *Nonuniform Rational B-spline* (NURBS) [31] and *Radial Basis Functions*.

Voxels represent in 3D the same thing a pixel does in 2D. The 3D space is divided into many uniformly spaced cells, called "voxels". Each voxel may either be termfilled, or *empty*, and may optionally contain colour information. Voxels are very good for representation volumetric information for arbitrarily shaped models. Voxels are often used as the representation during 3D reconstruction from images [96, 64, 4, 76, 48]. Voxels are also widely used within medical research and within such medical processes as Magnetic Resonance Imaging (MRI). Unlike most other representation methods, voxel models are particularly easy to combine (assuming they have already been registered); simply setting a voxel to "filled" if either of the corresponding voxels in the two input models are filled. Directly storing such a data structure is too inefficient as many voxels are left empty. Consequently more efficient data structures have been developed, the most popular being the *octree* [83]. Srihari [82] provides a good survey of data structure schemes for storing voxel information.

A common representation used within Computer Graphics, and more recently, within Computer Vision, is the *polygon mesh*. Polygon meshes are popular because it is easy to design very efficient algorithms for processing them and for their flexibility in the forms they can represent. Polygon meshes are well suited to representing surfaces. Often a 3D reconstruction will be stored in a voxel format during reconstruction, and then converted to a polygon mesh as a post-processing step to smooth the final result and to store in a more efficient form.

Technically, a polygon mesh is capable of representing surfaces in any number of dimensions and scales well with the amount of data to be stored, however only three dimensional data is usually represented. Such a mesh consists of many individual flat polygons connected side by side in some arrangement. The arrangement of these polygons is referred to as the *tessellation*. Some meshes use polygons of different numbers of faces, and some use the same number of faces for all faces, the most common being the *triangular mesh* where each polygon contains exactly three edges.

Figure 2.6 shows an example polygon mesh of the head of a horse. Data structures used to store the mesh data vary. The simplest is to store two lists. The first

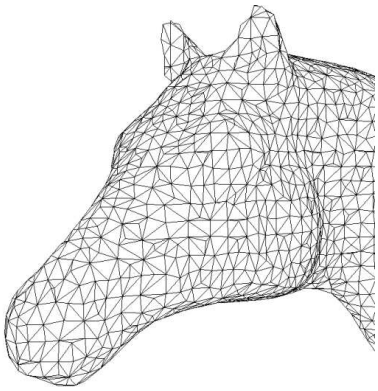


Figure 2.6: Polygon mesh
Notice that the horse head is represented entirely by triangles. This is a triangular mesh.

gives the (x, y, z) coordinates of the vertices. The second list gives the vertices of the faces, by specifying the indices into the vertex list. For the case of a triangular mesh, a list of F faces can be represented as an $F \times 3$ matrix, where the three columns represent the indices of the first, second and third vertices.

The order in which the vertices are listed for each face matters in some situations. Surface rendering packages calculate the illuminance of each face based on the angle and sign of its normal vector. A face's normal vector can be calculated by taking the cross product of the two vectors defined from the first to the second vertex and from the first to the third vertex. Consequently, assuming a face is on a plane perpendicular to the observer (the "virtual camera" in the surface renderer), if its vertices are listed in one order, the normal vector will be pointing towards the observer, but if listed in the opposite order, the normal vector will be pointing away from the observer. Surface renderers which rely on the sign of the face normal may draw those faces black. The usual standard is that vertices should be ordered counter-clockwise for faces observed from the outside of the object.

Polygon meshes are not as easy to combine as voxel based models, as faces may overlap and need reducing. Also, overlapping faces will never exactly correspond to the same coordinates, rather a small gap may exist, and thus some consensus must be made between the sets of faces being combined. Turk and Levoy [92] present a method for combining polygon meshes, using an intermediate voxel representation to help determine which faces should be combined.

Other work on other representation methods and in extending the mesh and voxel representations include, among others: Deformable Surfaces [85], a 3D extension to Deformable Contours, commonly called "snakes"; and a hybrid approach by Allegre *et al.* [5], combining implicit surfaces and polygon meshes in order to make both blending operations and local surface modifications very efficient, tasks which are only efficient for one or the other representation when treated on their own. For a review of other representation methods, including coverage of some methods for

recognising models using such representations, see Campbell and Flynn [16].

2.3.3 A Final Note on Terminology

In the remainder of this thesis, the term *model* or *surface model* is generally used as an abbreviation for "3D surface model", referring usually to a polygon mesh model, unless otherwise stated. A *scene* is the real-world environment which is captured. This scene may be captured from different viewpoints and combined to create a complete, or semi-complete, *surface model*. A *view* refers to a scene as observed from one particular viewpoint.

Chapter 3

Literature Review

The research covered by this thesis spans several areas within Computer Vision. The review of literature that follows is thus divided into several sections: 3D Capture for Localisation, covering a brief overview of some methods used for capturing 3D data for robotic localisation; Surface Recognition and Registration, giving an overview of global feature and local feature based methods for 3D model recognition and course registration; Decomposition, describing some methods for decomposing 3D models; and finally 3D Point Pattern Matching, introducing work on generic point set and point pattern matching schemes.

3.1 3D Capture for Localisation

A short discussion of some of 3D capture methods follow for the purposes of briefly covering the types of 3D data sources, but a full discussion is beyond the purpose of this thesis.

Mahon and Williams [56] present an approach for extracting 3D surface mesh data using two 2D lasers: one horizontal laser, and one vertical laser. The horizontal laser is used for the usual SLAM process and the vertical laser is used to capture 3D data as the robot moves. Localisation coordinates from the SLAM process are used to register consecutive sets of coordinates acquired from the vertical laser. An off-line post-processing mesh simplification algorithm is used to reduce the overall complexity by fitting planes to the polygon mesh faces.

Thrun *et al.* [89] do something similar to Mahon and Williams, using the same arrangement of horizontal and vertical laser. The raw laser data is represented as a polygon mesh initially and the Expectation-Maximisation (EM) algorithm is used to reduce this to planar surfaces wherever possible. They attempt both an on-line and an off-line EM process. They found the off-line process reduced the number of surface faces to between 0.3% and 0.6% of the original, and that the on-line process produced only around ten times as many faces.

Nüchter *et al.* [65] use a single 2D laser mounted on a servomotor which rotates

the laser from a position point down to a position pointing up. In this way 180° (vertical) by 120° (horizontal) is captured, made up from many 2D scans. They present a novel method for registering the 2D scans and for eliminating noise within corridor scenes by fitting large planes and eliminating outliers.

Cobzas [23] use a combination of 2D camera and laser mounted on a pivoting base. The laser is directed along the vertical plane in line with the line of sight of the camera. The camera and laser are registered so that, as the base is pivoted, the camera scans a 360° panoramic image and the laser simultaneously scans a depth map which is registered with the 2D image. This combination of image and depth map is used within an appearance-based localisation scheme. The image is used to extract vertical edges and the depth map used to determine the 3D coordinates of these edges. Localisation is performed by projecting the extracted vertical edges onto the scanned image at the previous location and using an error minimisation method to determine the most likely position. Their method was only attempted using a robot which moved only within one room and used hand chosen edges for the representation of the initial position.

Gregor and Whitaker [37] provide a detailed analysis of the noise involved when capturing laser data using a 3D laser scanner, presenting a generative model which can be used in algorithms which require a model of signal noise. An accurate final 3D surface model is generated from many 3D range scans after smoothing the scans using an anisotropic diffusion technique which preserves edges while smoothing out noise. However, correspondences between 3D scans are acquired via human interaction. Laser scans could be combined automatically if their work was extended to use algorithms to find the correspondences without human interaction.

For an excellent discussion of the current stereoscopic 3D reconstruction techniques see the work by Pollefeys *et al.* [69, 70, 71, 72].

3.2 Surface Recognition and Registration

The majority of 3D model and surface recognition research has been done for recognition of complete models, making the use of global features applicable.

Campbell and Flynn [15] perform object recognition of non-obstructed models using an appearance-based scheme they call "eigensurfaces". They render a number of different 2D views of objects by rotating around two axis, and by using principle component analysis to select a single rotation about the optical axis of each 2D view. These views are then combined in a process which creates a series of range scans representing eigenvectors of the original views in a k-dimensional eigenspace. Recognition is performed by likewise processing the input model and all models within a database in the above manner and then using a nearest neighbour approach.

Osada *et al.* [66] developed what they call "Shape Distributions", which measure the distribution of global geometric properties. Possible geometric properties

include: distances between pairs of points, areas of triangles formed from three points, volumes of tetrahedrons formed from four points, and angles between two of the possible lines formed from three points. The points are randomly sampled on a 3D polygon surface using a uniform sampling distribution. Each object is processed in such a way and recognition is performed by computing the difference between the distributions of objects and finding the matching distribution with least error.

Wahl *et al.* [94] present a novel four dimensional feature and uses a histogram over its values to produce a feature representation of the original polygon mesh. These representations are used for comparison with entries within a database of previously observed representations. Their method uses a very small subset (0.005%) of all mesh vertices, randomly sampled, however they found this to be sufficient for recognition purposes. They tested their system with up to 20% noise on the angle and position of mesh triangles, and performed some tests with surfaces containing small amounts of occlusion.

Shum *et al.* [78] represent models by deforming a regularly tessellated sphere onto the model's surface. The amount of deformation measures the curvature of the surface in a spherical polar coordinate system. The curvature is used to define a metric for comparing the difference between 3D models. Their metric can be used for comparison of complete 3D models (that is, without holes or partial occlusions).

An increasing amount of research has attempted to address the problem of partial matches. Usually this work focusses on taking data from an observed "scene" and finding the database model which the object in the scene matches, or finding the matching database model for each of multiple objects in a cluttered scene. Local feature based methods generally have a greater ability to handle partial matches. Some approaches randomly pick points on the surface and calculate a feature vector relative to each point. The typical features vectors used are distributions of distances to the remaining points, or of surface curvature at the local region.

Zuckerberger *et al.* [99] suggested decomposing into separate local surface parts and then classifying each part as being similar to a plane, cylinder or cone. A representation of the whole surface is made by constructing a graph of these labelled parts. The recognition problem was then reduced to finding the best sub-graph isomorphism between the graph of the current view and each graph in turn from the database. The sub-graph isomorphism problem has been shown to be an NP complete problem, however heuristics exist for matching small graphs, such as the publicly available "Graph Matching Toolkit", developed by Messmer [61] and used by Zuckerberger *et al.*. Their system was shown to be effective for recognition under nonrigid transformations of objects, particularly for recognising shapes which have distinct moveable parts, such as the limps on a human body.

Chua and Jarvis [21] use "point signatures" as image features, sampled at regular grid positions. A point signature at P, with surface normal $\vec{n}$, encodes the curvature of the surface surrounding P by a 1D parametric curve $d(\theta)$. A discretized version,

$d[i] = d(i \cdot \Delta\theta)$, where $\Delta\theta = 15°$, measures the signed distance of surface points, sampled at fixed angular intervals ($\Delta\theta$) around the surface at a fixed radius from P, to a plane intersecting the point P and having normal approximately equal to $\vec{n}$. By calculating the signatures for each point within a 3×3 window, error bars for each entry of $d[i]$ can be measured. Matches between two point signatures, $d_1[i]$ and $d_2[i]$, is done by determining if the curve of $d_1[i]$ is equal to the curve of $d_2[i]$ to within the limits of the error bars of $d_2[i]$. The model points are matched to each of the scene points in turn. This matching process is made efficient by using a 2D hash table. The model points are placed in the table at an index depending on their minimum and maximum $d[i]$ vales. In this way, each bin contains a list of model point signatures whose minimum and maximum $d[i]$ are similar and the table can be quickly indexed when matching. The database models are sorted by the number of matches achieved between that model's points and the scene points and the models with least number of matches are eliminated. The remaining models are verified to determine the best scene to model match. The point signatures also encode information which can be used for calculation of relative position. When two point signatures are matched, the approximate rotation and translation necessary to transform the model and the scene can be computed, assuming the point signatures were generated from sufficiently noise-free data. This approximate transformation is used to detect the correct matches from the incorrect matches in a partial correspondence search, making the registration of scene and model more efficient. If N_s scene points are extracted, N_m model points extracted, and on average H model points are matched with each scene point (due to false matches), then the worst case computational complexity is $O(N_s N_m H^3)$. This systems shows quick recognition times, however its efficiency relies in part on accurate matches between point signatures (which relies on relatively noise-free data), and on the data being sufficiently noise free for accurate approximation of rotation and translation in order that the partial correspondence search can be successful.

Johnson and Hebert [44] use "spin-images" to represent a polygon mesh model at each of many oriented points. A spin-image is a histogram over a 2D parameterisation of all other model points relative to the oriented point. The spin-images effectively measure the density of all other points under a polar parameterisation. They show that spin-images are effective for finding point correspondences, even when the models are sampled at very different resolutions, and in the presence of high clutter from other objects. Ten percent of points within each model are randomly sampled and spin-images constructed for each point. Recognition is performed by comparing every model to every scene spin-image and producing a histogram of similarity measures for each match. The upper outliers from the histogram are used to select only the best matches and these are used to sort the models in order of most likely match (depending on which models receive the most matching spin-images). For each selected model, the rigid transformation from model to scene is

computed by finding a geometrically consistent set of corresponding points. These are selected by way of a weighting function which attempts to reject point-to-point correspondences which would cause erroneous rigid transformations relative to the other correspondences. The result of that process is used as an initial starting position for a modified ICP algorithm to refine the registration. The best registered model is selected as the most likely matching model. The system is shown to match objects when only 20% of the scene model is observable. The spin-images are constructed and stored in a pre-processing stage requiring $O(MI)$ memory complexity for each model, where M is 10% of the number of points in the model representation and I is the number of pixels in each spin-image. The time complexity to generate the spin-images for each model is $O(M^2)$, because each of M spin-images are constructed from M points. The execution time required for this pre-processing step renders this method too inefficient for robotic localisation purposes unless the robot is moving very slowly and has a processing unit devoted to constructing spin-images. The computation complexity for establishing correspondences between spin-images is $O(SMI + SM\log(M))$, where S is the number of randomly sampled scene points. The complexity for finding the geometrically consistent set of correspondences is not given.

Yamany and Farag [97] represent the 3D surface model by parameterising all points other than a single interest point, P, using their distance and angle relative to P and encoding this within a 2D image. Their work is similar to the "spin-images" of Johnson and Hebert [44], except that they represent the curvature of the model relative to the feature point, whereas the histogram approach of Johnson and Hebert measures the density. The authors claim to be able to match partially obstructed objects, however only small occlusions can be handled using their method.

Methods for registration often incorporate a measure of how well the surfaces are registered. Typical approaches just find the average or root-mean-square distance between corresponding points. Often the correspondence between points is determined by the points being close to each other. An alternative was suggested by Aspert *et al.*.

Aspert *et al.* [8] developed an error measure for how well two 3D surfaces are aligned. Their alignment error measure is based on the Hausdorff distance. The Hausdorff distance measures the largest distance encountered between any two points, one in each of two surfaces. Formerly, the Hausdorff distance, $d(S_1, S_2)$, measures the maximum of $d(p_1, S_2)$, over all points, p_1, belonging to surface S_1, where $d(p_1, S_2)$ is the minimum Euclidean distance from point p_1 in surface S_1 to points, p_2, in surface S_2. Specifically, $d(S_1, S_2) = \max\{d(p_1, S_2),\ \forall p_1 \in S_1\}$, and $d(p_1, S_2) = \min\{\|p_1 - p_2\|,\ \forall p_2 \in S_2\}$. To make this measure symmetric, it is usually repeated from surface S_2 to surface S_1 and the maximum taken. Thus the final symmetric Hausdorff distance is: $d_s(S_1, S_2) = max[d(S_1, S_2), d(S_2, S_1)]$. The authors describe an efficient implementation for calculating the Hausdorff distance between

two 3D polygon surfaces. Other error measures, such as root-mean-square error, require knowing correspondences between all faces within the two surfaces. The Hausdorff distance avoids this problem by not requiring explicit correspondences, however it may be affected badly by noisy data, causing the distance minimisation of $d(p_1, S_2)$ to favour rare outlying noise points over the majority of clean data points.

3.3 Decomposition

Decomposition of 3D polyhedral shapes has been a focus of research mostly within the last thirty years. A number of researchers have focussed on decomposing surfaces into separate convex regions because convex regions are efficient for rendering.

Chazelle *et al.* [18] gives a comparison of three heuristics to aid in decomposing 3D polygonal surfaces into separate convex regions. They first show that the task of recognising regions where convexity is broken is an NP-complete problem and that heuristic approaches must be used. Three heuristics are offered and compared, based on Binary Space Partitioning (BSP), Space-Sweep, and Flooding techniques, all of which are capable of running in order $O(n \log n)$ in the number of faces. Implementation, however, for convex region decomposition can be difficult. The BSP and Flooding techniques where shown to outperform the Space-Sweep technique and are easier to implement.

Zuckerberger *et al.* [99] present two decomposition algorithms to be applied to polygon meshes. The first method, "flooding convex decomposition", decomposes the surface into multiple *patches* which are "convex hull" regions, traversing the surface while it maintains a convex hull shape, and dividing the surface when convexity is violated. The effects of noise on the surface is reduced by applying a threshold. Convexity must be violated by a certain threshold before the regions are split separate patches. Further to this, any patches which are too small are combined with the closest larger patch. The second method is based on the watershed segmentation algorithm which was first proposed for 2D image segmentation, and then extended by Mangan *et al.* [57] to 3D polygon mesh models and volumetric representations.

Other approaches have attempted to decompose the surface in a more "natural" way, attempting to automatically detect separate *parts* within the whole object, such as limps on models of people or animals. These are usually selected by detecting "deep concavities", such as the second of Zuckerberger *et al.*'s method.

Katz and Tal [46] present a hierarchical solution to the decomposition problem, performing a k-way subdivision of the polygon mesh surface at each level. At each subdivision, two points (*representatives*) are found which are deemed to be the furthest apart. Then, each face is labelled with a value from 0.0 to 1.0 representing the relative distance from the two representative points. The faces which are close to one representative form one patch. Thus giving two patches, and a region in

between which is roughly in the middle. This middle region is further divided by posing the problem as a graph partitioning problem and applying a minimum cut (maximum flow) algorithm. The algorithm proceeds iteratively, improving guesses of representative points at each iteration. The algorithm produces good results on their test data, yielding smooth edges between the patches (rather than the jagged edges which some algorithms produce), however it is computationally very expensive and potentially runs very slowly. Their algorithm, applied to models containing around 3999 faces, 67 170 faces, and 654 666 faces, took 57 seconds, 244 seconds, and 1654 seconds to execute, respectively.

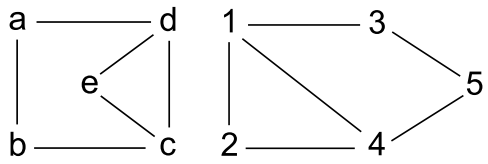
Boier-Martin [12] present a method of decomposition whereby a 3D polygon mesh is flattened and the mesh normal projected onto this flattened mesh. The image representing these normals are then segmented using a K-means clustering algorithm. This segmentation forms the basis of the decomposed regions, which are obtained by re-projecting the 2D image back onto the original mesh.

3.4 3D Point Pattern Matching

3D polygon meshes can easily be thought of as 3D point clouds without any connections, or they can be thought of as large graph structures. Attempting to match two polygon meshes – finding the correspondences between faces – using all all vertices, faces, or edges is far too time consuming, and the computation explodes exponentially with the number of points used. Thus local features are sampled in a controlled manner to produce no more than a pre-set number of sampled feature points, or decomposition can be used in a similar manner. By nature of these approaches, the complexity has merely been reduced, but the type of structure still remains: the resultant data can be thought of as 3D point clouds or as a graph structure.

After extraction of local features, the point cloud representing the positions of these features usually contains additional information describing these features. These points can be said to have *attributes*, and you have an *attributed point cloud*. Most methods which extract local features loose the connection information inherent to the original mesh structure, however a graph structure can be reproduced by various means, such as by placing edges between all pairs of adjacent points. Delaunay Triangulation is the most common approach used, for which efficient divide-and-conquer approaches exist, capable of executing in $O(n \log n)$ time. When such a graph is produced using the attributed features, the graph is called an *attributed graph*.

It is worth noting that algorithms based on attributed graphs are a popular way of recognising 3D objects in 2D images. Rigid transformations of 3D objects become nonrigid transformations when the objects are projected onto 2D. Structural information of the 2D images, such as edges, corners, and colours, can be represented as an attributed graph. Matching the 2D objects to 3D models reduces to finding



(a) (b)

Figure 3.1: Isomorphic graphs
Although graphs (a) and (b) appear different, it is possible to rearrange the vertices so that the edges are in the same arrangement for both, without breaking any connections.

sub-graph isomorphisms.

A *graph isomorphism* is a bijective mapping from the vertices in one graph, G, to the vertices in another graph, H:

$$f : V(G) \rightarrow V(H)$$

with the requirement that any two vertices, u and v, from G are adjacent (that is, there is an edge between them) if and only if $f(u)$ and $f(v)$ are adjacent in H. This effectively states that two graphs are *isomorphic* if and only if the arrangement of edges is the same, but does not require that the positions of the vertices are the same. For example, the graphs in Figure 3.1 are isomorphic. A *Sub-graph isomorphism* is one where either one or both graphs contain vertices not present within the other graph.

Finding sub-graph isomorphisms is considered very hard and to be computationally expensive. Most methods resort to random sampling approaches or a heuristically driven search. Nevertheless, research does exist applying the sub-graph isomorphism matching approach to 3D surface recognition, such as one of Zuckerberger *et al.*'s methods [99].

Messmer [61] noted that it is not yet known whether finding graph isomorphisms is P or NP, but finding sub-graph isomorphisms is well known to be NP-complete. However, the author also noted that many algorithms exist which are capable of finding solutions in approximately polynomial time using heuristics. Messmer developed what he called the "Graph Matching Toolkit", a set of tools for performing graph and sub-graph isomorphisms on attributed graphs. Source code is available at `http://iamwww.unibe.ch/~fkiwww/projects/GraphMatch.html`.

Gold and Rangarajan [36] reduced the sub-graph isomorphism problem to an assignment problem with two-way constraints. The assignment problem is that of assigning a correspondence between vertices in one graph to vertices in the other graph. This has two sets of constraints on it because the correspondence must be one-to-one and that this one-to-one mapping must be the same in both directions

(from vertices in one graph to vertices in the other and back again). The assignment of correspondences between edges and vertices in the two graphs can be represented as a 2D matrix which has 1s to represent assignment of a correspondence. The authors showed that the two-way constraint assignment problem can be solved using an iterative approach based on deterministic annealing. Deterministic annealing enables the algorithm to attempt to avoid local minima solutions to the assignment problem by initially approximating the solution with a low degree of accuracy followed by incrementally improving this approximation until a consistent solution is found. At each step, the assignment of correspondences is recalculated and parameters adjusted to minimise an error function. The assignment correspondences are represented in a "fuzzy" way be storing a matrix containing the weighted assignment from each vertex in one graph to all possible vertices in the other graph. Their algorithm executes in time $O(lm)$, where l is the number of vertices in one graph and m the number in the other graph.

Gold *et al.* [35] then reapplied Gold and Rangarajan's sub-graph isomorphism solution to that of solving the simultaneous correspondence and registration problem of registering two unlabelled 3D point clouds, in the presence of noise. Assuming a rigid transformation, if either of the transformation or correspondence between points is known, then registering two 3D point clouds is relatively easy. Many fast and accurate methods exist for calculating rigid transformation from corresponding points [7, 29, 95]. However, when neither the correspondences nor the transformation are known, the problem is considerably harder. Gold *et al.* showed how this can be solved using the same principle of describing the problem as an assignment problem, using deterministic annealing and "fuzzy" assignment in order to find the solution iteratively. Their algorithm has $O(lm)$ time complexity, where l and m are the number of points in the two data sets.

B. van Wyk and M. van Wyk [93] also described the sub-graph isomorphism problem as one of an assignment problem with two-way constraints, but solved the problem without the use of annealing methods or penalty terms. Rather, they used what they called "Projections Onto Convex Sets" (POCS). They note that existing research has found that the set of possible row constrained assignment matrices is closed and convex, and likewise the set of column constrained assignment matrices is closed and convex (where the metric space is defined by vectorization of the matrices). The union of these two sets is also closed and convex and represents the set of all possible doubly-stochastic matrices. Thus anything in the union of these two sets satisfies the two-way constraints. Using these results, and other results from existing literature, the authors are able to calculate an approximately optimal solution with worst case time complexity $O(n^4)$ if both graphs contain n vertices.

Lowe *et al.* [51] use the "Random Sample Consensus" (RANSAC) algorithm [33] to perform matching on two sets of 3D feature points extracted for the purposes of 2D global localisation. One set of feature points is from a "current view", and

the other is taken from a single view within a database. A greedy estimate of the correspondences from current view feature points to the database view feature points is first produced. These tentative matches are produced by finding, for each feature from the current view, the best matching feature from the database view. RANSAC is employed by randomly selecting two feature points from the current view and using the tentative matches to select two corresponding feature points from the database view. These two pairs of points are used to calculate an estimate of the 2D rotation and translation representing the relative orientation of the two point sets. The number of feature points supporting this orientation, within some error tolerance, is computed. This process is repeated a number of times and the best outcome is selected as the final orientation. The initial tentative matching between features requires $O(lm)$ time complexity, for l and m feature points in the two data sets. The time complexity of the RANSAC algorithm is hard to calculate because it requires an arbitrary number of iterations. The number of iterations required is dependent on the level of noise within the data set, the percentage of correct correspondences within the tentative matches, and the desired probability of finding the correct result. In some situations it is possible to estimate the number of iterations required to achieve a certain probability of success.

Cox and de Jager [24] provide a study of some other methods within three broad types of point pattern matching techniques, namely clustering based approaches, interpoint distance based approaches, and relaxation methods. They found the majority of methods they examined to be incapable of handling partial matches. Further to this, the authors describe a new algorithm which they claim is capable of handling partial matches and which executes with time complexity $O(n^3)$, for two data sets of approximately n elements.

Chapter 4

Existing Algorithms

A detailed description of some existing algorithms for various tasks will now be given in order that the reader may understand their usage within the larger context of surface recognition. Some of these algorithms have been modified slightly when implemented by the author of this thesis. Such modifications are described within chapter 5.

The order of the algorithms described here follows their usage within the implementation of the recognition system. Firstly, 3D surface decomposition by the "Watershed algorithm" is described. Secondly, the concept and calculation of "Shape Distributions" is described as a suitable *patch descriptor*. Thirdly, the generic RANSAC algorithm is described, which provides a way of finding solutions to regression problems and model fitting problems under the influence of great noise. Fourthly, a simple and accurate method is presented using Singular Value Decomposition for calculating the rigid transformation (rotation and translation) giving the least squares error when attempting to transform one set of points onto another set with known correspondences. Lastly, an iterative algorithm is presented which attempts to find the rigid transformation with least squares error between two sets of points with unknown correspondence.

Within the algorithms presented here, the following notation is used:

- Variables of form $\{p_i\}$ or $\{P_i\}$ indicate that p or P is a set and that individual entries within it are specifically accessed using variable i. Those individual entries are referred to using the notation p_i or P_i.
- Other sets will be specifically indicated and will use the usual form of p or P.
- Variables of form $\{m_{jk}\}$ or $\{M_{jk}\}$ indicate that m or M is a matrix and that individual entries within it are specifically accessed using variables j and k. Those individual entries are referred to using the notation m_{jk} or M_{jk}.
- Matrices are represented in bold form when the indices are not explicitly stated: $\mathbf{m}$ or $\mathbf{M}$.

4.1 Decomposition with Watershed Algorithm

The Watershed algorithm was initially developed by Serra [77] as a method for segmenting 2D images. Mangan *et al.* [57] later extended this popular algorithm from the 2D image domain to 3D triangular meshes.

In its most basic 2D form, the watershed algorithm uses a *height function*, $h(x, y)$, on the original image data. Typical height functions are the grey level intensity values or the gradient of intensity values. The watershed algorithm then uses this height function to assign each image point to a *catchment basin*. The catchment basins, after processing, are local regions of pixels containing a single point or small region of minimum $h(x, y)$ value, and with all other $h(x, y)$ within the catchment basin increasing monotonically radially outwards from the minimum point or region. The catchment basin can be thought of as an irregularly shaped bowl or basin, capable of storing water – thus the term "watershed". Any drops of water placed at points other than the bottom will descend towards the bottom. The image is then represented by a number of catchment basins, all having other catchment basins adjacent to them, and leaving no gaps in between. The edges of catchment basins form the edges of segmentation.

The typical problem encountered with the watershed algorithm is its sensitivity to noise, causing many small catchment basins to be produced. The solution usually employed is to use the *watershed depth* and merge those catchment basins having a depth below some threshold with one of its immediately neighbouring catchment basins. The watershed depth, also called the *watershed height*, is defined as the distance between a catchment basin's minimum point and the lowest height of its edge. This is most easily understand to represent the maximum depth of water capable of being stored within the catchment basin without spilling into any adjacent catchment basins. The shallow catchment basins are merged with those basins into which they would first flow if they overflowed.

In order to extend the watershed algorithm to 3D surfaces, Mangan *et al.* define the height function over a measure of the curvature of the immediate surface and present a number of methods for computing such a height. Zuckerberger *et al.* [99] show that these methods do not produce consistent decompositions when different tesselations are used (the selection of edges between vertices), and present a much simpler method which is not affected by tessellation to the same extent. Zuckerberger *et al.* calculate the height function using the angle between face normals.

4.1.1 Description of Algorithm

The 3D surface watershed algorithm, as described by Mangan *et al.* and including the height function developed by Zuckerberger *et al.*, is presented here. The process is divided into three parts. Firstly, a height value is assigned to each surface face. Secondly, the faces are labelled and assigned to catchment basins. Thirdly, a post-

processing step is used to reduce the effects of noise by merging shallow catchment basins.

Consider that the surface mesh is represented by a list of vertices, faces, and edges between vertices. The height function is initially calculated per edge and is given the value $h = 1 - \cos(\alpha)$, where α is the angle between the normals of the two faces attached to the edge and calculated as Equation (4.1), where $\vec{n}_1$ and $\vec{n}_2$ are the normal vectors for each of the two faces. Each face is then assigned the minimum height value of the heights assigned to its edges. In this way it is possible to calculate the height based on the angle of curvature and assign height values to the faces. Measuring the angle *across* the edges is easier to calculate, but decomposition itself is easier to perform on the faces rather than the edges.

$$\alpha = \cos^{-1}\left(\frac{\vec{n}_1 \cdot \vec{n}_2}{\|\vec{n}_1\| \cdot \|\vec{n}_2\|}\right) \tag{4.1}$$

Note that care needs to be taken to ensure that the polygon mesh is represented correctly within the data structure used. As discussed within the Background chapter, the order in which vertices are listed for each face affects the sign of the face normal vectors. Equation (4.1) produces incorrect results if vertices of adjacent faces are listed in different orders.

Face labelling is performed as follows. First, all local minima are found and labelled. Local minima are found by examining the height of adjacent faces and selecting those faces which have no adjacent faces with lesser height. Then, all "flat" regions are found: regions of faces having equal height. If all faces adjacent to a flat region have height greater than the height of the faces within the flat region, then that region is labelled as a "minimum" region and treated the same as for single face minima. All other flat regions are labelled as "plateau" because some or all adjacent faces have heights less than those faces within the plateau region. Then all plateau are labelled with the index to a single adjacent face with the least height. Finally, all faces are iterated through once, collecting into sets of faces forming the catchment basins. Faces within the catchment basins which result form the final regions and are labelled as complete *patches*, represented by a list of face indices, and stored as an entry within a list of patches, $\{P_i\}$.

The creation of patches proceeds like drips of water dropped onto each face. For each face already part of a complete patch, do nothing. For each face found to be unlabelled, start from it and move to each steepest adjacent face, accumulating all the face indices within a temporary set, C, and stopping when encountering a minimum or a face which is already labelled as part of a patch. At each plateau encountered, add all those faces to C and move to the next steepest descending face from the plateau. When a minimum is encountered, add that face to C and add C as a new patch, at index i, in $\{P_i\}$; then assign each of the faces within C a label indicating that it is now part of patch i. Faces which were minima will no longer

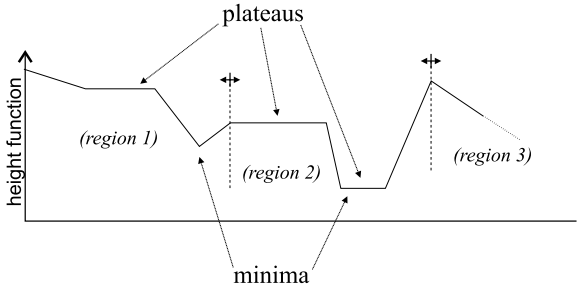


Figure 4.1: Collecting regions

Non-minima flat regions are labelled as "plateau" and fall into the region with their steepest descending edge. Minima are either single points or flat regions without any descending edges. All other faces combine with their steepest descending adjacent face.

be labelled as such. If a face is encountered which is already assigned to a patch, say patch l, then add C to patch l and assign labels to all faces within C indicating that they are now part of patch l.

If labelling of minima and plateau is done correctly, then no faces will remain after the previously stated process. Note that it is possible to label a face on the edge of the surface as a local minimum.

Block A of Algorithm 1 presents the process of grouping faces in pseudo-code form, which labels local minimum faces, minimum flat regions, and plateaus as illustrated in Figure 4.1.

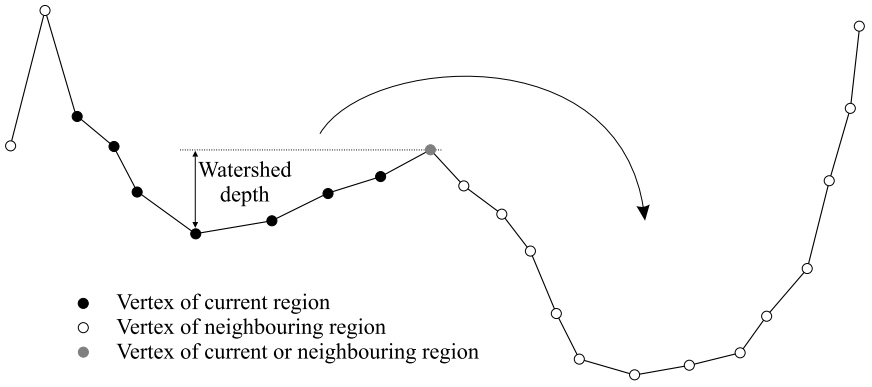


Figure 4.2: Merging shallow regions

Watershed depth is based on the lowest point and the lowest of the edge points. The lowest edge point may be within the current region or the adjacent region. Shallow watersheds are merged with their deepest counterparts.

The reliance on catchment basins containing only a single local minimum causes many small patches, many of which are formed from only small changes in curvature. Figure 4.2 illustrates how the *watershed depth* is calculated from the minimum face within the catchment basin and from the height of the lowest edge face, which may be on the edge of the current catchment basin or on the edge of the neighbouring catchment basin.

Algorithm 1 Watershed Decomposition

Input: face heights, face labels, local minima and plateau regions
Output: $\{P_i\}$
where: $\{P_i\}$ is a list of patches, each patch being represented by a list of face indices.

Begin A: For all j in $(1, 2, \ldots, F)$, where F is the number of faces
$C \leftarrow \emptyset$
$k \leftarrow j$
Begin B: Loop while face k has not been assigned a patch index
if face k currently unlabelled **then**
$C \leftarrow C \cup \{k\}$
$k \leftarrow$ index of face adjacent to current face k with least height
else if face k labelled as part of plateau region **then**
Add all faces within that plateau to C
$k \leftarrow$ index of face adjacent to plateau with least height
else if face k labelled as minimum **then**
$C \leftarrow C \cup \{k\}$
Add C as a new patch in $\{P_i\}$
Assign the patch index as new labels to each of faces within C
else if face k labelled as part of patch P_i AND C non-empty **then**
$P_i \leftarrow P_i \cup C$
Assign existing patch index, i, as new labels to each of faces within C
end if
End B
End A

Begin C: For all P_i in $\{P_i\}$
$d \leftarrow$ CalculateWatershedDepth(P_i)
if d less than *minWatershedDepth* **then**
$j \leftarrow$ select neighbouring region, P_j, which is adjacent to the lowest boundary point (point A in Figure 4.2)
Merge P_i and P_j, and update list of patches:
$P_j \leftarrow P_i \cup P_j$
$P_i \leftarrow \emptyset$
end if
End C

The region merging process follows that of Block C within Algorithm 1. The set of patches is iterated over, selecting each P_i patch and merging with one of its neighbours, P_j, if the watershed depth of P_i is below the specified threshold, *minWatershedThreshold*. The face indices within P_i are added to P_j when merging (it is guaranteed that $P_i \cap P_j = \emptyset$ before this operation), and P_i is set to the empty set. If j is less than i then P_j will already have been examined, so its watershed depth will satisfy the threshold condition and it will still satisfy the threshold condition once merged with P_i. If j is greater than i, then P_j may not satisfy the threshold condition but it will be examined when its turn comes and further merged if necessary.

With careful effort, the entire decomposition and merging process can be implemented very efficiently. Although Block A of Algorithm 1 contains a nested loop, each face is only considered once since the outer loop skips faces already processed. With the use of additional data structures the union operations can be performed without the need to perform search operations on the entries within set C or any of the sets within $\{P_i\}$. Furthemore, the merging process of Block C needs only to examine the faces within the edges of patches when calculating watershed depth and finding the adjacent patch; and these patch-edge face lists can be updated very efficiently during merging of two patches. The time complexity and memory complexity are therefore both $O(n)$ in the number of faces.

4.2 Shape Distributions

Shape Distributions were devised by Osada *et al.* [66] as a way of creating a *shape signature* for arbitrary 3D polygonal models. Shape distributions are frequency distributions over some *shape function* measuring global geometric properties (such as distances between points). By reducing the description of a 3D object to a 1D vector of frequencies, similarity calculations and recognition of 3D objects is made very efficient. Osada *et al.* found their method to be effective for discriminating between classes when tested on data containing closed 3D models of such things as land vehicles, aeroplanes, people, animals, and other objects.

Shape distributions are a desirable approach because they are only affected by noise proportionately to the percentage of noise present and are generic enough to handle any 3D surface, unlike some methods which, for example, can't handle 3D models with holes. Furthermore, shape distributions are more efficient to compute than many other methods.

The authors compare five distance functions, measuring such things as: the angle between lines subtended by three points, distance from the centre of mass, distance between two points, area of the triangle formed from three points, volume of the tetrahedron formed from four points. All distance functions are used by randomly selecting points and calculating the distances, angles, areas or volumes. Their most effective measure is the "D2" distance function, which measures the Euclidean distance between two points.

The description of their algorithm, which follows, concentrates on the use of the D2 distance function. The interested reader can replace the D2 distance function with any other, such as those described in the original authors' paper.

Figure 4.3 shows a few distributions for some common shapes, illustrating how the distributions can vary for different shapes. Figure 4.4 illustrates the effects of noise on the distributions produced by showing how the distribution degenerates quickly with noise added to an initially perfect square.

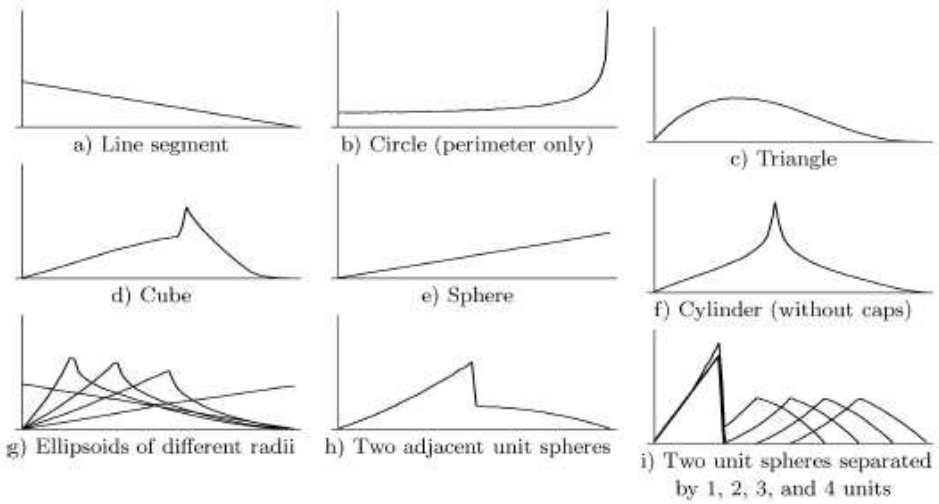


Figure 4.3: Shape distributions for some typical shapes
For each distribution, the horizontal axis follows increasing distance while the vertical axis is the probability of finding two points at the given distance. (source: Osada *et al.*, 2002 [66])

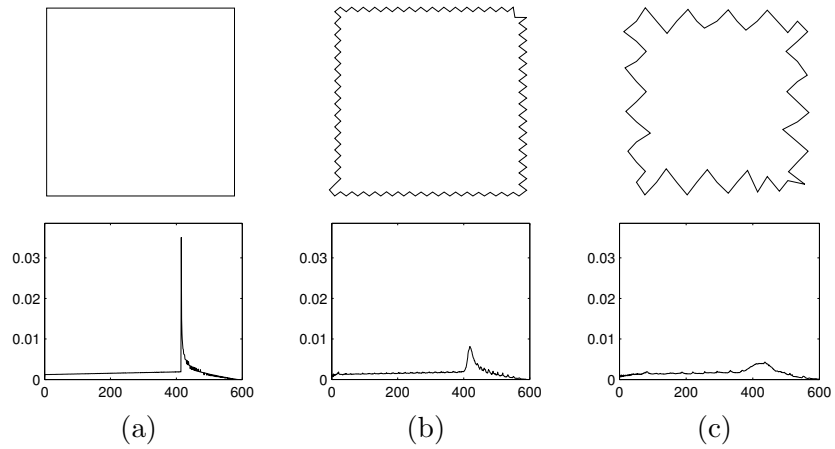


Figure 4.4: Effects of noise on shape distributions
Shows the shape distributions (bottom row) produced from the perimeter of each object (top row). The horizontal axis shows distance, in pixels. The vertical axis shows the probability distribution of distances.

4.2.1 Computing Shape Distributions

Osada *et al.* employ a stochastic method for calculating the distribution curve over the chosen distance function. N samples of the distance function applied to the 3D surface model are taken and a histogram is formed by counting the number of samples which fall into each of B fixed sized bins. The histogram is then used to construct a piecewise linear function representing the distribution with $V(\leq B)$ equally spaced vertices (taking the value in each or several bins for the value at each vertex). The piecewise linear function, with V vertices, is stored as the final representation for the *Shape Distribution* for the given 3D model.

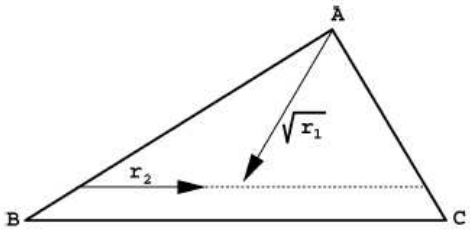


Figure 4.5: Selecting a point in a triangle with uniform probability
(source: Osada *et al.*, 2002 [66])

Care must be taken to ensure that the number of samples, N, is sufficient to sample the distribution accurately enough while avoiding excessive computation times, and to choose B and V sufficiently large that the measured distribution is represented accurately enough for the application, while avoiding unnecessary memory usage and computation time.

The selection of points within the surface model is performed so that any point has equal probability of selection, regardless of the area of the polygon face to which it belongs. In order to achieve this, faces are chosen randomly but using a selection weight proportional to the area of the face, and points within chosen faces are generated rather than simply selecting the face vertices or mid-points.

First, if any non-triangular polygons exist, they are split into separate triangles. Then, for each triangle, its area is computed and a reference to it is stored in an array along with the cumulative area of triangles visited so far. Triangles are then selected with probability proportional to their area by generating a random number between 0 and the total cumulative area and by performing a binary search on the array of cumulative areas to select the face having the assigned cumulative area closest to the random number. Points within the selected faces are chosen by generating two random numbers, r_1, $r_2 \in [0, 1]$, and evaluating the following equation, where $\vec{A}$, $\vec{B}$ and $\vec{C}$ are the vertices of the chosen triangle and $\vec{P}$ is the generated point:

$$\vec{P} = (1 - \sqrt{r_1})\vec{A} + \sqrt{r_1}(1 - r_2)\vec{B} + r_2\sqrt{r_1}\vec{C}$$

Here, $\sqrt{r_1}$ determines the percentage from vertex A to the opposing edge, while r_2 represents the percentage along that edge. The points are chosen uniformly with respect to area by using the square root of r_1. Figure 4.5 illustrates this uniform distribution selection within a triangle.

Algorithm 2 summarises the process of computing the shape distribution for a single 3D surface model after all non-triangular polygons have been split into triangular polygons. DistanceFunction is some distance function, such as the "D2" function described earlier which computes the Euclidean distance between two points. Osada *et al.* performed their experiments using a sample size of $N = 1024^2$, a histogram size of $B = 1024$ bins, and then simplified and represented the histogram

Algorithm 2 Computing Shape Distribution

Input: $\{mesh_i\}$, a set of F faces, where each face is represented by three vertices in 3D space.

Output: s, the shape distribution, represented as the vertices of a piecewise linear function forming the PDF.

Construct array of cumulative face areas:
$cumFaceAreas_i \leftarrow \sum_{j=1}^{i} \textsc{FaceArea}(mesh_j), \qquad \forall i \in 1 \ldots F$
$totArea \leftarrow \sum_{j=1}^{N} \textsc{FaceArea}(mesh_j)$

Compute histogram, $\{h_b\} = \{h_1, h_2, \ldots, h_B\}$, of distances:
$h_b = 0, \qquad \forall b = 1 \ldots B$
$d_{\max} \leftarrow$ maximum distance between faces in *mesh*.
Begin A: For $k = 1 \ldots N$
  Calculate enough random points for distance function (two if using "D2"):
  $a_1 \leftarrow$ random number $\in [0, totArea]$
  $a_2 \leftarrow$ random number $\in [0, totArea]$
  $i_1 \leftarrow \textsc{BinarySearch}(cumFaceAreas,\ a_1)$
  $i_2 \leftarrow \textsc{BinarySearch}(cumFaceAreas,\ a_2)$
  $\vec{A}_1, \vec{B}_1, \vec{C}_1 \leftarrow$ coordinates of vertices of face i_1
  $\vec{A}_2, \vec{B}_2, \vec{C}_2 \leftarrow$ coordinates of vertices of face i_2
  $\vec{p_1} \leftarrow (1 - \sqrt{r_1})\vec{A}_1 + \sqrt{r_1}(1 - r_2)\vec{B}_1 + r_2\sqrt{r_1}\vec{C}_1, \qquad (r1,\ r2 \text{ random} \in [0, 1])$
  $\vec{p_2} \leftarrow (1 - \sqrt{r_1})\vec{A}_2 + \sqrt{r_1}(1 - r_2)\vec{B}_2 + r_2\sqrt{r_1}\vec{C}_2, \qquad (r1,\ r2 \text{ random} \in [0, 1])$

  Calculate distance value:
  $d \leftarrow \textsc{DistanceFunction}(p_1, p_2)$

  Bin into histogram:
  $b \leftarrow \left\lfloor \frac{d}{d_{\max}}(B - 1) \right\rfloor + 1$
  $h_b = h_b + 1$
End A

Compute piecewise linear function approximation of $\{h_b\}$:
$s \leftarrow$ convert $\{h_b\}$ into piecewise linear function using V vertices.

using $V = 64$ vertices.

4.2.2 Comparing Shape Distributions

The shape distributions are used to select the best match between one surface model and a list of surface models. The authors investigated several methods for comparing shape distributions, based on existing work for comparing probability distributions. These include the Minkowski L_N norms of the probability distribution functions (PDF) and of the cumulative distribution functions (CDF), the chi-squared statistic, χ^2, and the Bhattacharyya distance [11]. They found the most effective to be the L_1 norm (often called the "Manhatten distance") applied to the PDF – the shape

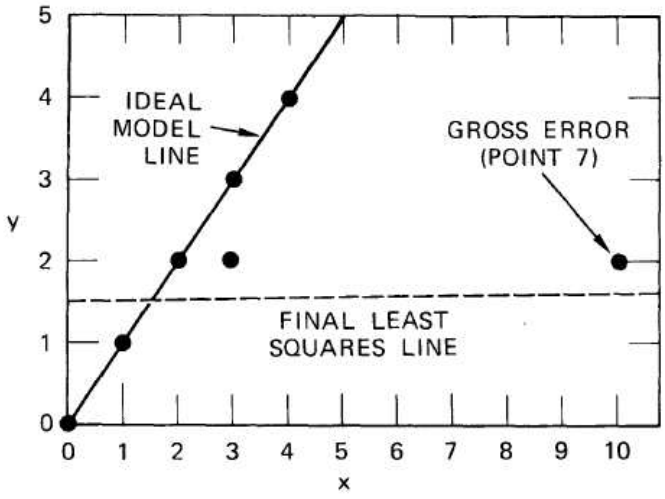
distributions calculated as aforesaid. The Minkowski L_1 norm of two PDF, f and g, is shown in Equation (4.2) and can be calculated efficiently because the shape distributions are represented as piecewise linear functions.

$$D(f,g) = \int |f - g| \tag{4.2}$$

The authors also presented and compared three methods for normalising the distributions so that comparisons of 3D models can be made scale invariant. The description of scale normalisation methods is omitted from this thesis as scale invariance is not a feature required for any use of shape distributions within this investigation.

4.3 Random Sample Consensus (RANSAC)

Random Sample Consensus, or RANSAC, was introduced by the seminal work of Fischler and Bolles [33] as a solution to the problem of outlying data points when fitting models to experimental data. The most common methods handle outliers by relying on sufficient correct data points to offset any bias caused by outliers, or assume (often incorrectly) that any noise is zero mean centred – that is, when averaged, the overall effect of the noise component is negligible.



POINT	x	y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

Figure 4.6: Linear regression problem
Applying iterative least squares regression and removal of outliers, starting with all data points, results in the dashed line as the fitted model. The solid line shows the desired model fit.
(Source: Fischler *et al.*, 1981 [33])

Consider the linear regression problem of Figure 4.6. The data set contains seven points, one of which is a large outlier. If it is known *a priori* that no valid data points are greater than 0.8 units from the ideal model line (the thick line), then a common approach is to perform a least squares fit on all data, followed by removing data

Algorithm 3 RANSAC

Input: P, a set of data points; n, the number of points required to instantiate the model.

Require: $|P| \geq n$ (where $|P|$ is the cardinality of set P)

Output: $M1^*$, the instantiated model, or *failure*.

Begin A: Repeat while number of iterations $\leq t_{\max}$
  $S1 \leftarrow$ randomly select subset of n points from P
  $M1 \leftarrow$ instantiate model from $S1$
  $S1^* \leftarrow$ select points from P which are within error tolerance, δ, of $M1$
  ($S1^*$ is called the *consensus set*)

  if $|S1^*| \geq$ minimum consensus size threshold, κ **then**
    $M1^* \leftarrow$ compute new model from all points in $S1^*$ (possibly using a least squares method)
    terminate in *success*
  end if
End A
if $|S1^*|$ was never $\geq \kappa$ **then**
  terminate in *failure*, or return the model with the greatest $|S1^*|$
end if

points which are outlying by more than 0.8 units and recalculating the least squares fit on the remaining data points. This approach is repeated until convergence is encountered. The dashed line of Figure 4.6 indicates the resultant model fit which is severely affected by the seventh point.

Rather than selecting all points and eliminating outliers after each iteration, the RANSAC approach begins with the minimum possible number of data points and then adds non-outliers. The initial points are randomly selected and this process is repeated many times until it is likely that a valid solution has been found.

Any given model requires a certain minimum number of points in order to *instantiate* its parameters. For example, a line requires two points, and a circle requires three. This defines the minimum number of points required to compute the model at each iteration of RANSAC.

The problem of Figure 4.6 can be solved using RANSAC by the following. Two points are selected and the model parameters for a line calculated. All points which are within 0.8 units of this line are accepted as non-outliers and the regression line is recalculated using all accepted non-outliers. If the number of non-outlying points is no less than some predetermined number, then accept this model. If, however, the number of non-outlying points is less than the threshold then this fit is rejected and the process repeats. If, after a predetermined number of iterations, no solution has been found, terminate in failure or return the best solution found so far.

The generic RANSAC algorithm is presented in Algorithm 3. It uses three user defined parameters, $t_{\max}$, δ, and κ, to control the operation. $t_{\max}$ sets the maximum

allowed number of iterations: this can be used to avoid infinite loops when a valid model fit does not exist with the given data. δ defines the level of acceptable noise: any data points within δ distance from the model fit will be accepted as part of the consensus set, $S1^*$. κ sets the minimum required size for the consensus set: if this requirement is satisfied, then the method can exit in success, returning the calculated model parameters, $M1^*$.

In some cases it is possible to calculate *a priori* how many iterations are required in order to achieve the desired accuracy (δ) and number of data points (κ) with a desired degree of certainty. This helps to set a value for $t_{\max}$. Fischler and Bollin's paper describes a method for calculating the required number of iterations.

The algorithm presented can be modified, for example, to incorporate weighted random selection methods instead of a uniform selection, or to continue executing a minimum of number of iterations, even if a valid $S1^*$ had been found, and to pick the best model from those valid models.

4.4 Computing Rigid Transformation with SVD

The method that follows was originally developed by Arun *et al.* [7]. This is a very popular method for calculating rigid transformations because it is stable under large amounts of noise, is very efficient to compute, and is often more accurate than other methods, as experiments performed by Eggert *et al.* [29] show.

Consider two sets of N corresponded points, $\{\vec{X}_i\}$ and $\{\vec{Y}_i\}$, where each $\vec{X}_i$ and $\vec{Y}_i$ is a column vector representing point coordinates in 3D space, and points $\vec{X}_i$ and $\vec{Y}_i$ are uniquely corresponded for all $i = 1 \dots N$. We wish to find the orthogonal rotation matrix, $\mathbf{R}$, and the translation vector, $\vec{T}$ which transforms the points $\{\vec{Y}_i\}$ onto the points $\{\vec{X}_i\}$. Both point sets may contain some amount of noise and so we wish to find $\mathbf{R}$ and $\vec{T}$ which transform the points with the least error.

Once transformed, point sets $\{\vec{X}_i\}$ and $\{\vec{Y}_i\}$ are related by the following equation:

$$\vec{X}_i = \mathbf{R}\vec{Y}_i + \vec{T} + \vec{V}_i \qquad \forall i = 1 \dots N$$

where $\vec{V}_i$ is the amount of noise between the data points $\vec{X}_i$ and $\vec{Y}_i$. Assuming both data sets contain gaussian distributed noise, the optimal values for $\mathbf{R}$ and $\vec{T}$ can be found by minimising the following *least squares* error function:

$$E(\mathbf{R}, \vec{T}) = \sum_{i=1}^{N} \|\vec{X}_i - (\mathbf{R}\vec{Y}_i + \vec{T})\|^2 \tag{4.3}$$

Calculation of $\vec{T}$ can be postponed until a later stage by translating the centroid of both data sets to the origin, reducing the number of unknown variables at the

same time. Let:

$$\begin{array}{ll} \bar{X} = \frac{1}{N}\sum_{i=1}^{N}\vec{X}_i & \vec{x}_i = \vec{X}_i - \bar{X} \\ \bar{Y} = \frac{1}{N}\sum_{i=1}^{N}\vec{Y}_i & \vec{y}_i = \vec{Y}_i - \bar{Y} \end{array} \tag{4.4}$$

then Equation (4.3) can be rewritten and reduced as follows:

$$\begin{array}{rl} E(\mathbf{R}) & = \sum_{i=1}^{N}\|\vec{x}_i - \mathbf{R}\vec{y}_i\|^2 \\ & = \sum_{i=1}^{N}(\vec{x}_i^T\vec{x}_i + \vec{y}_i^T\vec{y}_i - 2\vec{x}_i^T\mathbf{R}\vec{y}_i) \\ & = \sum_{i=1}^{N}\vec{x}_i^T\vec{x}_i + \sum_{i=1}^{N}\vec{y}_i^T\vec{y}_i - 2\sum_{i=1}^{N}\vec{x}_i^T\mathbf{R}\vec{y}_i \end{array} \tag{4.5}$$

The first two terms are constant and so Equation (4.5) is minimised when the last term is maximised. If each $\vec{x}_i$ is represented as $\vec{x}_i = (x_{i1}, x_{i2}, x_{i3})$, and likewise $\vec{y}_i = (y_{i1}, y_{i2}, y_{i3})$, the last term can be expanded as follows:

$$\begin{array}{rcl} \sum_{i=1}^{N}\vec{x}_i^T\mathbf{R}\vec{y}_i & = & \sum_{i=1}^{N}[x_{i1}\ x_{i2}\ x_{i3}]\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}\begin{bmatrix} y_{i1} \\ y_{i2} \\ y_{i3} \end{bmatrix} \\ & = & \sum_{i=1}^{N}[x_{i1}\ x_{i2}\ x_{i3}]\begin{bmatrix} r_{11}y_{i1} + r_{12}y_{i2} + r_{13}y_{i3} \\ r_{21}y_{i1} + r_{22}y_{i2} + r_{23}y_{i3} \\ r_{31}y_{i1} + r_{32}y_{i2} + r_{33}y_{i3} \end{bmatrix} \\ & = & \sum_{i=1}^{N}[r_{11}x_{i1}y_{i1} + r_{12}x_{i1}y_{i2} + r_{13}x_{i1}y_{i3}) + \\ & & r_{21}x_{i2}y_{i1} + r_{22}x_{i2}y_{i2} + r_{23}x_{i2}y_{i3}) + \\ & & r_{31}x_{i3}y_{i1} + r_{32}x_{i3}y_{i2} + r_{33}x_{i3}y_{i3})] \\ & = & \sum_{i=1}^{N}\mathit{Trace}\left(\mathbf{R}\vec{y}_i\vec{x}_i^T\right) \\ & = & \mathit{Trace}\left(\mathbf{R}\sum_{i=1}^{N}\vec{y}_i\vec{x}_i^T\right) \end{array}$$

And so, maximising the last term of Equation (4.5) is equivalent to maximising $\mathit{Trace}(\mathbf{RC})$, where $\mathbf{C}$ is the *correlation matrix*, or *cross-dispersion matrix*, given as:

$$\mathbf{C} = \sum_{i=1}^{N}\vec{y}_i\vec{x}_i^T \tag{4.6}$$

The singular value decomposition of $\mathbf{C}$ is:

$$\mathbf{C} = \mathbf{udv}^T \tag{4.7}$$

where $\mathbf{u}$ and $\mathbf{v}^T$ are orthogonal matrices, and $\mathbf{d}$ is a diagonal matrix containing the *singular values* of $\mathbf{C}$. Substituting in the decomposition of $\mathbf{C}$, $\mathit{Trace}(\mathbf{RC})$ now becomes:

$$\begin{array}{rl} \mathit{Trace}(\mathbf{RC}) & = \mathit{Trace}(\mathbf{Rudv}^T) \\ & = \mathit{Trace}(\{\mathbf{v}^T\mathbf{Ru}\}\mathbf{d}) \\ & = \mathit{Trace}(\mathbf{wd}) \qquad (\mathit{where}\ \mathbf{w} = \mathbf{v}^T\mathbf{Ru}) \end{array}$$

Since $\mathbf{d}$ is a diagonal matrix, only the diagonal elements of $\mathbf{w}$ contribute to the

value of *Trace*(**wd**). Furthermore, all matrices composing **w** are orthogonal, and thus **w** must be orthogonal itself ($\mathbf{w}\mathbf{w}^T = \mathbf{I}$, the identity matrix). Matrix **d** is constant so *Trace*(**wd**) is maximised when **w** is the identity matrix:

$$\mathbf{v}^T\mathbf{R}\mathbf{u} = \mathbf{I}$$

Rearranging the above leads to the least squares solution for **R**:

$$\begin{aligned} \mathbf{v}(\mathbf{v}^T\mathbf{R}\mathbf{u})\mathbf{u}^T &= \mathbf{v}\mathbf{I}\mathbf{u}^T \\ (\mathbf{v}\mathbf{v}^T)\mathbf{R}(\mathbf{u}\mathbf{u}^T) &= \mathbf{v}\mathbf{u}^T \\ \mathbf{R} &= \mathbf{v}\mathbf{u}^T \end{aligned} \tag{4.8}$$

The optimal translation vector, $\vec{T}$, shifts the rotated centroid of set $\{\vec{Y}_i\}$ onto the centroid of set $\{\vec{X}_i\}$ by:

$$\vec{T} = \bar{X} - \mathbf{R}\bar{Y} \tag{4.9}$$

One final point remains. Usually the determinant of **R** is $+1$, in which case nothing more needs to be done. However, it is possible for the determinant of **R** to be -1. In this case **R** represents a reflection as well as a rotation. This can be corrected by modifying the result of Equation (4.8) with the following:

$$\mathbf{R} = \mathbf{v}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(\mathbf{u}\mathbf{v}^T) \end{pmatrix}\mathbf{u}^T \tag{4.10}$$

A summary of the salient calculations is presented as Algorithm 4.

4.5 Gold *et al.*'s Method

Gold *et al.*'s method, presented in [35], solves a similar problem to that solved by the singular value decomposition based approach described in Section 4.4. Given two sets of data points, $\{\vec{X}_i\}$ and $\{\vec{Y}_j\}$, we want to know the rotation matrix, **R**, and the translation vector, $\vec{T}$, which transform points in $\{\vec{Y}_j\}$ onto points in $\{\vec{X}_i\}$ so that the cumulative distances between corresponding points is the least possible. However, there are two distinctive things which make this much harder than the problem solved by the method presented in Section 4.4. Firstly, the number of points in the two data sets are not the same. Secondly, and more importantly, the correspondences between data points are not known.

The method presented here must attempt to simultaneously calculate four things: the rotation matrix, the translation vector, the correspondences between points in the two data sets, and which points are outliers.

The correspondence information is represented by a matrix called the *match matrix*. The match matrix represents, for every possible combination of picking one

Algorithm 4 SVD calculation of rigid transformation

Input: $\{\vec{X}_i\}$, $\{\vec{Y}_i\}$, for $i = 1 \dots N$ $(N > 2)$
Output: $\mathbf{R}$, $\vec{T}$, such that $E(\mathbf{R}, \vec{T})$ of Equation (4.3) is minimised

Translate centroids to origin:
$\bar{X} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \vec{X}_i$
$\bar{Y} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \vec{Y}_i$
$\vec{x}_i \leftarrow \vec{X}_i - \bar{X}$
$\vec{y}_i \leftarrow \vec{Y}_i - \bar{Y}$ (see Equation (4.4))

Perform intermediate calculations:
$\mathbf{C} \leftarrow \sum_{i=1}^{N} \vec{y}_i \vec{x}_i^T$ (see Equation (4.6))
$\mathbf{u}$, $\mathbf{v}^T \leftarrow$ result of singular value decomposition of $\mathbf{C}$ (see Equation (4.7))

Compute results:
$$\mathbf{R} \leftarrow \mathbf{v} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(\mathbf{u}\mathbf{v}^T) \end{pmatrix} \mathbf{u}^T \text{ (see Equation (4.10))}$$
$\vec{T} \leftarrow \bar{X} - \mathbf{R}\bar{Y}$ (see Equation (4.9))

point from $\{\vec{X}_i\}$ and one point from $\{\vec{Y}_j\}$, whether those two points are corresponded – that is, whether they correspond to the same point on some object, or at least are very close. The match matrix also represents which points are outliers by not corresponding them to any other points.

Gold *et al.* developed a method for simultaneously calculating this match matrix and the relative orientation (rotation and translation) by starting with an initial guess and improving over many iterations. A full description of the match matrix and a discussion of the problem of assigning to it is described in the text that follows, followed by a description of the algorithm itself.

4.5.1 The Match Matrix and Soft Assignment Techniques

Gold *et al.*'s algorithm incorporates a technique called *soft assign* to iteratively improve the estimate of a match matrix, $\{m_{jk}\}$. For two point sets, $\{\vec{X}_i\}$ and $\{\vec{Y}_j\}$, having J and K points respectively, the match matrix is a $J \times K$ matrix giving the correspondences between the points in the two sets. Traditionally a match matrix should be a permutation matrix, perhaps with extra rows and/or columns of zeros if outlying non-matchable points exist, such as when $J \neq K$. Thus the matrix must satisfy the following constraints:

$$\begin{aligned} &\textstyle\sum_{k=1}^{K} m_{jk} \leq 1 && \forall j \\ &\textstyle\sum_{j=1}^{J} m_{jk} \leq 1 && \forall k \\ &m_{jk} \in \{0, 1\} && \forall j, k \end{aligned} \tag{4.11}$$

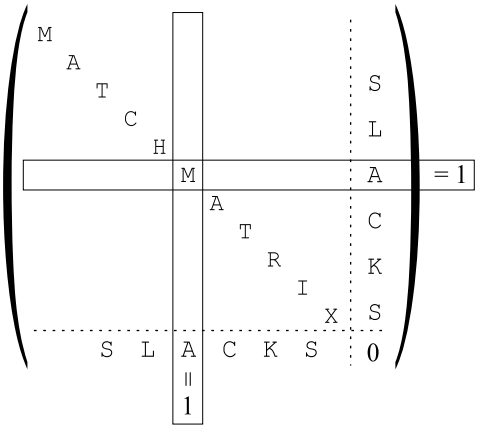


Figure 4.7: Match Matrix with Slack Variables

Gold *et al.* generalised the concept of the discrete match matrix in two ways. Firstly, rather than limiting entries to 0 or 1, entries are given values in the range $[0, 1] \subseteq \mathbb{R}$, enabling the representation of partial matches. The matrix entries are given values using the *soft assign* approach explained below.

Secondly, the inequalities constraining the rows and columns are modified into equalities by introducing *slack variables*, a standard trick in linear programming [22]. Where the original match matrix, $\mathbf{m}$, is a $J \times K$ matrix, the new match matrix is a $(J+1) \times (K+1)$ matrix, and is represented by placing a hat over the variable's symbol: $\hat{\mathbf{m}}$.

The form of $\hat{\mathbf{m}}$ is illustrated in Figure 4.7. A single column and a single row are added, the intersecting entry being set to zero (an important point not mentioned in the original author's paper). The entries within the extra column and extra row take the "slack" when the sum of the columns or rows, respectively, of the original match matrix do not sum to one. The row and column constraints are updated to include values in the extra row and column (the slacks) within their equations so that the rows or columns within the original match matrix are permitted to sum to less than one without the need of inequality.

$$\begin{array}{rl} \sum_{k=1}^{K+1} \hat{m}_{jk} = 1 & \forall j \\ \sum_{j=1}^{J+1} \hat{m}_{jk} = 1 & \forall k \\ m_{jk} \in [0, 1] \subseteq \mathbb{R} & \forall j, k \end{array} \tag{4.12}$$

Gold *et al.*'s algorithm attempts to find a solution to what is known as an *assignment problem* [67]:

> Given a matrix, $\{Q_{jk}\}$, where $Q_{jk} \in \mathbb{R}$, associate a variable m_{jk} with each Q_{jk} and assign the m_{jk} values satisfying the constraints in

Equation (4.11) and the constraint

$$m_{jk} = \begin{cases} 1 & \text{if } Q_{jk} \text{ is the maximum number in row } j, \text{ and in column } k \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

The above problem is equivalent to finding the $\{m_{jk}\}$ which maximise the equation $\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}Q_{jk}$. This discrete problem can be reformulated as a continuous problem by doing three things. Firstly, m_{jk} are allowed to hold values in the range $[0,1] \subseteq \mathbb{R}$ as already suggested. Secondly, a control parameter $\beta > 0$ is introduced which will be given an initially low value and increased. Thirdly, an extension to 2D of the following well known equation is used to normalise $\mathbf{m}$:

$$y_i = \frac{\exp(\beta x_i)}{\sum_{j=1}^{N} \exp(\beta x_j)} \tag{4.14}$$

Equation (4.14) is known as the *softmax* [14] equation. It can be shown that, as β is increased, the y_i corresponding to the maximum x_i approaches 1 while all other y_i approach 0. An exception occurs when the x_i contain ties, in which case the corresponding y_i will receive a fraction of the value of 1 equal to one over the number of ties.

The *softmax* equation can be extended to handle doubly stochastic constraints by a result due to Sinkhorn [79]. Sinkhorn proved that a doubly stochastic matrix can be obtained from a square matrix by an iterative process of alternating row and column normalisations, provided that the original matrix contains only zero or positive values.

Using the *softmax* equation with the process of increasing the control parameter, β, to assign values to $\mathbf{m}$ is called *soft assign.*

4.5.2 Description of Algorithm

The original authors did not explain exactly some important details and so the algorithm we use, and describe below, is slightly modified from their description in order to make it work successfully in our tests.

Gold *et al.*'s algorithm takes two sets of 3D points $\{\vec{X}_j\}$ and $\{\vec{Y}_k\}$ and attempts to find the rotation matrix, $\mathbf{R}$, the translation vector, $\vec{T}$, and the match matrix, $\mathbf{m}$, which registers data set $\{\vec{Y}_k\}$ onto $\{\vec{X}_k\}$ by minimising the error function below and maintaining the constraints on m already stated. The error function is:

$$E(\mathbf{m}, \vec{T}, \mathbf{R}) = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}\|\vec{X}_j - (\vec{T} + \mathbf{R}\vec{Y}_k)\|^2 - \alpha\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk} \tag{4.15}$$

For fixed $\vec{T}$ and $\mathbf{R}$, the task of minimising the error function can be reduced to

β	control parameter of the deterministic annealing method
β_0	initial value of the control parameter β
β_f	maximum value of the control parameter β
β_r	rate at which the control parameter β is increased ($\beta_r > 1$)
E	point matching objective, Equation (4.15)
$\{m_{jk}\}$	match matrix variables (excluding slacks)
$\{\hat{m}_{jk}\}$	match matrix variables including slack variables (see Figure 4.7)
$\{Q_{jk}\}$	*assignment problem* variables
τ_0	maximum number of iterations allowed for each value of β
τ_1	maximum number of iterations allowed for Sinkhorn's method
δ_0	convergence criteria for transformed position of points in $\{\vec{Y}_{jk}\}$
δ_1	convergence criteria for Sinkhorn's method
ϵ	a very small number

Table 4.1: Variable and constant definitions used within Gold *et al.*'s point matching algorithm

solving the doubly stochastic assignment problem where:

$$Q_{jk} = -(\|\vec{X}_j - (\vec{T} + \mathbf{R}\vec{Y}_k)\|^2 - \alpha) = -\frac{\partial E}{\partial m_{jk}} \tag{4.16}$$

(the sign is reversed because we are minimising the total error, rather than maximising a value as in the original form of the assignment problem).

The error function contains an extra parameter, α, used to discourage the algorithm from unnecessarily rejecting points as outliers. α effectively sets a weight on how important it is to find point-pair correspondences. In our tests, we found setting α to zero yielded the best results and did not cause points to be unnecessarily rejected as outliers. Further to this, our implementation actually normalises across the slack variables in the last row and down the slack variables in the last column, thus the contents of the rows and columns do not always sum to one. No doubt this is evidence that we have not implemented Gold's algorithm in the way originally intended, however we found our implementation to be successful nonetheless and we found better results were achieved this way than when we did not normalise along the extra column and row.

Table 4.1 describes the variables and constant definitions used within the point set matching algorithm. Algorithm 5 lists pseudo-code describing the process. Equation (4.16) is used to calculate the values of matrix $\mathbf{Q}$. $\hat{\mathbf{m}}$ is assigned to using the form of Figure 4.7, initially setting the slack variables to one plus a very small number. $\hat{\mathbf{m}}$ is normalised using Sinkhorn's method within Block D, using τ_1 as an upper limit on the number of iterations to avoid infinite loops in the case of oscillations (a possibility when normalisation does not have a unique solution). Convergence of Sinkhorn's method is detected when the average entry-wise change in $\hat{\mathbf{m}}$ is less than threshold δ_1.

The relative orientation is updated from the normalised match matrix, $\mathbf{m}$, using the method described below. This is followed by repeating the process of calculating $\mathbf{Q}$ with the new orientation information. This is repeated until the orientation parameters, $\mathbf{R}$ and $\vec{T}$, have converged. Convergence of the orientation parameters is detected by examining the greatest distance moved by any of the points within $\{\vec{Y}_i\}$ between successive iterations of Block B: if the greatest distance moved is less than threshold δ_0, then Block B terminates. A fixed upper limit on the number of iterations of Block B is applied using parameter τ_0.

The above process is repeated, using the updated values for orientation, until β is large enough to cause the entries within $\mathbf{m}$ to converge to close to zero or one, as determined by setting parameter β_f.

To give an idea of the magnitude of values used for some of the parameters, Gold *et al.* used the following values after a process of trial and error: $\beta_0 = 0.01/S$, $\beta_r = 1.053$, $\beta_f = 100/S$, $\tau_0 = 10$, $\tau_1 = 30$, where $S = \frac{1}{JK}\sum_{jk}\|\vec{X}_j - \vec{Y}_k\|^2$, the mean distance between points. They mention that Block D does not need to converge exactly and so set τ_1 to a small number. In our experiments it was found that $\hat{\mathbf{m}}$ does indeed usually converge within that limit, however, when it does not, it is unwise to limit the number of iterations in such a simplistic approach. We found that, if α is set non-zero, Q_{jk} can become non-negative, causing m_{jk} to become very large. In this case it can take up to many hundreds of iterations to converge, depending on the value of δ_1. If, however, Block D is not given sufficient iterations to converge, the algorithm can become unstable or produces bad results. It is considered better to only use τ_1 to avoid infinite loops, or handle this situation in a different fashion.

Walker *et al.* [95] present a method for calculating the orientation of two sets of points with known correspondences, and Gold *et al.* extended the method to use the "fuzzy" correspondence matrix, $\mathbf{m}$. The rotation and translation are represented by a dual number quaternian, (r, s), corresponding to a screw coordinate transform. Here, $r = (r_1, r_2, r_3, r_4)^T$, $s = (s_1, s_2, s_3, s_4)^T$, and r and s are subject to the constraints $r^T r = 1$ and $r^T s = 0$. The rotation can be written as $\mathbf{R}(r) = \mathbf{W}(r)^T\mathbf{P}(r)$ and the translation as $T(r, s) = \mathbf{W}(r)^T s$, where:

$$\mathbf{W}(r) = \begin{pmatrix} r4 & r3 & -r2 & r1 \\ -r3 & r4 & r1 & r2 \\ r2 & -r1 & r4 & r3 \\ -r1 & -r2 & -r3 & r4 \end{pmatrix}$$

$$\mathbf{P}(r) = \begin{pmatrix} r4 & -r3 & r2 & r1 \\ r3 & r4 & -r1 & r2 \\ -r2 & r1 & r4 & r3 \\ -r1 & -r2 & -r3 & r4 \end{pmatrix}$$

Algorithm 5 Point Set Matching

Input: $\{\vec{X}_j\}$, $\{\vec{Y}_k\}$

Output: $\mathbf{R}$, $\vec{T}$, $\{m_{jk}\}$, such that $E(\mathbf{m}, \vec{T}, \mathbf{R})$ of Equation (4.15) is minimised

$\vec{T} \leftarrow (0,0,0)$

$\mathbf{R} \leftarrow \mathbf{I}_3$ (3×3 identity matrix)

$\beta \leftarrow \beta_0$

$maxMoved \leftarrow \infty$

$\forall k,\ \vec{Y}_{previous,k} \leftarrow \vec{Y}_k$

Begin A: Do A until $\beta \geq \beta_f$

Begin B: Do B until $maxMoved < \delta_0$ or # of iterations $> \tau_0$

Begin C (update correspondence parameters by softassign):

$Q_{jk} \leftarrow -\frac{\partial E}{\partial m_{jk}}$

$\hat{m}^0_{jk} \leftarrow \begin{bmatrix} [\exp(\beta Q_{jk})] & (1+\epsilon) \\ (1+\epsilon) & 0 \end{bmatrix}$

Begin D: Do D until $\hat{m}$ converges or # of iterations $> \tau_1$

Remember current value of $\hat{\mathbf{m}}$:

$\forall j,k,\ \hat{m}'_{jk} \leftarrow \hat{m}_{jk}$

Update $\hat{m}$ by normalising across all rows:

$\forall j,k,\ \hat{m}^1_{jk} \leftarrow \frac{\hat{m}^0_{jk}}{\sum_{l=1}^{K+1} \hat{m}^0_{jl}}$

Update $\hat{m}$ by normalising across all columns:

$\forall j,k,\ \hat{m}^0_{jk} \leftarrow \frac{\hat{m}^1_{jk}}{\sum_{l=1}^{J+1} \hat{m}^1_{lk}}$

$\hat{\mathbf{m}}$ has converged if $\frac{\sum_{j=1}^{J+1}\sum_{k=1}^{K+1} |m'_{jk} - m_{jk}|}{(J+1)(K+1)} < \delta_1$

End D

End C

Begin E (update relative orientation parameters):

$\mathbf{R}$, $\vec{T} \leftarrow$ update orientation using Walker *et al.*'s method

Detect convergence of $\mathbf{R}$ and $\vec{T}$:

$maxMoved \leftarrow \max_k \|(\vec{T} + \mathbf{R}\vec{Y}_k) - \vec{Y}_{previous,k}\|$

$\forall k,\ \vec{Y}_{previous,k} \leftarrow (\vec{T} + R\vec{Y}_k)$

End E

End B

$\beta \leftarrow \beta_r \beta$

End A

Substituting these into Equation (4.15) the error function becomes:

$$E = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk} \|x_j - \mathbf{W}(r)^T s - \mathbf{W}(r)^T \mathbf{P}(r) y_k\|^2$$

where $x_j = (\vec{X}_j, 0)^T$ and $y_k = (\vec{Y}_k, 0)^T$ are the quaternion representations of $\vec{X}_j$ and $\vec{Y}_k$, respectively. The error function can be reformulated using the properties that

$\mathbf{P}(a)b = \mathbf{W}(b)a$ and $\mathbf{P}(a)^T\mathbf{P}(a) = \mathbf{W}(a)^T\mathbf{W}(a) = (a^Ta)\mathbf{I}$. The error function thus becomes:

$$E = r^T\mathbf{C}_1r + s^T\mathbf{C}_2s + s^T\mathbf{C}_3r + \text{constant}$$

where:

$$\mathbf{C}_1 = -2\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}\mathbf{P}(y_k)^T\mathbf{W}(x_j)$$

$$\mathbf{C}_2 = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}\mathbf{I}$$

$$\mathbf{C}_3 = 2\sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}(\mathbf{W}(x_j) - \mathbf{P}(y_k))$$

The objective function is now minimised by (r^*, s^*) where r^* is the eigenvector corresponding to the largest positive eigenvalue of matrix $\mathbf{A} = \frac{1}{2}C_3^T(\mathbf{C}_2+\mathbf{C}_2^T)^{-1}\mathbf{C}_3 - (\mathbf{C}_1 + \mathbf{C}_1^T)$, and $s^* = -(\mathbf{C}_2 + \mathbf{C}_2^T)^{-1}\mathbf{C}_3r^*$. The rotation and translation is then computed from r^* and s^* using the equations for $\mathbf{R}(r)$ and $\mathbf{T}(r, s)$. For derivation of this solution the reader is referred to Walker *et al.*'s original paper [95].

Note that the equations for $\mathbf{A}$ and s^* can be simplified to $\mathbf{A} = \frac{1}{4n}\mathbf{C}_3^T\mathbf{C}_3 - \frac{1}{2}(\mathbf{C}_1 + \mathbf{C}_1^T)$ and $s^* = -\frac{1}{2n}\mathbf{C}_3r^*$, where $n = \sum_{j=1}^{J}\sum_{k=1}^{K} m_{jk}$, making the implementation more efficient.

A consistent match matrix, $\mathbf{M}$, where $M_{jk} \in \{0, 1\}$, can be constructed from the output of Gold *et al.*'s algorithm by the following process. Firstly, compute two intermediate matrices, one produced by setting to TRUE the entries corresponding to the largest value in each row within $\mathbf{m}$, the other produced by setting to TRUE the entries corresponding to the largest value in each column within $\mathbf{m}$. All other entries within these two intermediate matrices are set to FALSE. Entries within $\mathbf{M}$ are then computed by taking the logical AND of the corresponding entries within the two intermediate matrices and setting the entry within $\mathbf{M}$ to one if the result is TRUE, zero otherwise.

Formally, construct two intermediate matrices $\mathbf{a}$ and $\mathbf{b}$ with all but the following entries set to FALSE:

$$\forall j = 1 \ldots J, \quad l = \underset{n}{\text{argmax}}\ m_{jn}, \quad a_{jl} = \text{TRUE}$$

$$\forall k = 1 \ldots K, \quad l = \underset{n}{\text{argmax}}\ m_{nk}, \quad b_{lk} = \text{TRUE}$$

Then construct **M** as follows:

$$M_{jk} = \begin{cases} 1 & \text{if } a_{jk} = \text{TRUE} \quad \text{AND} \quad b_{jk} = \text{TRUE} \\ 0 & \text{otherwise} \end{cases}$$

Notice that the expanded form of M_{jk}, below, looks much like the constraint given in Equation (4.13).

$$M_{jk} = \begin{cases} 1 & \text{if } k = \underset{n}{\text{argmax}}\ m_{jn} \quad \text{AND} \quad j = \underset{n}{\text{argmax}}\ m_{nk} \\ 0 & \text{otherwise} \end{cases}$$

Chapter 5

Implemention of New Methods for Surface Registration

5.1 Overview

This chapter presents two methods for 3D surface registration. The more complete task of surface recognition is covered briefly to illustrate how 3D surface registration can be incorporated and an efficient system devised. However, the focus is on 3D surface registration, specifically, as stated in the Introduction and Background chapters, on registering 3D surfaces efficiently in the presence of occlusions.

The surface registration methods given are: "Decomposition, Match Assignment, and RANSAC Alignment" (DMARA) and "Decomposition and Point Set Alignment" (DPSA). These methods take as inputs two 3D surface models, represented as polygon meshes. One of these surfaces, the *current surface*, is captured from the current immediate surroundings. The second surface, the *database surface*, is retrieved from the database of observed location surfaces. For the sake of simplicity, only triangular polygon meshes are considered, however most parts of the algorithms used are sufficiently generic enough to work on other polygon meshes, and where they are not, it is possible to extend them. The outputs from the registration methods contain the rotation and translation necessary to describe the pose of the current surface relative to the database surface, and contain a measure of how well or how badly the current surface is aligned onto the database surface. This information is used within a larger system to select the most likely match to a database surface and thus determine the robot's location within its map.

DMARA (pronounced "dee–mara") uses surface decomposition to aid in the extraction of features. The decomposed regions, called *patches*, are themselves used directly as features for part of the processing. A tentative correspondence from each of the current surface patches to the most similar database surface patch is found by an exhaustive search over the patches. Similarity between patches is determined using the Shape Distributions method described in Chapter 4. The RANSAC algo-

rithm, also described in Chapter 4, is used to attempt to find the best registration of the current surface onto the database surface using a random selection process. The tentative patch correspondences are used to guide the random selection process within the RANSAC algorithm.

DPSA, like DMARA, uses surface decomposition to aid in feature extraction. However, the decomposed patches are matched and the registration process is performed differently to that of DMARA. DPSA does not perform any separate processing step to find tentative patch correspondences. Instead, Gold *et al.*'s method for uncorresponded point set alignment is used to simultaneously find the optimal patch correspondence assignment and attempt to find the best registration of the current surface onto the database surface. Gold *et al.*'s method is used to attempt to find the best alignment between two point sets, the point sets being the centre points of the decomposed patches from each of the current surface and database surface.

A short description of how these 3D surface registration algorithms can be applied to the task of 3D surface recognition follows within the next section. The remainder of the chapter gives details on the implementation of DMARA and DPSA.

5.1.1 Description of a Full Surface Recognition System

In a complete robot navigation system, the robot will observe and record information about a vast number of separate areas. These areas may be individual rooms, or distinct parts of larger rooms. Representations of these areas are assumed to be stored in a database as a list of distinct surfaces and robotic localisation is to be performed by finding the database surface onto which the current surface can be most accurately registered. However, accurately registering the current surface onto each and every surface within the database is much too time consuming for a robot to do.

The system assumed for the purposes of this thesis is that a two phase process of eliminations is used, as indicated by Figure 5.1. The first phase uses a course registration process which can be computed quickly for every database surface. Those database surfaces which poorly match the current surface are eliminated during this first phase. The second phase uses a fine registration algorithm to accurately register the current surface onto each of those remaining database surfaces. The registration information resulting from the course registration algorithm of phase one is used as an initial registration for phase two. After phase two, the database surface onto which the current surface is best registered is selected as the winner.

The registration error between the winning database surface and the current surface is used to determine whether to accept the winning database surface and use the relative pose information to specify the robot's location, or to decide that the database surface is sufficiently different to the current surface that the robot must

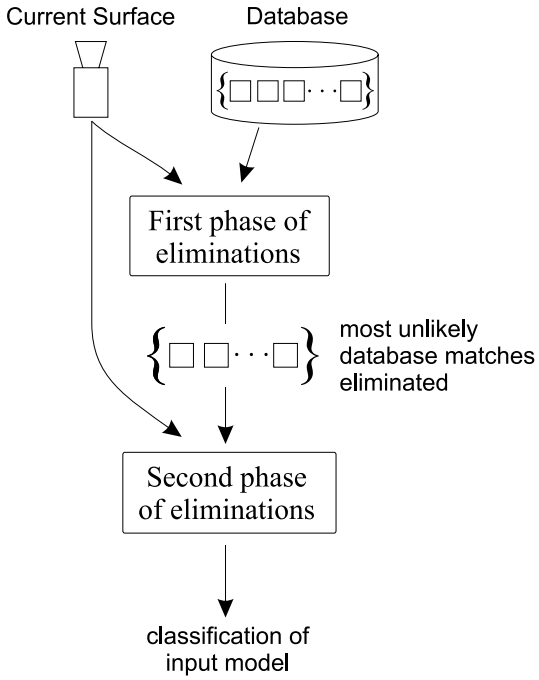


Figure 5.1: Overview of recognition process
The input model is compared against all database models and either a single database model is selected as the closest matching model, or "no match" is returned.

have entered an area which it has not previously observed.

Many optimisations are possible within this framework. For example, an extra initial phase could be used to more quickly eliminate the very most unlikely database surfaces based solely on the number of matching features. The features used within such a process, and the many other optimisations which are necessary to make robotic localisation possible, are well outside the scope of this thesis.

5.2 Registration Method One – DMARA

DMARA, "Decomposition, Match Assignment, and RANSAC Alignment", is used to find a course registration of two 3D surfaces. It is intended that one of these surfaces is captured from the current immediate surroundings, the *current surface*, and that the other is selected from a database of surfaces representing the surroundings at each location previously observed by the robot. The surface selected from the database is called the *database surface*. Generally the current surface will contain less information than the database surface because the database surface will have been acquired from multiple views and combined as one surface representation, whereas the current surface may have been acquired from only one view.

DMARA uses decomposition to extract features by dividing a surface into many patches. Each patch is then processed and assigned a *patch descriptor*. The patch descriptors are signatures measuring the general shape of each patch. This is used

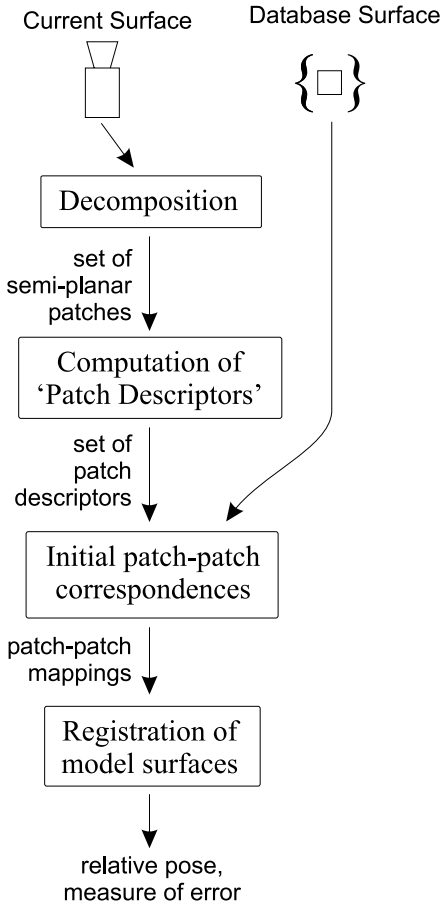


Figure 5.2: Overview of DMARA registration process
The current surface is taken as a raw polygon mesh and pre-processing performed. The database surface is assumed to have that pro-processing already done and stored with the surface. The results of decomposition and calculation of the patch descriptors are used to calculate tentative patch-to-patch correspondences. This correspondence information is used to guide the registration process.

to determine a tentative correspondence between patches by finding patches with similar shape. Shape Distributions are used as patch descriptors. Final registration is performed by randomly selecting patches from the current surface and using the tentative correspondence list to select corresponding patches within the database surface. The centres of patches are used to calculate the relative pose of the two surfaces.

The decomposition process and the Shape Distribution calculation produce results which are assigned to each surface. These results can be pre-processed for all surfaces contained within the database. As such, the DMARA algorithm presented here assumes that all surfaces within the database are assigned with decomposition and Shape Distribution information.

The process used by DMARA is described pictorially in Figure 5.2. Two inputs, the current surface and the database surface, are accepted. The current surface undergoes decomposition, producing a set of patches which are approximately planar.

Each patch is assigned a patch descriptor, representing the shape of the patch, by calculating the Shape Distribution for that patch. Initial, tentative, correspondences are calculated between patches in the current surface and patches in the database surface. Registration is performed using the RANSAC algorithm. The output of the DMARA process is a rotation and translation representing the relative pose of the surfaces, and a measure of the alignment between the surfaces, indicating whether the surfaces were closely aligned or have large gaps between them.

5.2.1 Decomposition for Feature Extraction

The decomposition process largely follows that of the Watershed Decomposition algorithm presented in Chapter 4. Values for height function, h, are computed from the angles between faces (that is, across the edges between the faces) using Equation (4.1)[page 39] and then assigned to each face. The algorithm presented in Chapter 4 assigns heights to each face by selecting the minimum of the heights for each of the face's edges. However, experimentally it was found that this gives low heights to some surface edges due to the tessellation. This causes the decomposition to ignore some surface edges and combine two regions into one when they should not be. Experiments also found that selecting the *maximum* yielded better results and thus this small change is applied to the Watershed Decomposition algorithm.

The Watershed Decomposition algorithm contains a post-processing step which attempts to merge shallow catchment basins to form larger patches. The shallow catchment basins are caused by noise in the original surface. It was found, however, that this still produces many patches for a single mostly-flat, surface region. For example, due to sensor noise, an initially planar surface may be decomposed into more than one patch. To rectify this, patches are further merged if they are close to co-planar.

5.2.2 Tentative Feature Matching

The RANSAC algorithm can be used to perform a random search for the best pose. However, it requires a tentative correspondence between patches in the current surface to patches in the database surface to be computed. This tentative correspondence information helps the RANSAC algorithm to be efficient. The patch correspondences represent a match from each patch in the current surface to any of the patches in the database surface. Uniqueness of matches is not maintained because this causes unnecessary computational expense. Neither do all matches have to be correct. The RANSAC algorithm is designed to handle outlying data such as caused by a match between two patches which should not be matched. However, the greater the number of correctly matched patches the more efficient the RANSAC algorithm is.

The correspondences between patches can be visualised as in Table 5.1, where

current surface patches	1	2	3	4	5	6	$\cdots$	39	40
database surface patches	54	23	5	11	54	39	$\cdots$	4	15

Table 5.1: Illustration of correspondences between patches

some of the correspondences from the patches within the current surface, having 40 patches, to the patches within the database surface, having 60 patches, are shown. Matches are found for every patch within the current surface, but not all patches within the database surface have matches. If the current surface and database surface are decomposed into N and M patches, respectively, then the correspondence information is represented by an N length list, $\{p_i\} = \{p_1, p_2, \ldots, p_N\}$, where each $p_i \in \{1, 2, \ldots, M\}$.

Determining the values for $\{p_i\}$ is performed by computing some form of patch descriptor for each patch and then finding, for each patch in the current surface, the most similar patch in the database surface. If an error measure is defined, $patchMatchError(\vec{s}, \vec{t})$, which compares two patch descriptors, $\vec{s}$ and $\vec{t}$, then the entries for p_i are found by:

$$p_i = \underset{j \in \{1,2,\ldots,M\}}{\operatorname{argmax}} \; patchMatchError(\vec{x}_i, \vec{y}_j) \tag{5.1}$$

where $\vec{x}_i$ is the patch descriptor for the i^{th} patch of the current surface, and $\vec{y}_j$ is the patch descriptor for the j^{th} patch of the database surface.

Matching from the database surface to the current surface can yield a different set of correspondences between the patches. For example, if the same process as Equation (5.1) is performed from database surface to current surface, producing the list $\{g_j\}$, one might assume that if $p_i = j$ then $g_j = i$, and *vice versa*. However, this is not the case because the argmax function maximises over a different set of patches. Despite this, the database surface is assumed to contain a corresponding patch for the majority of patches within the current surface, and so this issue is simply disregarded by only performing correspondence calculations in one direction: from current surface patches to database surface patches.

One suitable patch descriptor is the "Patch Distribution" of Osada *et al.* [66]. Its use within DMARA follows.

5.2.3 Patch Distributions as Patch Descriptor

Patch Distributions were initially intended for 3D surface models, however they are also effective for matching semi-planar patches by measuring the distribution of Euclidean distances between points along the edge of the patches.

Figure 5.3 illustrates the Shape Distributions for two corresponding patches selected from the data presented in Chapter 6. Patch (a) and (b) are similar, however

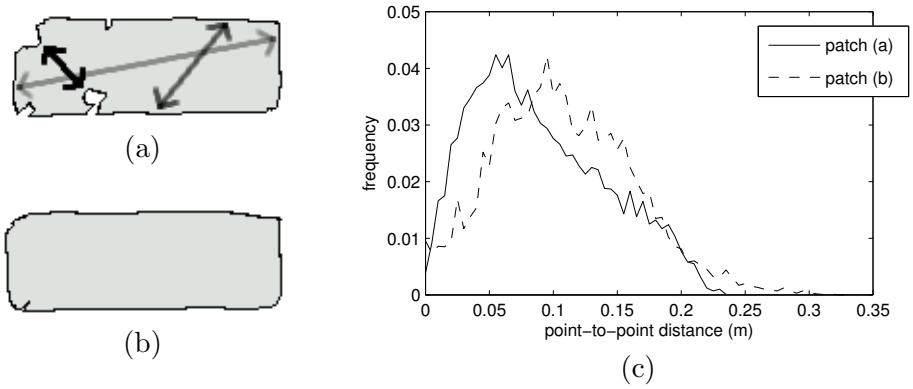


Figure 5.3: Example patch distributions
(a) a single patch selected from a decomposed surface.
(b) the corresponding patch on another surface of the same physical region.
(c) the shape distributions for these patches.

patch (a) has four small sections removed from it. As a consequence, the average distance between edge points in patch (a) is less than the average distance between edge points in patch (b). As Figure 5.3(c) shows, patch (a) has its greatest frequency at a distance of about 0.05 m, whereas patch (b) has its greatest frequency at distance of about 0.1 m.

Osada *et al.*, who developed Shape Distributions, went to great lengths to ensure that the tessellation of the triangular mesh does not affect the calculation of the distributions. They uniformly selected faces and then selected points uniformly over the area of each face, in order to guarantee that points are sampled across the surface with uniform distribution regardless of the relative size and shape of the faces within the surface. However, Osada *et al.* applied Shape Distributions to recognition of 3D models with very low noise. The models they used were generated using CAD software, whereas the patches generated from the decomposition process described here have a high level of noise. For example, the average distance between points on the edge of patches (a) and (b) from Figure 5.3 is 0.0921 m and 0.1108 m, respectively, while the average face occupies a square of size 0.0011 m $\times$ 0.0011 m. The difference in average edge-point to edge-point distance caused by noise and the decomposition process is 0.0187 m, but the difference which might be caused by badly sampling points on a face is something close to only 0.0011 m. As a consequence, it is not worthwhile to take the extra effort to perform accurate uniformly distributed sampling, and shortcuts can be made.

A detailed description of the implementation used is now given. This implementation takes a single patch and computes a Shape Distribution for it. This is done for all patches. The distribution is produced by taking N samples of Euclidean distances measured between points along the edge of the patch. A histogram is formed by counting the number of samples which fall into each of B bins. The bins are of fixed width, equal to B_{width}, and B is determined by the maximum sampled

distance, divided by B_{width}. The resultant histogram is represented as a vector, $\vec{s} = (s_1, s_2, \ldots, s_B)$.

The faces which belong to the edge of the given patch are extracted by selecting only those faces which have less than three adjacent faces (remember that only triangular faces are being used, a more complicated approach would be required for generic polygon meshes). Points are chosen by randomly selecting faces and using the centres of those faces. To help select patch edge points more uniformly, faces are selected with probability proportional to their areas. This is done in a similar fashion to that of the original Shape Distribution algorithm. For each face, its area is computed and a reference to it is stored in an array along with the cumulative area of faces visited so far. Faces are then selected by generating a random number between 0 and the total cumulative area and the face corresponding to that cumulative area is selected.

The final vector representing the histogram of distances, $\vec{s}$, is scaled so that it sums to 1.0, as in the following:

$$s'_i = \frac{s_i}{\sum_{j=1}^{B} s_j}$$

The apostrophe is dropped in later uses of this vector, but it is always scaled to sum to 1.0. $\vec{s}$ now represents the probability distribution over distances between edge points.

Osada *et al.* defined the difference of two linear piecewise shape distributions, f and g, using Equation (4.2)[p. 46], repeated below:

$$D(f, g) = \int |f - g|$$

The corresponding difference between two discretised shape distributions, $\vec{s}$ and $\vec{t}$, is:

$$D(\vec{s}, \vec{t}) = \sum_i |s_i - t_i| \tag{5.2}$$

where $|\cdot|$ is the absolute value operator. However, $\vec{s}$ and $\vec{t}$ may be of different lengths. To handle this, the shorter of the two vectors is extended to the same length as the other, filling the extra entries with zero. This is perfectly valid as the absence of a count at the extra positions simply indicates a zero probability for the corresponding point-to-point distance. The difference between two discrete shape distributions, $\vec{s}$ and $\vec{t}$, of lengths J and K, respectively, is:

$$D(\vec{s}, \vec{t}) = \sum_{i=1}^{max(J,K)} \begin{cases} |s_i - t_i| & i \leq J \text{ and } i \leq K \\ s_i & i > K \\ t_i & i > J \end{cases} \tag{5.3}$$

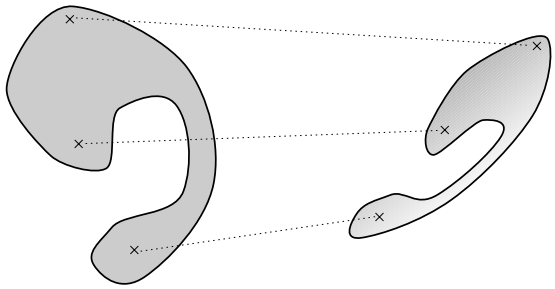


Figure 5.4: Required data to compute rotation and translation of 3D surfaces

The definition of $D(\vec{s}, \vec{t})$, above, is used for the *patchMatchError* function of Equation (5.1).

5.2.4 Registration using RANSAC

The RANSAC algorithm, as described in Chapter 4, attempts to fit a model to the given data using an iterative process. The model represents whatever is being fit to the data. In a linear regression problem, the model is a line, with two parameters. In the case of registering two surfaces, the model is the relative pose, represented by rotation and translation. RANSAC uses the minimum number of points necessary to compute the parameters for the model. The linear regression scenario requires only two points. To compute the rotation and translation to register two surfaces in 3D, three pairs of points are needed. The first point of each pair must be from the first surface, and the second point of each pair must be from the second surface, as illustrated in Figure 5.4.

Singular value decomposition (SVD) is used to compute the rotation and translation (described in Chapter 4).

The original RANSAC algorithm is modified to the specific task of finding the registration of two surfaces. The modified RANSAC algorithm, "mRANSAC", takes the following inputs:

- a list of patch correspondences, $\{p_i\} = \{p_1, p_2, \ldots, p_N\}$ with $p_i \in \{1, 2, \ldots, M\}$, from N patches in the current surface to M patches in the database surface,
- a list of patch centres, $\{\vec{C}_{C,i}\}$ and $\{\vec{C}_{D,j}\}$ such that $i \in \{1, 2, \ldots, N\}$ and $j \in \{1, 2, \ldots, M\}$, for the current surface patches and the database surface patches, respectively,
- and a list of the centre points of all faces within the original surfaces themselves, $\{\vec{F}_{C,k}\}$ and $\{\vec{F}_{D,l}\}$ such that $k \in \{1, 2, \ldots, K\}$ and $l \in \{1, 2, \ldots, L\}$, for the current and database surfaces with K and L faces, respectively.

The face centres are calculated by taking the average vertex coordinate for each face, and each patch centre is computed by taking the average of the face centres for the faces within that patch.

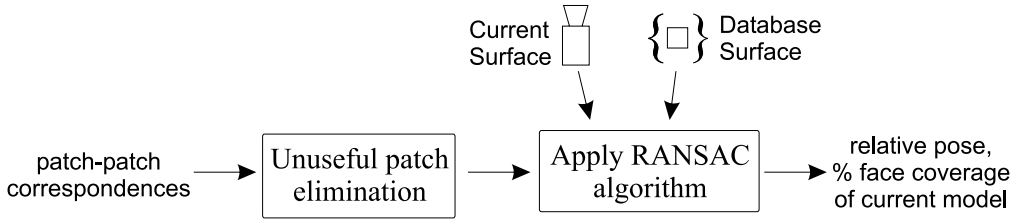


Figure 5.5: Incorporating RANSAC

The patch correspondences, the patch centres, and the face coordinates are used to find the registration for the surfaces through the usual random selection process of RANSAC. The result is the best selected relative pose and a measure of how well the surfaces are aligned, as indicated by Figure 5.5. The percentage of current surface faces which are "covered", within some error tolerance of database surface faces, are used to measure the successfulness of the registration. An optional pre-processing stage may be used to eliminate outlying patches which are not useful for the registration process.

The mRANSAC algorithm attempts to find the rotation and translation which registers the current surface onto the database surface. The mRANSAC algorithm transforms the current surface using the rotation and translation during this process, while the database surface remains in a static position. For this reason, it is sometimes more convenient to refer to the current surface as the *dynamic surface*, and the database surface as the *static surface* for the purposes of this part of the registration.

The mRANSAC algorithm is given in Algorithm 6. Each iteration uses two phases. The first randomly selects three patches from the current surface. The patch correspondence list, $\{p_i\}$, is used to determine the corresponding patches in the database surface. The centres of these patches are used to compute the rotation and translation. The consensus set is formed by finding those current surface faces which have a face within the database surface close to it. For each k^{th} face within the current surface, $closest_k$ is assigned the index, l, to the closest face within the database surface. Each of $distance_k$ are then assigned the distance between the k^{th} face within the current surface and the chosen l^{th} face within the database surface, as indicated by $closest_k$. The size of the consensus set is then determined by simply counting the number of distances which are no greater than the maximum distance threshold, *maxDistThreshold*. If the consensus set is equal or greater than the minimum threshold for the consensus set, *minConsensusCountThreshold*, then accept this consensus set of faces and move to phase two, otherwise reject it and try again. *minConsensusCountThreshold* would likely bo set to some fixed percentage of the total number of faces within the current surface.

The second phase of mRANSAC re-computes the rotation and translation using all faces within the consensus set. The relative pose computed using three patch

Algorithm 6 Modified RANSAC (mRANSAC)

Input:
$\{p_i\}$, a length N vector of patch correspondences;
$\{\vec{C}_{C,i}\}$ and $\{\vec{C}_{D,j}\}$, the patch centres;
$\{\vec{F}_{C,k}\}$ and $\{\vec{F}_{D,l}\}$, the coordinates of face centres.
Require: $N \geq 3$ (to ensure that the rotation and translation can be calculated)
Output: $\mathbf{R}, \vec{T}$, the rotation matrix and the translation vector, and some measure of surface registration accuracy.

Begin A: Repeat while number of iterations $\leq t_{\max}$
Begin Phase One (approximate registration from three patches):
Select three patches from the current surface, and use $\vec{p}$ to
select three corresponding patches from the database surface:
$a, b, c \leftarrow$ randomly select three values from $\{1, 2, \ldots, N\}$
$d, e, f \leftarrow p_a, p_b, p_c$

Get centres of selected patches:
$patchCentres_C \leftarrow \{\vec{C}_{C,a}, \vec{C}_{C,b}, \vec{C}_{C,c}\}$
$patchCentres_D \leftarrow \{\vec{C}_{D,d}, \vec{C}_{D,e}, \vec{C}_{D,f}\}$

Compute rigid transformation:
$\mathbf{R}, \vec{T} \leftarrow$ ComputeTransformation($patchCentres_C$, $patchCentres_D$)

Apply transformation to current surface using $\mathbf{R}$ and $\vec{T}$:
$\vec{G}_{C,k} \leftarrow \mathbf{R}\vec{F}_{C,k} + \vec{T}, \qquad \forall k \in \{1, 2, \ldots, K\}$

Find faces in database surface which are closest to faces in current surface:
$closest_k \leftarrow \underset{l \in \{1,2,\ldots,L\}}{\operatorname{argmin}} \|\vec{G}_{C,k} - \vec{F}_{D,l}\|, \qquad \forall k \in \{1, 2, \ldots, K\}$
$distance_k \leftarrow \|\vec{G}_{C,k} - \vec{F}_{D,l}\|$, where $l = closest_k, \qquad \forall k \in \{1, 2, \ldots, K\}$
($\|\cdot\|$ denotes the Euclidean distance operator)

Test size of consensus set:
$consensusCount \leftarrow$ number of $distance_j$ which are $\leq maxDistThreshold$
if $consensusCount < minConsensusCountThreshold$ **then**
skip Phase Two and begin again from Phase One.
end if
End Phase One

(*continued on page 70*)

centres may be inaccurate, and the second phase helps to improve the calculation. The new rotation and translation is calculated by treating the list $\{closest_k\}$ as giving the correspondences from faces within the current surface to faces within the database surface, in the same way as $\{p_i\}$ gives the correspondences between patches. The updated rotation and translation is computed from the centres of the corresponding faces.

(*continued from page 69*)

All faces in the current surface satisfying
$distance_k \leq maxDistThreshold$ are considered non-outliers.
Begin Phase Two (update registration from all non-outlying points):
Get coordinates of selected faces:
$faceCentres_C \leftarrow \{\vec{F}_{C,k} | distance_k \leq maxDistThreshold\}$
$faceCentres_D \leftarrow \{\vec{F}_{D,l} | l = closest_k \text{ and } distance_k \leq maxDistThreshold\}$

Compute rigid transformation from non-outlying points:
$\mathbf{R}^*, \vec{T}^* \leftarrow$ ComputeTransformation($faceCentres_C$, $faceCentres_D$)

Apply transformation to current surface using $\mathbf{R}^*$ and $\vec{T}^*$:
$\vec{G^*}_{C,k} \leftarrow (\mathbf{R}^*)\vec{F}_{C,k} + \vec{T}^*, \qquad \forall k \in \{1, 2, \ldots, K\}$

Compute surface fit error or accuracy (see Section 5.2.6)
End Phase Two
End A Return $\mathbf{R}$ and $\vec{T}$ from iteration with least surface fit error.

Two important points must be addressed before this algorithm is effective:

1. How can the closest faces be efficiently selected in order to compute $\{closest_k\}$?
2. How is the surface fit error or surface fit accuracy measured?

These issues are addressed in the following sections.

5.2.5 Finding Closest Faces

RANSAC attempts to improve the originally computed alignment between the two surfaces by re-computing the rotation and translation from all non-outlying data points. The outlying data points are eliminated by having a distance greater than some threshold from all other data points. For our purposes, this means that we need to eliminate surface faces which are a great distance from any face on the other surface and then to re-compute the rotation and translation from all (accepted) faces using some correspondence between the faces. However, no correspondence information exists between faces and so the simple approach of finding the closest face (as measured by Euclidean distance) is used to determine the correspondences.

This process can be made very efficient by a pre-processing step applied to one of the two surfaces. Recall that the current surface may be thought of as a *dynamic surface* and the database surface as a *static surface*. Since coordinates of the current surface are constantly changing, it is hard to perform any pre-processing for it. However, the coordinates of the database surface are used as a static reference, and thus a hash structure can be created in a pre-processing step to hold the coordinates of the faces within the database surface and this structure can be used to efficiently access faces based on coordinate.

The pre-processing step constructs a 3D equally spaced grid and assigns each face of the static surface to one or more of the grid cells, based on the position of the face relative to the location of the grid. Once constructed, each cell within the grid will contain a list of face IDs and possibly their locations in (x,y,z) coordinate space. The grid is then used when the closest static surface face to each of the dynamic surface faces is needed. For each face within the dynamic surface, calculate the cell position, based on the location of the face, and search through the small list of faces within that cell in order to find the closest corresponding face.

In practice, the closest face could be in the directly mapped cell or in any of the immediately adjacent cells, and so each face within the static surface is assigned to the cell it directly maps to and all $3 \times 3 \times 3 - 1 = 26$ adjacent cells.

For each face within the dynamic surface, the closest face within the static surface can now be found within a single grid cell. Compute the index of the cell which maps to the coordinate of the face from the dynamic surface, then search through those faces within that single cell and select the face closest to the dynamic surface face. In some cases, the current surface face will map to a cell which contains no face indices, in which case that face is immediately excluded from the consensus set.

The size of the cells, denoted by the variable *cellSize*, determines the efficiency of this approach. With *cellSize* too small, the memory required to store this structure becomes too large. With *cellSize* too large, the number of faces within each cell becomes too many and the processing requirements for mRANSAC become prohibitively large. Care must be taken to choose this value correctly.

A further effect is that *cellSize* determines a maximum distance threshold, similar to *maxDistThreshold*. Each grid cell is a cube with sides equal to *cellSize*. Each cell can contain static surface faces which map to that cell and faces which map to any of its adjacent cells. Therefore, the maximum distance possible between faces which map to within a cell is three times the diagonal distance within a cube of sides *cellSize*, specifically $3\sqrt{3} \times$ *cellSize*. Setting *maxDistThreshold* to any value greater than $3\sqrt{3} \times$ *cellSize* will not accept any further faces as part of the consensus set because they will not be located.

This algorithm for closest face selection is incorporated into the mRANSAC algorithm by constructing a grid with cell size equal to $cellSize_1$ (the 1 is added to make this value distinct from another cell size which is introduced below). The ideal value for $cellSize_1$ is unclear. The grid cells must be large enough to enable Phase Two to find sufficient correspondences in order to improve the registration on that achieved using Phase One. Therefore, $cellSize_1$ affects the maximum registration error that can be corrected by Phase Two. Experiments in Chapter 6 will address this issue by examining the performance using different values for $cellSize_1$.

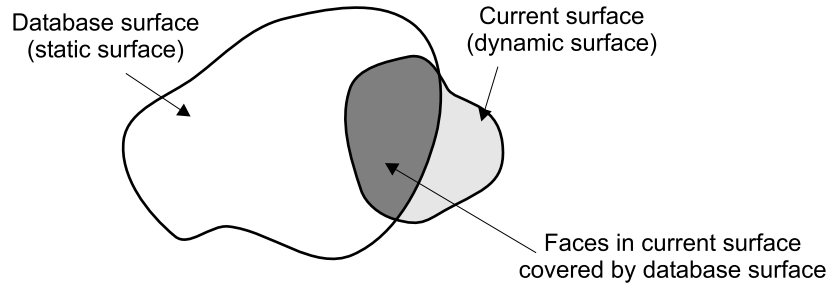


Figure 5.6: The coverage of the current surface
Coverage measures the percentage of faces within the current model which are *covered* by any faces within the database model.

5.2.6 Calculating Surface Fit Error

The last step within each iteration of the RANSAC process is to determine how closely the two surfaces have been registered. The simplest approach is to calculate the root-mean-square (RMS) distance between corresponding faces within the two surfaces. However, again the correspondences between faces in the current and database surfaces are required. The same process as described above can be used to find these correspondences, however there is a further optimisation to be made.

Let us say that we want to calculate the surface fit error when fitting the dynamic surface (the current surface) onto the static surface (the database surface). Thus, for each face within the dynamic surface, we want to know the distance to the closest face within the static surface. But using the closest face is only an approximation to finding an accurate face-to-face correspondence. Many faces in the static surface will be matched to from more than one dynamic surface face. Reducing this to a one-to-one relationship would be computationally very expensive.

A further problem with this approach has not been addressed: how do the faces with no correspondence affect the error measure? An RMS distance between corresponding points does not give easily to measuring the number of points without correspondences.

A much easier and efficient solution is to approximate this process by considering that the exact face-to-face distances are not required. An approximation would be to know only that a corresponding face exists within some maximum distance. Any face-to-face distances greater than this maximum distance can be discarded and considered to be caused by either obstructions or bad registration. We can calculate this very efficiently by using the grid structure devised previously and checking to see whether any entries exist without finding the very closest face. If an entry exists within the mapped grid cell then it can be interpreted that *some* corresponding face exists within the maximum distance. For a grid with some specified *cellSize*, this maximum distance is determined to be $3\sqrt{3} \times cellSize$.

The final registration measure determines the accuracy of the registration, rather than the registration error. The registration accuracy is measured by computing

the percentage of dynamic surface faces which have a correspondence within this maximum distance. This is called the *coverage*.

Put in terms of the current and database surfaces, the coverage is the percentage of current surface faces which have corresponding faces within the database surface, as determined by finding a non-empty cell within the grid. Coverage is illustrated in Figure 5.6.

A coverage of around 0% indicates that the two surfaces are very poorly registered and probably do not even intersect. A coverage of 100% indicates that all faces within the dynamic surface have some corresponding face existing within the static surface. The coverage measure indicates nothing about the percentage of faces within the static surface which have correspondences, however, being the surface selected from the database, we are not concerned with how much of it is covered.

The coverage measure is incorporated into mRANSAC by finding the pose which maximises the coverage. The grid used for calculation of coverage may need to use a different cell size than that used for finding the closest faces. The cell size here determines how close faces in the surfaces need to be in order to increase the measure of coverage. In other words, it determines the error tolerance. The mRANSAC algorithm can use two separate grids, one with cell size equal to $cellSize_1$ for the purposes of closest face finding, and another with cell size equal to $cellSize_2$ for calculating coverage. In the case that $cellSize_1 \neq cellSize_2$, two grids must be constructed. The second grid, for calculation of coverage, need only store a binary value for each cell, indicating the presence of static surface faces. In this case, the second grid is a typical voxel representation and memory saving optimisations can be used, provided that efficient calculation of coverage is still permitted.

The mobile robot's considered for this thesis navigate structured indoor environments and so a $cellSize_2$ of 20 mm may be a suitable value; it allows a maximum face-to-face distance of $3\sqrt{3} \times 20 = 104$ mm.

5.2.7 Efficiency Improvements

In practice, finding the closest face within the mapped cell is still very time consuming and increases exponentially with the size of the grid cells so that a suitably small grid cell must be used. But the relative pose cannot be updated without using the coordinates of the faces from the current surface and some coordinates representative of the database surface. Three alternative approaches for selecting database surface coordinates were attempted and compared to the existing solution (referred to as "Closest Face Selection"). The alternative solutions are:

- **Random Face Selection:** random selection of face within selected grid cell.
- **Average Face Coordinate:** average face coordinate per grid cell
- **Grid Cell Centre:** coordinate of centre of grid cell

The first alternative uses the same grid construction, however it randomly selects any face out of the list of faces assigned to the mapped grid cell, rather than considering all faces. The effect of this on overall accuracy can be considered as the effect of averaging out noise. Although a single selected face may be much further than the true closest face, when the error is averaged over all faces used the overall error is much less. Further to this, it is possible that the simplest approach of accurately picking the very closest face may be more subject to noise than this approach.

The second alternative modifies the pre-processing step constructing the grid. Instead of producing a grid of face ID lists, each cell contains only a single 3D coordinate. This coordinate is calculated by averaging the coordinates of all faces which would have been assigned to that cell in the original approach. This approach relies on a similar process to the previous alternative but implements it in a different manner by performing the averaging as a pre-processing step and by averaging *within* each grid cell, rather than averaging *across* all cells accessed.

The final alternative performs a pre-processing step whereby the coordinate at the centre of each grid cell is assigned directly to the grid cells, without considering any faces within the static surface. This alternative is considered for the purposes of setting a worst-case benchmark. This is the most simplistic solution, but should also yield very bad results compared to the original approach and the first two alternatives. However, if the overall error after applying this approach is not considerably worse than when applying the other approaches, then it is not worth using the other approaches as they require considerably more computation time than this simplistic approach.

In each of these alternatives, for each face within the dynamic surface, only a single face or a single coordinate from the grid is considered, rather than considering a list of faces, thus efficiency is gained by avoiding the calculation of distances to other faces within the cell. We shall examine the effect on accuracy within Chapter 6, and also illustrate the actual gain in computation time.

5.2.8 Patch Elimination

The mRANSAC algorithm relies on randomly selecting correctly matched patches in the two surfaces, however not all surfaces may be correctly matched during the tentative feature matching stage. This means that the mRANSAC algorithm may become inefficient if a large number of current surface patches are incorrectly matched to database surface patches.

It may be possible to devise a heuristic which selects those patches which are most likely to be correctly matched and eliminates those patches which are most likely to be incorrectly matched. A pre-processing step can be performed, prior to applying the mRANSAC algorithm which eliminates patches based on this heuristic. Two simple possibilities are to (1) eliminate the very small patches, or (2) eliminate

patches which have been matched with low certainty. The decomposition process produces many small patches due to noise and correspondences for these patches are less likely to be correct than for large patches because they have less structure than the larger patches. The second possible heuristic stems from the tentative matching process, which uses Equation (5.1) for the *patchMatchError* function. The current surface patches which were matched with database surface patches with a large *patchMatchError* should represent uncertain matches.

Chapter 6 investigates the effectiveness of these patch elimination methods.

5.3 Registration Method Two – DPSA

DPSA, "Decomposition and Point Set Alignment", treats the decomposed patches as providing a set of points. Two point sets, one extracted from the current surface and one extracted from the database surface, have no inherent correspondence. Gold *et al.*'s point set matching algorithm [35], provides a means of simultaneously calculating the relative pose of the points and the correspondences between points in the two sets by aligning one set onto the other in such a way as to minimise the average distance between corresponding points. This forms the basis of the DPSA approach to surface registration.

DPSA follows the same general outline as DMARA. A current surface is captured and provided as input the DPSA registration algorithm, along with a single database surface. DPSA attempts to register the surfaces as best possible and returns a rotation and translation and some error measure of how well the surfaces are aligned. Decomposition is used to extract features within the surfaces, resulting in a list of patches for each surface. The centres of these patches are used as a point set for each surface. An extended version of Gold *et al.*'s point set alignment method is used which performs the same calculation ten times and picks the best result. The reasons for applying the algorithm more than once and the issues involved are described below.

5.3.1 Decomposition and Feature Extraction

Decomposition is performed exactly as for DMARA, using the Watershed Decomposition algorithm as described in Chapter 4 and using the modifications described earlier in this chapter (see Section 5.2.1).

Gold *et al.*'s method requires point sets as its input. The simplest method for selecting points from the decomposed regions is to use the centre points of each patch (computed by taking the average of the coordinates of the faces within each patch). An alternative is to use multiple points from each patch. However, the execution time of Gold *et al.*'s method increases with the square of the number of points (specifically, it's $O(lm)$ for l and m points in two data sets, respectively).

Consequently, picking a single point for each patch seems to be a likely choice for an upper limit on the number of points which can be used for registration purposes.

5.3.2 Extension to Gold *et al.*'s Method

By applying Gold *et al.*'s algorithm to simulated data sets, it was found that it occasionally found local minima solutions, localising onto a possible registration which was neither the best solution nor the correct solution. Gold *et al.*'s algorithm uses an initial guess at a rotation matrix, $\mathbf{R}$. Algorithm 5 (page 56) uses the identity matrix, $\mathbf{I}_3$, as an initial value for $\mathbf{R}$. It was found that a good solution to the problem of local minima was to apply the algorithm a number of times from different randomly selected initial rotation matrices.

The majority of capture devises on robots are calibrated in such a way that the vertical alignment and registration of surfaces is already done fairly accurately. Under normal circumstances, the current surface may be rotated and translated along the horizontal plane parallel to the floor. Rotations thus constitute mostly yaw (rotations around the vertical axis), with only small amounts of rotations around the other axes due to instability in the physical mounting and calibration of the capture device. Consequently, random initial rotations can be limited to being selected from 360° around the vertical axis, with a very small deviation from horizontal.

10 such attempts of the algorithm from different initial rotations provides a suitable increase in accuracy without considerably extending the computation time.

The best out of the attempts must be automatically selected. The output from Gold *et al.*'s algorithm gives the correspondence information between some or all of the points. The number of points which have correspondences and the distances between these points and their corresponding points is combined in a registration fitness measure called the *correspondence fitness*.

The correspondence fitness, referred to by the variable *corrFit*, is calculated using Equation (5.4) after registration of two point sets with l and m points, respectively. $\{D_i\}$ is a set of $(n \leq \min(l, m))$ point-to-point distances. These distances are computed by calculating the Euclidean distances between the n corresponding points, as produced from Gold *et al.*'s method.

$$corrFit = \sum_{i=1}^{n} distanceFitness(D_i) \tag{5.4}$$

The function *distanceFitness* measures the fitness of a single point pair distance. This is a value in the range $[0, 1]$, where 1 indicates a perfect point-to-point alignment (zero distance), and 0 indicates a very large point-to-point distance. Intuitively, *corrFit* gives a weighted count of the number of points which have correspondences, so that points which are far apart receive little weight, while points which are close to being exactly aligned receive close to one as their weight.

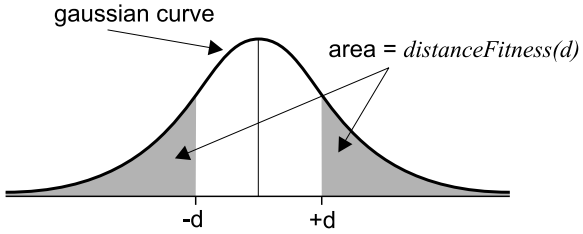


Figure 5.7: Measuring distance fitness on gaussian curve
The *distanceFitness* for distance d measures the area under the gaussian curve on either side of the $(-d, +d)$ range.

Equation (5.5) gives the calculation for *distanceFitness* of a single point-to-point distance, d. It is the area under the curve of the gaussian distribution from $+d$ to positive infinity, and from $-d$ to negative infinity, is indicated by Figure 5.7. A well known statistics result says that the values of differences between points sampled from gaussian (normal) distributions are themselves sampled from a gaussian distribution. The process which obtains the coordinates of the patch centres has an inherent noise, or error. Assuming that this process produces noise with a gaussian distribution, the distance between points can be modelled by a gaussian distribution. Equation (5.5) measures the accuracy of a single point-pair alignment, given the expected level of noise inherent within the calculation of patch centre coordinates, and specified by σ_{pn} (the standard deviation of point sample noise).

$$distanceFitness(d) = 1 - \frac{1}{\sigma_{pn}\sqrt{2\pi}} \int_{-d}^{d} \exp\left(\frac{-x^2}{2{\sigma_{pn}}^2}\right) dx \tag{5.5}$$

Of 10 attempts, the one which gives the greatest correspondence fitness is selected as the final result.

5.3.3 Effectiveness of Correspondence Fitness

To test the effectiveness of this method to accurately detect the best of 10 attempts, an experiment is performed using simulated data. Two 3D point sets, $\{\vec{X}\}$ and $\{\vec{Y}\}$, are created with 50 and 30 points, respectively. The coordinates of the first 20 points within $\{\vec{X}\}$ are randomly generated using a uniform distribution over the range $[-1, 1]$. The coordinates of the first 20 points within $\{\vec{Y}\}$ are assigned by using the first 20 points in $\{\vec{X}\}$ and adding gaussian distributed noise with standard deviation of 0.2 (mean 0.0). The coordinates of the remaining 30 points in $\{\vec{X}\}$ and 10 points in $\{\vec{Y}\}$ are independently generated using a uniform distribution over the range $[-1, 1]$. Finally, all points within $\{\vec{X}\}$ are rotated about the origin by 20° around the vertical axis.

In this way, 20 points are produced with known correspondences, a noise with standard deviation of 0.2, and a known rotation of 20°. Additionally 30 and 10 outlying points are created within the two data sets. Gold *et al.*'s method is then

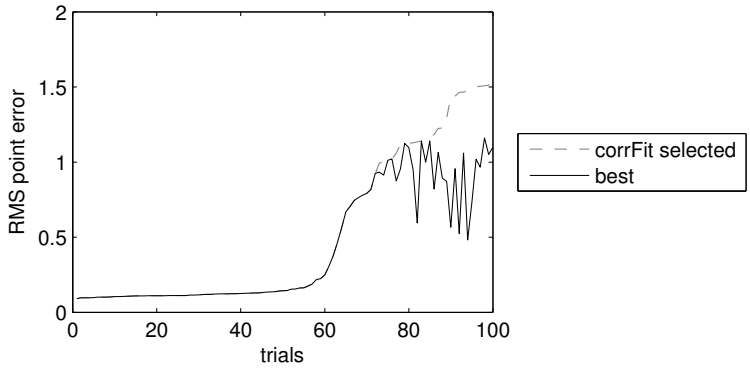


Figure 5.8: Effectiveness of correspondence fitness
All trials with best *rmsError* less than 0.45 have *corrFit* selected *rmsError* equal to best *rmsError*.

applied to attempt to align the point sets by finding the correct rotation ($\mathbf{R}$) and translation ($\vec{T}$) to transform the points in $\{\vec{Y}\}$ so that they are aligned to points in $\{\vec{X}\}$, and to find the correct point correspondences.

The success of Gold *et al.*'s method is measured by calculating the RMS of the Euclidean distance between the first 20 points after its resulting rotation and translation has been applied to. Specifically, the RMS error is measured as:

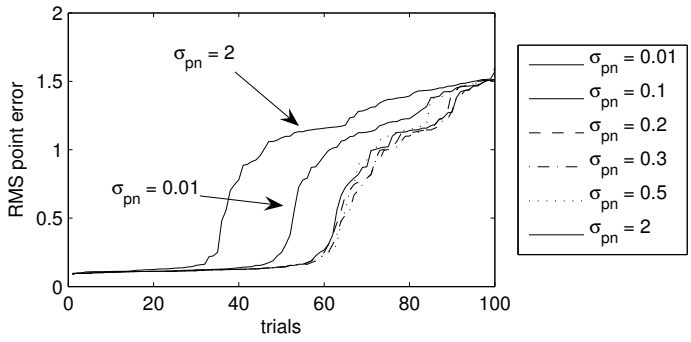
$$rmsError = \sqrt{\frac{1}{20}\sum_{i=1}^{20} \|X_i - (\mathbf{R}Y_i + \vec{T})\|^2}$$

where $\|\cdot\|$ is the Euclidean distance operator.

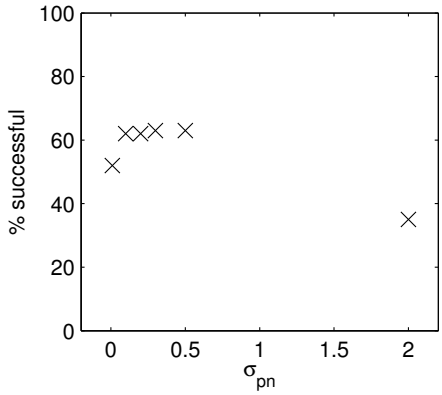
100 trials are performed. Each trial runs Gold *et al.*'s method for 10 attempts, using *corrFit* to select the best attempt (the attempt having the greatest value for *corrFit*) out of 10. The actual standard deviation for the point noise is 0.2 and so $\sigma_{pn} = 0.2$ for this experiment. Also for each trial, the *rmsError* for rotation and translation results of each attempt is calculated. The best overall *rmsError* is selected regardless of *corrFit*, and the *rmsError* for the attempt selected with the greatest *corrFit* is also selected.

Figure 5.8 shows the "best" *rmsError* (least overall *rmsError* of the 10 attempts) and the *rmsError* selected using *corrFit*, for all 100 trials. The solid line shows the "best" *rmsError*, and the dashed line shows the *rmsError* selected using *corrFit*. The results for the trials have been sorted in ascending order of *corrFit* selected *rmsError* for sake of clarity.

The data has been generated with a noise having standard deviation 0.2. The expected *rmsError* for an ideal point set alignment is thus around 0.2 (technically, it is $0.2\sqrt{3} = 0.35$). Any trials resulting in best *rmsError* greater than about 0.45 can be considered to have failed: none of the 10 attempts achieved a registration with *rmsError* less than 0.45. As can be seen, the two curves follow each for all



(a)



(b)

Figure 5.9: Effect of σ_{pn} on success rate

(a) RMS point error for each of 100 trials using different σ_{pn}. Each curve is independently sorted in ascending order for sake of clarity.
(b) percentage of experiments giving RMS point error no greater than 0.45 for each of the six different values for σ_{pn}.

cases that the best *rmsError* is less than 0.45. This indicates that, so long as at least one of the 10 attempts was successful, the correspondence fitness method is 100% successful at detecting that best solution, with the parameters used in this experiment.

5.3.4 Determining Sample Point Noise

The experiment in the previous section used simulated noise with a known standard deviation, and thus the value for σ_{pn} was easy to choose. But how is σ_{pn} chosen for arbitrary data when the level of noise is unknown, and how important is it to chose σ_{pn} correctly?

Figure 5.9(a) and (b) shows the effects of choosing different values for σ_{pn} in the experiment described above. Figure 5.9(a) shows the *rmsError* values for each of 100

σ_{pn}	*corrFit* values (5 of 10 attempts)					min	max	diff.
0.02	0.062	0.000	0.128	0.062	0.062	0.000	0.128	0.128
0.05	0.790	0.129	0.660	0.790	0.790	0.129	0.790	0.661
0.10	2.883	1.445	1.801	2.883	2.883	1.445	2.883	1.438
0.20	7.431	6.066	5.657	7.431	7.431	5.657	7.431	1.774
0.30	10.991	9.832	9.439	10.991	10.991	9.439	10.991	1.552
0.40	13.425	12.353	12.143	13.425	13.425	12.143	13.425	1.282
0.50	15.103	14.069	14.031	15.103	15.103	14.031	15.103	1.072

Table 5.2: Effects of σ_{pn} on *corrFit*

corrFit values using different σ_{pn} for each of 5 out of 10 attempts of Gold *et al.*'s method, minimum and maximum values and the difference between the minimum and maximum values. Data generated using gaussian noise with standard deviation of 0.2.

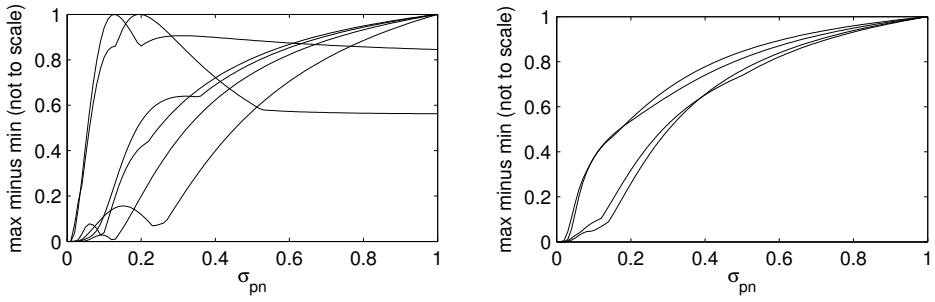
trials using five different values for σ_{pn}. Notice that three of the curves are tightly bunched together. Figure 5.9(b) gives the percentage of trials achieving *rmsError* no worse than 0.45. For σ_{pn} of 0.1, 0.2, 0.3 and 0.5, the extended Gold *et al.*'s method succeeded in about 65% of the trials by achieving a *rmsError* less than 0.45. However, σ_{pn} of 0.01 and 2.0 both cause the extended Gold *et al.*'s method to succeed less often.

It can be seen that σ_{pn} does not have to be known exactly, rather it needs to be set to within the vicinity of the correct value. Values of σ_{pn} which are much too small or much too large cause the correspondence fitness calculation to become increasingly inaccurate, resulting in a lower success rate for the extended version of Gold *et al.*'s method.

A further result enables us to automatically select σ_{pn}, without having to know *a priori* the level of noise.

The value of σ_{pn} acts like setting a soft threshold. Setting σ_{pn} to a very small value discards most point pair distances by setting their weights close to zero. Setting σ_{pn} to a very large value accepts most distances by setting most weights close to one. After 10 attempts of the Gold point matching algorithm have been applied with random initial starting rotations, some will likely produce bad registrations, and some (though it can be none) will produce valid registrations. If σ_{pn} is chosen well, the difference between the least correspondence fitness of the 10 attempts, and the greatest fitness, will be large. As σ_{pn} is chosen to be smaller, more point pairs will be given weights closer to zero, and the difference between the least and the greatest correspondence fit will diminish. As σ_{pn} is increased beyond the optimal value, more point pair distances will be given significant values and again the difference between least and greatest fitness will diminish from the optimal.

This effect can be seen in Table 5.2. Each row shows the *corrFit* values after five attempts of applying Gold *et al.*'s method to the same point sets as used in the experiment above. The last three columns show the minimum *corrFit*, the maximum *corrFit*, and the difference between the maximum and minimum. Each



(a) (b)

Figure 5.10: Max-Min CorrFit difference
Shows the difference between min. and max. values of CorrFit for different σ_{pn}.
(a) Some typical difference curves when at least one of 10 attempts is successful.
(b) Some typical difference curves when none of 10 attempts are successful.
Note: for purposes of highlighting the shape of the curves, the curves are re-scaled on the vertical axis so that their maximum values are all the same.

row represents the *corrFit* values for different values of σ_{pn}. A σ_{pn} of 0.2 yields the greatest difference between maximum and minimum. 0.2 is the correct value!

Figure 5.10(a) shows in more detail how the value of σ_{pn} affects the difference between least and greatest correspondence fitness. Each curve represents the difference of least and greatest correspondence fitness after 10 attempts of Gold *et al.*'s method on a single pair of point sets. The curve is produced be recalculating *corrFit* using different values for σ_{pn} and using the same corresponding point-pair distances for each value of σ_{pn} (that is, without re-running Gold *et al.*'s method). Six such curves can be seen. The actual curves are all shown at different scales because their maximum values vary a lot, and so they are re-scaled individually to fill the vertical axis of the plot, making it easier to observe their shape. Each of the curves are for trials where at least one of the 10 attempts was successful. Figure 5.10(b) gives more examples of such curves, except that these curves are produced from trials where Gold *et al.*'s method did not succeed in any of the 10 attempts.

It can be seen that the curves in Figure 5.10(a) all have a local maximum point around the σ_{pn} value of 0.2. In some cases the value is slightly larger or slightly smaller, but by no more than about 0.1. This follows the observation shown in Table 5.2. However, a further characteristic can be noticed: many of the curves have a second maximum point or even rise asymptotically beyond the initial local maximum point. The second maximum point is due to the variability of the output from Gold *et al.*'s method. For the same point sets and input parameters, Gold *et al.*'s method will detect one number of corresponding points in one instance, and a different number in another. As σ_{pn} becomes large, *corrFit* converges to simply counting the number of corresponding points. Consequently, the difference between maximum and minimum will converge to the difference in number of correspondences found

on different attempts of Gold *et al.*'s method.

The curves of Figure 5.10(b) indicate that the first local maxima point does not exist when none of the 10 attempts was successful.

This result enables us to automatically select a (close to) correct value for σ_{pn} by considering how it affects the values of the correspondence fitness as already mentioned. A gradient ascent approach could be used to find the optimal value, being careful to not overstep the local peak and continue towards the horizontal asymptote.

In practice, σ_{pn} does not have to be known exactly and it is easier to test the min-max differences using a pre-set list of possible values, using some predefined sampling resolution, and to pick the best σ_{pn} from that list, rather than to implement a gradient ascent algorithm. Thus it helps to know some approximate idea of the level of noise *a priori*. A discretised search is then performed around this value.

Notice that implementations need only store a list of the corresponding point pair distances to calculate the correspondence fitnesses at each attempted value of σ_{pn}. The original point position data does not have to be stored, nor do the point-to-point distances have to be recalculated for each value of σ_{pn}.

5.3.5 Registration and Recognition using the Extended version of Gold *et al.*'s Method

Registration of the current surface onto a database surface is performed using the extension of Gold *et al.*'s method. 10 attempts are performed from different random initial starting rotations and *corrFit*, together with automatic σ_{pn} selection, is used to determine the best attempt.

After surface registration, surface recognition must compare the accuracy of registration of the current surface onto each of many database surfaces and select the best match. For this purpose, *corrFit* is unsuitable because it depends on the value of σ_{pn}, which may be different for each registration result.

For purposes of comparing the registration accuracy of the current surface onto each of the database surfaces, the *coverage* measure developed for DMARA may be used. In this case, the database of previous locations stores a voxel representation of each surface along with the polygon mesh representation and decomposition information. Computation of coverage is then efficient to perform each time any current surface is registered onto a database surface.

Chapter 6

Experimental Results

Chapter 1 gave three questions which this thesis tries to address. The first question asked how decomposition can be used for feature extraction. That question is largely addressed within the previous chapter. This chapter gives a "Proof of Theory" for the decomposition approach by offering answers to the second and third research questions. Specifically, "what success rate can be achieved for registration using decomposition for features?" and, "how efficient is registration based on decomposition?". The experiments within this chapter, and the analysis within the following chapter, serve to give an introductory examination of these questions using the new recognition methods described in this thesis. The results presented here are indicative of the possible usefulness of this approach, however, they are not conclusive. But the results that follow offer some interesting insights into the possibilities and lead naturally towards research in this exciting area.

This chapter presents experiments measuring the *effectiveness* and the *efficiency* of the two new recognition methods, DMARA ("Decomposition, Match Assignment, and RANSAC Alignment") and DPSA ("Decomposition and Point Set Alignment"). The full task of surface recognition is outside the scope of this thesis and, instead, the specific task of registering two surfaces is focussed on. The effectiveness is measured in terms of the *registration accuracy*, or in other words, how well the methods can register two surfaces. The efficiency is measured in terms of the computational complexity of the algorithms. To aid the reader in understanding the efficiency of these algorithms, measurements of actual computation times are also given.

The body of this chapter is organised as follows. Firstly, the data set is described, giving various information regarding its dimensions, its resolution, and the level of noise present. The results of decomposition are also given. This decomposition serves as the input for all tests. Secondly, two measures of registration accuracy are defined and some concrete goals set in order to measure the "successfulness" of the methods to register surfaces. Thirdly, various experiments are performed to measure the effectiveness of DMARA and DPSA. Finally, the efficiency of the algorithms is examined.

Figure 6.1: Data set surface

6.1 3D Model Data Set

One original surface is used for the experiments. Two copies of this surface are used, having different levels of noise. The raw surface from the 3D capture device contains a high level of noise. Two version of this are thus made by smoothing by different amounts. The decomposition process, being highly affected by noise, produces sufficiently different decompositions of either version for valid testing of patch matching *et cetera*. An advantage of using the same original surface is that the exact face-to-face correspondence is known *a priori*. This makes it easy to calculate the error in registration after applying the registration methods. The detail of how the surfaces are smoothed is not important and so is excluded. It is sufficient to say that an edge preserving smoothing technique is used. For an example of one such edge preserving technique see the paper by Gregor and Whitaker [37], although this is not the approach used here.

An optically based range imager was used for scene capture, using the system developed by Carnegie *et al.* [17]. This produced a 2D matrix of values, where each entry represents the distance, in metres, from the focal point of the camera to a point on the surface being observed. The raw data contains projective distortion which is corrected for from known focal length.

The raw data also contains noise inherent within the capture process. The measurement noise affects the distance measurements (Z-axis) within the raw data and creates a small measurement error within the X- and Y-axis once the projective distortion is corrected for.

The captured range data also contains measurement artefacts at the edges of the captured region. Most of this is removed by a filtering process, but some remains. In

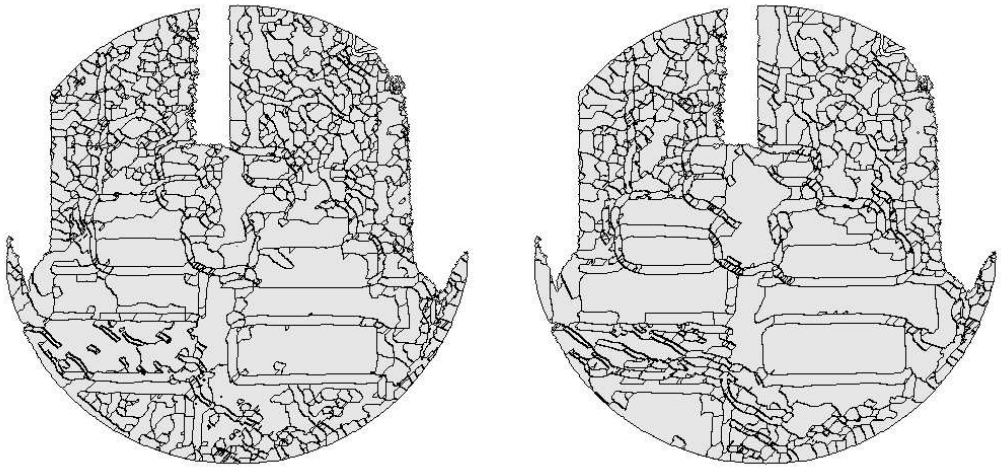
	NoisyScene	*SmoothScene*
Noise: (RMS error)	5.311 mm	4.835 mm
Dimensions:		
width	1.40 m	1.39 m
height	1.09 m	1.09 m
depth	7.37 m	4.95 m
# Faces	328988	328988
# Vertices	165698	165698
Face Areas:		
min	0.748 mm^2	0.720 mm^2
max	4450 mm^2	1360 mm^2
mean	22.0 mm^2	21.0 mm^2
median	5.80 mm^2	4.51 mm^2
Face Sides:		
min	1.18 mm	1.19 mm
max	3560 mm	1290 mm
mean	15.0 mm	14.1 mm
median	4.21 mm	3.45 mm

Table 6.1: Data Set

particular, a large noisy region exists in the upper half of the range image. Figure 6.1 shows a rendering of the noisy version of the surface data. The surface is of several blocks of different heights positioned at different distances from the ranger. At the time of capturing this surface, the ranger was still in early prototype stage, and consequently two large artefacts cover the upper half of the image shown.

The noisy surface data is referred to as *NoisyScene*, and the smoothed version as *SmoothScene*. Both are represented as triangular polygon meshes. Table 6.1 summarises the data set by providing useful statistics about the surfaces and the faces which make them up. The noise is measured by fitting a plane to the facing region of block at the bottom right of the surface, and measuring the root-mean-square (RMS) error between the plane and the surface data. The dimensions data gives the range of the physical coordinates of the surfaces. Face areas are calculated by determining the physical area covered by each face, and face sides measure the average lengths of the sides of the faces. The large maximum face side length and face area are due to the artefacts within the data set.

The limitations of using only a single surface for testing are recognised. For this reason, results are understood to be indicative only, and any conclusions drawn consider that some characteristics may be due to the data being tested on this occasion.



(a) (b)

Figure 6.2: Patch boundaries after decomposition
(a) *NoisyScene*, (b) *SmoothScene*

DataSet	Decomposition (without co-planar merging)	Final Decomposition
NoisyScene	6227	1386
SmoothScene	4557	1143

Table 6.2: Effects of co-planar patch merging
The number of patches produced from the initial decomposition process, and after applying the co-planar patch merging

6.1.1 Decomposition Results

A single decomposition of the *NoisyScene* and *SmoothScene* are performed, and adjacent patches which are close to co-planar are merged. This decomposition result is used as input to most of the experiments within the chapter. Figure 6.2 shows the boundaries of the decomposition of the two surfaces. It can be seen that the decomposition process results in different sizes of patches, with different boundaries in each case. Also, some patches are similar in each patch, but have notches in the patches in one surface but not in the other.

Table 6.2 lists the number of patches produced after the initial decomposition process and after merging close to co-planar patches. The co-planar merging phase combines adjacent patches by fitting planes to each patch, calculating the plane normals and merging the planes if they are at in similar positions and the plane normals are similar. This helps to alleviate the noise sensitivity of the decomposition process.

6.2 Measures of Success

The registration effectiveness of DMARA and DPSA is determined by using their rotation and translation results to attempt to align the two surfaces and then measuring the registration error. The registration error is measured in two ways: (i) the RMS error between faces, and (ii) the rotation and translation error. Typically the experiments will treat *NoisyScene* as the current surface, and *SmoothScene* as the database surface, and so the registration methods attempt to find the transformation which registers *NoisyScene* onto *SmoothScene*. This section assumes that order, however the error measures defined here are symmetric.

The RMS error is easy to measure because the correspondences between faces is known. Specifically, every i^{th} face of *NoisyScene* directly corresponds to the i^{th} face of *SmoothScene*. Therefore, the RMS error is simply the root-mean-square of the distances between all corresponding faces.

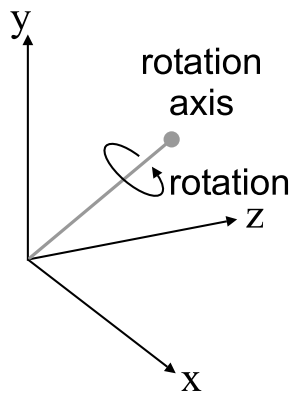


Figure 6.3: Rotation axis and rotation angle

The rotation and translation error are computed by examining the matrix representing the rotational part of the alignment, and the relative distance between the centres of the surfaces, respectively. The rotation matrix represents the rotation of the current surface. The rotation matrix can alternatively be represented as a single "axis of rotation" and a rotation angle, as shown in Figure 6.3. Thus, rotation error is measured by converting the matrix to the representation of rotation axis and angle, and the relative angle between the current surface and the database surface gives the rotation error. Formerly, if the rotation matrix, $\mathbf{R}$ is given by:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

then the rotation angle is given by:

$$\alpha = cos^{-1}\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right)$$

The translation error is calculated by calculating the Euclidean distance between the centre of the current surface and the centre of the database surface after registration.

6.2.1 Benchmark Goals

In order to provide a benchmark to compare experimental results, Table 6.3(a) presents the best possible outcomes for registration of *NoisyScene* onto *SmoothScene* (and, in fact, *vice versa*). The RMS error is non-zero because of the difference in noise between the two surfaces. In a complete system, the results of these course registration method would be used as input for a fine registration using, for example, ICP. ICP is capable of handling only a small angular and translational error. Therefore, we arbitrarily define a "successful" registration as one which results in no greater than 30° rotation error and 0.1 m translation error. Based on the best possible RMS error, we also define a "successful" registration as one which results in RMS error no greater than 0.2 m, by approximately doubling the best possible RMS error (as it turns out, this compares quite closely with success as defined by rotation and translation error). These *benchmark goals* are listed in Table 6.3(b).

Best possible results	
RMS error	0.0990 m
rotation error	0.1°
translation error	0.008 m

(a)

Benchmark goals	
RMS error	0.2 m
rotation error	30°
translation error	0.1 m

(b)

Table 6.3: Best possible results and benchmark goals

6.3 Registration Method One – DMARA

The general RANSAC algorithm is a statistical approach to outlier removal, requiring many iterations of the same process. As such, the form modified for registration, mRANSAC, must be given a certain minimum number of iterations in order to have a certain probability of success. The iterations of mRANSAC are independent, and so it can be said that a single iteration has a certain probability of successfully finding the correct registration. A result from statistics says that, after n trials, where each trial has p probability of success, the probability of succeeding at least once is given by:

$$P_{\text{success}} = 1 - (1 - p)^n \tag{6.1}$$

In other words, if p is the probability of an individual iteration of mRANSAC resulting in a successful registration, the probability of succeeding after n iterations is given by the equation above. By rearranging this, the number of iterations can be

Patch distributions:	
# samples (N)	10 000
bin width (B_{width})	5 mm

Table 6.4: Parameters used in tentative patch matching experiment

calculated in order to achieve a certain desired probability of success:

$$n = \left\lceil \frac{\log(1 - P_{\text{success}})}{\log(1 - p)} \right\rceil \tag{6.2}$$

For this reason, it is important for the success of DMARA that each individual iteration of mRANSAC has a high chance of success. The experiments that follow examine DMARA from this point of view.

6.3.1 Tentative Patch Matching

The mRANSAC algorithm, as applied in the way presented here, requires an initial guess at the correspondence between patches of two surface models being registered. The goal of the patch matching process is to find the patch correspondences with as high a degree of accuracy as possible.

Remember that the patch matching and registration process is always performed from a *dynamic surface* (the 3D model most recently captured) onto a *static surface* (a 3D model chosen from the database). As a consequence, the patch matching process within the main algorithm is inherently one-sided, while the tests described here are symmetric and examined in both directions in order to indicate the level of variation possible within results.

The patch matching accuracy is now examined by comparing to a ground truth list of patch correspondences. The experimental set up, calculation of ground truth, and measurements of accuracy are described below.

Experimental Setup

In this test, two surface models, model A and model B, are decomposed, and the best patch matches calculated using the method described in Section 5.2.2. Patch matching is performed from *NoisyScene* patches to *SmoothScene* patches, and in the reverse direction.

A *ground truth correspondence* is calculated, giving the ideal matching from patches in model A to patches in model B. The effectiveness of the patch matching algorithm is calculated as the number of *correct* matches divided by the total number of patches in model A, or in other words, the number of patches in model A which are matched to the same model B patches as in the ground truth match.

The parameters for the Shape Distribution calculations were found by trial and error and are listed in Table 6.4.

Ground Truth Calculation

The calculation of the ground truth is done by overlaying one of the decomposed model surfaces onto the other and examining how patches overlap. The calculation takes two sets of patches, $\{U_j\}$ and $\{V_k\}$, from model A and model B respectively, each entry representing a single patch by specifying the faces within that patch. Then, firstly, the vector $\vec{u} = (u_1, \ldots, u_J)$ is computed using Equation (6.3) which holds the best match from model A onto model B by picking the model B patch which overlaps a model A patch the most, for each patch in model A. Secondly, Equation (6.4) is used to compute vector $\vec{v} = (v_1, \ldots, v_K)$ which holds the best match from model B patches onto model A patches in the same way as before. Lastly, the ground truth match vector, $\vec{g} = (g_1, g_2, \ldots, g_J) \in \{1, 2, \ldots, K, \epsilon\}$ (where ϵ = "unassigned / no match"), is calculated using Equation (6.5).

$$u_j = \underset{k}{\operatorname{argmax}}\ \mathit{overlap}(U_j, V_k) \tag{6.3}$$

$$v_k = \underset{j}{\operatorname{argmax}}\ \mathit{overlap}(V_k, U_j) \tag{6.4}$$

$$g_j = \begin{cases} k & \text{if } v_k = j, \text{ where } k = u_j \\ \epsilon & \text{otherwise} \end{cases} \tag{6.5}$$

$\mathit{overlap}(a, b)$ measures the amount of overlap between patches a and b. This would ideally be based on the total area of faces within the patches, however the number of overlapping faces within patches a and b is used within experiments presented here.

Equation (6.5) eliminates patches which are not consistent between matches performed from model A to model B and from model B to model A. In other words, if patch a_1 in model A matches b_1 in model B when performing a match from model A to model B, but patch b_1 is matched to patch a_2 instead of a_1, then those three patches are eliminated for being inconsistent.

By nature of this consistency constraint, the final value for vector $\vec{g}$ is the same when computed in either direction.

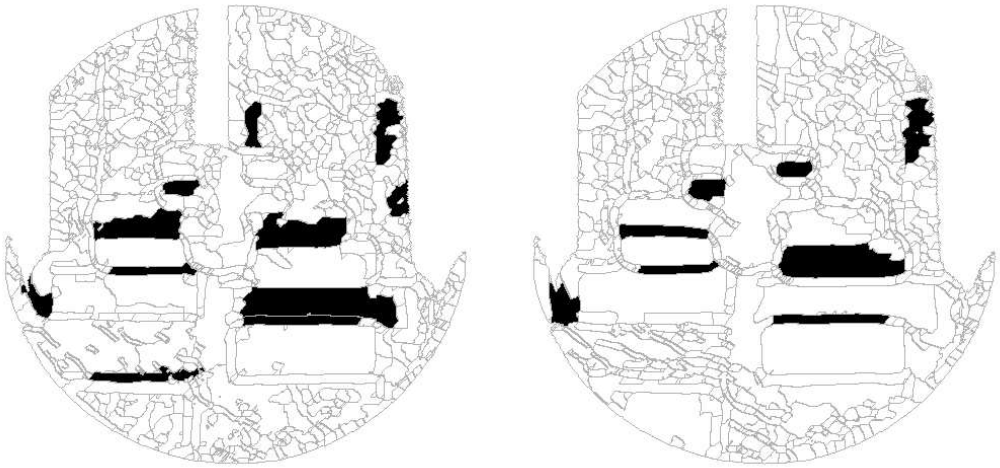
Results

Table 6.5 lists the number of ground truth matches found after applying the ground truth finding method, described above, to the decompositions.

DataSet	Number of Patches
NoisyScene	433
SmoothScene	433

Table 6.5: Number of ground-truth patch matches

The number of ground-truth matches found between *NoisyScene* and *SmoothScene*.



(a) (b)

Figure 6.4: Correctly matched patches
(a) *NoisyScene*, (b) *SmoothScene*

Table 6.6 lists the number and percentage of *NoisyScene* patches which are correctly matched to *SmoothScene* and *vice versa*. Figure 6.4 illustrates which patches are correctly matched for data set *NoisyScene*. These patches correspond to some of the patches within Figure 6.2.

The patch matching process correctly matches, on average, approximately 1.9% of the decomposed patches from the first model to the second, as Table 6.6 indicates. This is a very low percentage, making the efficiency of the RANSAC registration algorithm very low, as will be discussed later. Furthermore, not all of the 1.9% of patch pairs will be useful for calculating relative orientation because the centres of patches can vary a lot depending on the exact patch shape produced by the decomposition process. These results indicate a high importance on some form of patch elimination process to discard patches which are unlikely to be correctly matched.

Current Surface	Database Surface	Total Patches	Correct Matches
NoisyScene	*SmoothScene*	1386	25 (1.8%)
SmoothScene	*NoisyScene*	1143	22 (1.9%)

Table 6.6: Number of correctly matched patches
The total number of patches in the dataset, and the number of patches correctly matched, together with the percentage of total patches.

6.3.2 Patch Elimination

The previous experiment showed that a very low proportion of patches are correctly matched. In order to improve this proportion, two heuristics are considered which might enable us to remove patches which are unlikely to yield correct matches. We are thus looking to remove as many patches as possible which are unlikely to yield *true positive* matches, while keeping as many patches as possible which do yield *true positive* matches.

By examining Figure 6.4 it may be noticed that the successfully matched patches include the majority of the salient features within the scene, being the patches which formed the most observable portions of the large objects within the scene. These patches tend to be large in comparison to the majority of patches produced from the decomposition process. It is not surprising that the smaller patches have produced less correct matches as there are many more small patches than large patches and they tend to be more uniform in shape, whereas the large patches represent more of the surface structure. It can be deduced from this that a suitable heuristic for eliminating unuseful patches may be based on the size of the patches.

An alternative approach for eliminating patches may be to examine the error between the shape distributions of the matched patches. It seems likely that patches with large match errors have less likelihood of being correctly matched than matches with low match error.

The remainder of this section describes how the experiments are performed and then individually examines the two patch elimination heuristics mentioned. Finally, the possibility of combining the two heuristics is examined.

Experimental Setup

The effectiveness of patch elimination is measured by using the patch correspondence, $\vec{p}$, and the ground truth correspondence information, $\vec{g}$, and combining with this information pertaining to the size of patches and the match error between matched patches. $\vec{p}$ is computed using Equation (5.1)[p. 64], and $\vec{g}$ is computed using Equation (6.5)[p. 90]. To increase the data available, a set of patches, with corresponding patch size and match error, is constructed by combining the patch matching results when matching from *NoisyScene* onto *SmoothScene* and from *SmoothScene* onto *NoisyScene*.

In the case of eliminating small patches, a threshold is set which eliminates patches that contain a lower number of faces than the threshold value. The threshold is varied in order to find a characteristic curve for the effects of such a threshold. By eliminating small patches, some patches which were correctly matched may also be removed. We hope to observe that a certain threshold value will result in a large proportion of incorrectly matched patches to be eliminated, while eliminating only a small proportion of correctly matched patches. It should be noted that any

values found for such a threshold may be very dependent on the data and so it is desirable that the exact value of the threshold is not very important for achieving a high percentage of correctly matched patches. It is also desirable that the threshold value is either similar across all data sets or that it be possible to determine some suitable threshold directly from the data.

Elimination of patches based on match error is similar to that for small patches. A threshold is set which eliminates patches that were matched with large error value. Again the threshold is varied in order to find a characteristic curve.

Results for Eliminating Small Patches

A threshold on the minimum patch size is varied, at constant intervals, from zero to the maximum patch size. The total number of patches remaining after applying this threshold is recorded. The number of remaining patches which correspond to correct matches is also recorded. Figures 6.5(a) and 6.5(b) show the number of remaining patches and number of remaining correctly matched patches, respectively, after applying patch elimination. The proportion of correct matches against the total number of matches remaining, is shown in Figure 6.5(c).

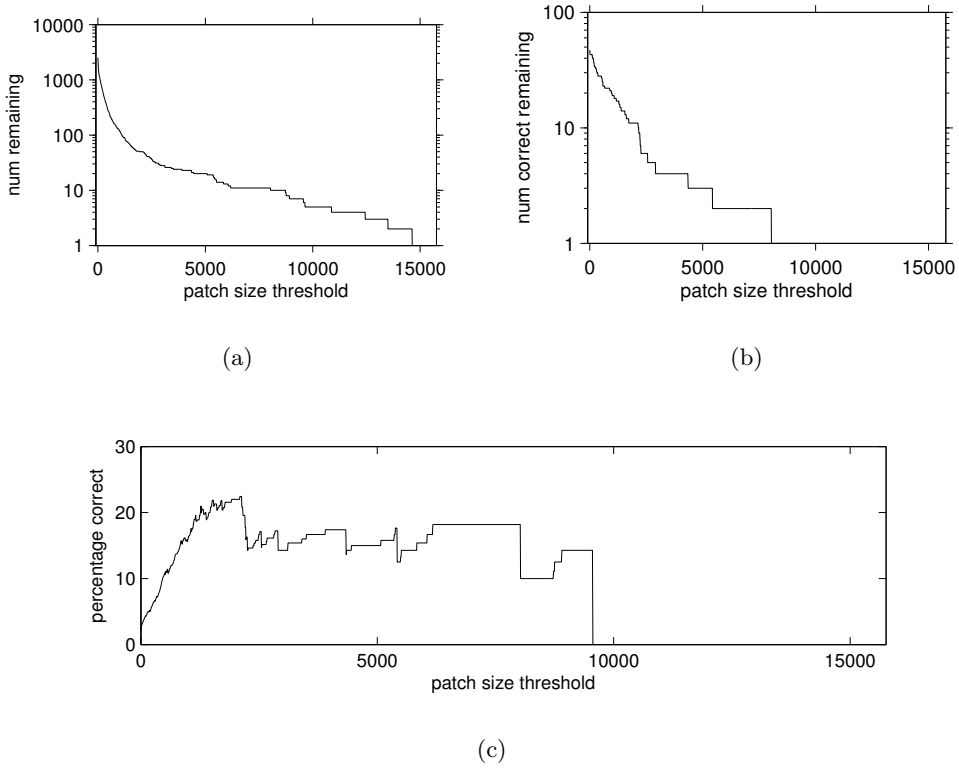
It can be seen from Figure 6.5(a) that the majority of patches are small. For example, the smallest 50% of patches have less than 51 faces, and the smallest 75% have less than 222 faces, leaving the remaining largest 25% to range from 222 to 15763 faces. Figure 6.5(b) indicates that the size of patches in correct matches follows the same trend as the total number of patches: a greater number of small patches yield correct matches than larger patches, however there is a vastly greater number of small patches to large patches so this characteristic on correctly matched patches is not surprising.

Figure 6.5(c) indicates the proportion of correct matches given a particular patch size threshold. It can been seen that patches with greater than about 1500 faces have a much greater likelihood of producing correct matches than smaller patches. A threshold of 2104 faces achieves the greatest proportion of correct matches, however this level only covers a very narrow region. The region of threshold values from 2000 to 7000 gives a proportion of between 15 and 20% correct matches. That same region begins with 50 total patches (11 of which are correct) at a threshold of 2000 faces, and at a threshold of 7000 faces only 11 patches remain, of which 2 are correct.

It can be concluded that eliminating patches of small size is an effective way of increasing the proportion of correctly matched patches. However, care must be taken to avoid eliminating too many patches by setting the threshold too high.

Results for Eliminating Badly Matched Patches

The patch matching process of Section 5.2.2 uses Equation (5.3)[p. 66] to calculate a normalised error of the match between patches. This error is in the range $[0, 1]$ and



(a) (b)

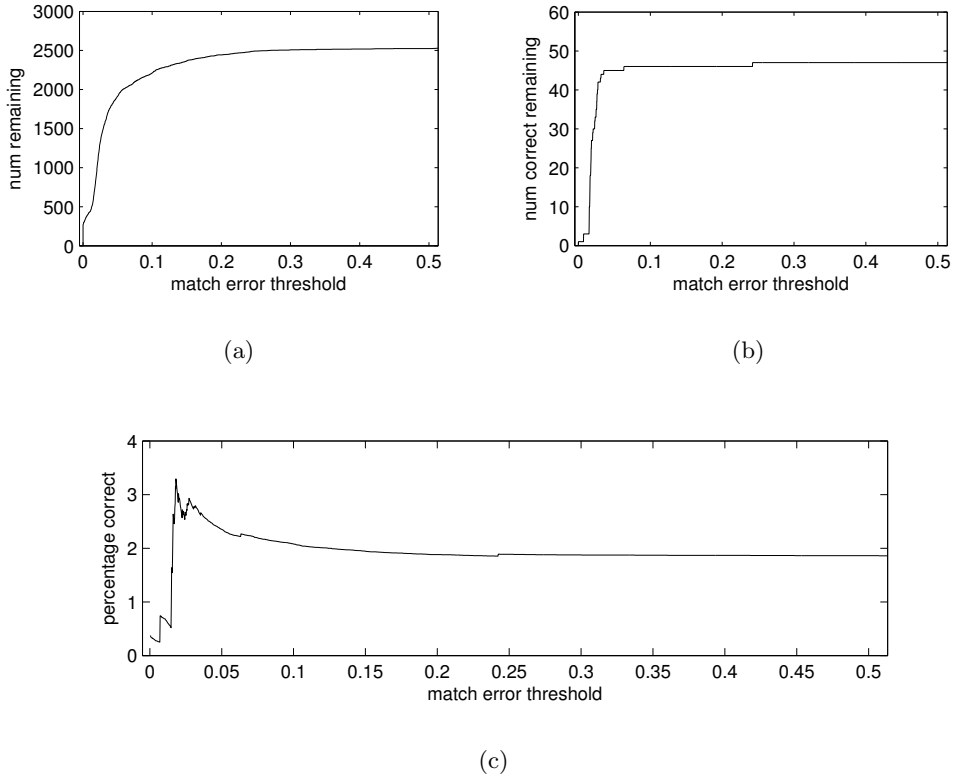
(c)

Figure 6.5: Eliminating small patches
(a) Number of patches remaining after setting a threshold on the minimum allowed patch size (number of faces within patch).
(b) Number of correct patches remaining after min-threshold applied.
(c) Percentage of correct patches remaining after min-threshold applied.

is least for patch pairs which match best. A threshold on the maximum allowed error value is used to eliminate patches in bad matches, or in other words, eliminate the patches which matched with an error value greater than the threshold. The threshold is varied in constant intervals from zero up to the maximum error encountered.

As the threshold is increased from zero, the number of patches accepted increases, as shown in Figure 6.6(a). This figure shows that the majority of patches are involved in patch matches with low error value. The first 50% of patches with lowest error have error values no greater than 0.0238. This result is desirable from the point of view of the Patch Matching process, however it is undesirable for purposes of eliminating unuseful matches because only a very small number of patches have large match errors. Figure 6.6(b) shows the number of correctly matched patches remaining after eliminating patches with match errors greater than the threshold values. Again the same trend is found whereby the majority of correctly matched patches have low match error.

Figure 6.6(c) shows the proportion of correctly matched patches of the total number of patches after eliminations, for each value of the threshold. Without any thresholding effects (such as is observed for a threshold value of 0.5), 1.96% of



(a) (b)

(c)

Figure 6.6: Eliminating poorly matched patches
(a) Number of patches remaining after setting a threshold on the maximum allowed patch match error.
(b) Number of correct patches remaining after max-threshold applied.
(c) Percentage of correct patches remaining after max-threshold applied.

patches are matched correctly. The highest percentage of correct patches is found at a threshold of 0.018, where 3.29% of remaining patches are correct (27 correct out of 820 patches). This is only a very small improvement, confirming our previous comment that thresholding the match error may be ineffective because most patch matches have low error.

6.3.3 Effects of Optimisations on mRANSAC Accuracy

The effects on registration accuracy by the optimisation methods for mRANSAC, described in Section 5.2.7, are now examined. A total of four different methods were attempted for extracting information necessary for computing an updated orientation (rotation and translation). These were: (i) *Closest Face Selection*, (ii) *Random Face Selection*, (iii) *Average Face Coordinate*, and (iv) *Grid Cell Centre*; the last being a naïve method used to compare against the first three in order to show the level of improvement they achieve.

The second and third approaches attempt to provide a means of calculating updated orientation without any significant loss in registration accuracy when com-

Input surfaces:	
current surface	*NoisyScene*
database surface	*SmoothScene*
mRANSAC:	
# iterations ($t_{\max}$)	56
minConsensusCountThreshold	0
$cellSize_1$	0.2 m
$cellSize_2$	varied m

Table 6.7: Parameters used in mRANSAC success rate experiment

pared against the extensive search performed by *Closest Face Selection*.

Experimental Setup

In order to analyse the effects of the four different approaches, without interference from randomisation and other errors, these tests are performed by predetermining the patches selected within Phase One of mRANSAC (see Algorithm 6[p. 69]).

Registration is performed from *NoisyScene* onto *SmoothScene*. Decomposition and patch matching are applied, followed by applying a minimum patch size threshold of 2000 faces. This results in eight correctly matched patches. These eight correctly matched patches give 56 possible combinations of picking three patches, some of which give very poor initial orientation calculations due to the variability of patch centres after the decomposition process. The 56 combinations are used as the predetermined patch selections within Phase One.

The cell size of the grid used for Phase One (mRANSAC algorithm parameter $cellSize_1$) is varied from close to zero up to 1.0 m. For each value of $cellSize_1$, all 56 patch selection combinations are attempted and the average final *coverage* and face-to-face RMS error is measured after Phase Two of mRANASC (whereby the orientation is updated from all closely registered faces and the *coverage* is computed).

The parameters used within the mRANSAC algorithm are given in Table 6.7.

Results

Figure 6.7(a) shows the average RMS error, after Phase Two, for different cell sizes, and for each of the four alternative methods. Figure 6.7(b) shows the average Phase Two *coverage* after processing using each of the different cell sizes but measured using a fixed cell size of 0.02m for all values of $cellSize_1$ – in other words, $cellSize_1$ is varied, but the Phase Two cell size, $cellSize_2$, is set to 0.02m for all tests. Figure 6.7(c) indicates the stability of the approaches by giving the standard deviation of RMS error across the 56 patch selections. No results exist for *Closest Face Selection* with a cell size greater than 0.3 m because of the computation time required to find the closest face within such large cells.

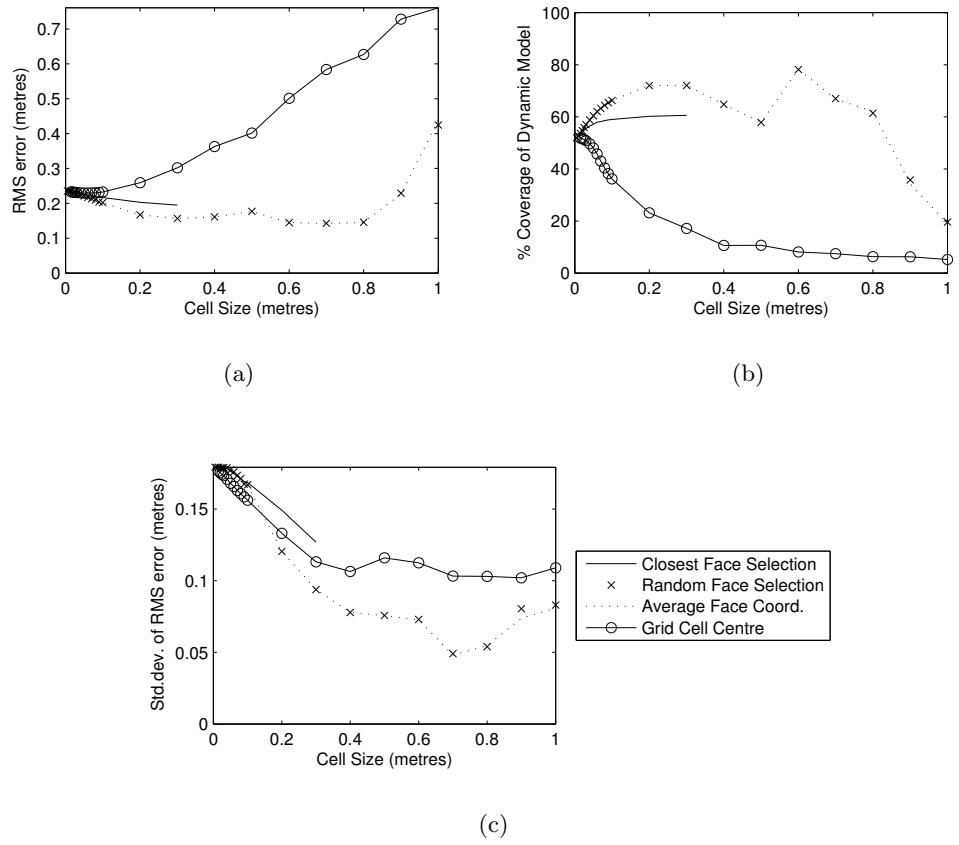


Figure 6.7: Effects of optimisation on registration accuracy
(a) Phase Two Coverage
(b) Face-to-face RMS error between models
(c) Standard deviation of RMS error

The plots of Figures 6.7(a) and (b) indicate a number of very interesting points:

1. the *Closest Face Selection* approach does not achieve the best results, rather *Random Face Selection* and *Average Face Coordinate* both achieve less RMS error and greater coverage.

2. *Random Face Selection* and *Average Face Coordinate* produce almost identical results in both RMS error and coverage. The greatest difference between these two approaches is observed only at very large cell sizes.

3. The accuracy produced using *Closest Face Selection* can be improved with larger cell sizes, however the improvement is only small. Furthermore, as will be discussed below in Section 6.5.3, cell sizes greater than about 0.02 m cause computation times to become prohibitively large.

4. For both *Random Face Selection* and *Average Face Coordinate*, a cell size of between 0.2 m and 0.3 m produces the least RMS error and greatest coverage while not setting the cell size too large. At a cell size of 0.6 m a slightly

greater accuracy is encountered for both approaches, however this trend is not supported by cell sizes immediately less or greater than 0.6 m. Furthermore, when using a cell size of 0.6 m, the distance between a face in the *dynamic model* and a corresponding selected point from the *static model* can be up to $0.6 \times 3\sqrt{3} = 3.12\,m$ (the diagonal distance across three diagonally positioned square boxes of size 0.6 m), which is a very large error tolerance considering that the original surface model is only 1.4 m wide and 1.1 m high. When using a cell size of 0.2 m, the largest accepted face-to-face distance between corresponded faces is 1.04 m, and a cell size of 0.3 m accepts separations up to 1.56 m.

5. The accuracy of results using *Grid Cell Centre* degrades very quickly as the cell size is increased. All other methods achieve much better RMS error and coverage than *Grid Cell Centre* for all cell sizes greater than a very small value. This shows that the three approaches, *Closest Face Selection*, *Random Face Selection* and *Average Face Coordinate*, produce worthwhile results when compared to a very naïve approach, and that the computation time is well spent.

Both *Random Face Selection* and *Average Face Coordinates* employ an averaging effect, whereas *Closest Face Selection* searches for a single closest face; this may explain why *Closest Face Selection* performs worse. How *Average Face Coordinates* employs an averaging effect is clear, however it may not be immediately obvious how *Random Face Selection* creates the effect of averaging, and why *Closest Face Selection* performs so badly. Figure 6.8 shows a 2D analogue to the surfaces and grid in 3D space. The noisy curve represents a *static surface* with sampling noise. The smooth curve represents a *dynamic surface* which is to be reoriented so that it aligns to the static surface through the usual calculation of rotation and translation.

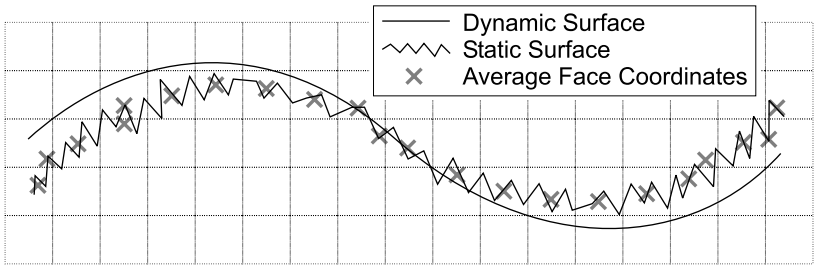


Figure 6.8: 2D analogue to surfaces and grid

Closest Face Selection will pick points on the static surface which are closest to each of the points in the dynamic surface, this will pull the dynamic surface closer to the static surface, but only by a small amount. The dynamic surface is pulled towards the closest edge of the range of noisy data, not towards the median curve.

Average Face Coordinates uses the average points within each grid cell, as represented by the grey crosses within the figure. The dynamic surface is pulled towards the centre of the range of noise within the static surface measurements.

Random Face Selection picks a random face from the static surface within the chosen cell. It does this random selection for each face of the dynamic surface. The random selections for those faces falling into a single grid cell cause an averaging effect, as, if enough random face coordinates are chosen, the average effect will be similar to using the average face coordinate of static surface faces within that cell. Furthermore, the effect of random selections average out over the entire surface – over all grid cells used – assuming the measurement noise is zero mean centred. Thus *Random Face Selection* produces very similar results to *Average Face Selection*, and both are more effective than *Closest Face Selection*, while simultaneously being much more efficient for execution.

6.3.4 Success Rate of mRANSAC

We are now in a position to measure the effectiveness of the DMARA algorithm. It's effectiveness is predominantly affected by the ability of the previously tested processes to provide a list of correspondences that contains a suitable percentage of correctly matched patches. So long as some correctly matched patches are present within that list (three is in fact sufficient, provided the noise is not too great), the exact percentage of correctly matched patches is only a matter of efficiency. However, if the percentage of correct matches is too small, DMARA becomes ineffective by way of requiring far too many iterations within mRANSAC. It is for this reason that the success rate of mRANSAC is now examined. Specifically, we wish to know the registration accuracy of an average iteration of mRANSAC, and to know what percentage of mRANSAC iterations achieve an accuracy within the rotational and translational error requirements set forward at the beginning of this chapter.

The experiments already performed on DMARA have shown that Random Face Selection and Average Face Coordinate approach are the most efficient and also the most accurate methods for finding face correspondences in order to improve the initial registration made within Phase One of mRANSAC. This experiment uses only the Average Face Coordinate approach, by way of being slightly more efficient than Random Face Selection.

The experiments of different methods for finding face correspondences has provided the answer to another question raised in the implementation chapter, specifically "what should $cellSize_1$ be set to?" Recall that $cellSize_1$ determines the physical dimensions over which a search for a corresponding face (or coordinate) is performed. Those experiments show that a good value for $cellSize_1$ is in the range 0.2 m to 0.4 m. 0.2 m seems to be a good value to use; it is conservative enough while potentially enabling a moderately inaccurate registration to be improved.

The experiments on patch elimination show that eliminating smaller patches may be effective for increasing the probability of accurate registrations from each individual iteration of mRANSAC.

Experimental Setup

The decomposed forms of *NoisyScene* and *SmoothScene* are used. Patch matching from *NoisyScene* patches to *SmoothScene* patches is perfomed using their shape distribution representations. The Average Face Coordinate approach is used in the registration improvement part of mRANSAC. The grid parameter $cellSize_1$ is set to 0.2 m for the reasons highlighted above, and $cellSize_2$ is set to 0.02 m for the same reasons as given in Section 6.3.3.

The mRANSAC algorithm is executed for 1000 iterations. After each iteration of mRANSAC, the resultant rotation and translation are used to calculate the RMS error, the rotation angle error and the translation error. The Phase Two coverage (using $cellSize_2$) is also measured.

Three different patch elimination schemes are used and the results for each are compared. These are: (i) no patch elimination, (ii) eliminate patches with less than 1000 faces, and (iii) eliminate patches with less than 2000 faces. The number of patches in each data set are given in Table 6.8 and the resulting number of correctly matched patches (measured using the ground truth calculation of Section 6.3.1) are given for each method.

Registration is performed from *NoisyScene* onto *SmoothScene*. All faces within the *NoisyScene* surface are used for calculation of improved registration and of coverage. Table 6.9.

Patch elimination scheme	# patches remaining		# correct matches
	NoisyScene	*SmoothScene*	(*NoisyScene*)
none	1386	1143	25
less than 1000 faces	57	63	11
less than 2000 faces	23	27	6

Table 6.8: Patch elimination schemes for mRANASC success rate experiment
Number of patches remaining and number of correct matches from *NoisyScene* to *SmoothScene* remaining after applying patch elimination scheme.

Results

For each patch elimination scheme, results for 1000 iterations were gathered, however only a small number of these iterations produce registrations even close to the benchmark goals. Figures 6.9 to 6.11 show the results for each of the elimination schemes. The results in the plots are sorted by ascending RMS error. For example, iteration 3 in Figure 6.9 shows approximately 0.15 m RMS error, 72% coverage, 12°

Input surfaces:	
current surface	*NoisyScene*
database surface	*SmoothScene*
mRANSAC:	
# iterations ($t_{\max}$)	1000
minConsensusCountThreshold	0
$cellSize_1$	0.2 m
$cellSize_2$	0.02 m

Table 6.9: Parameters used in mRANSAC success rate experiment

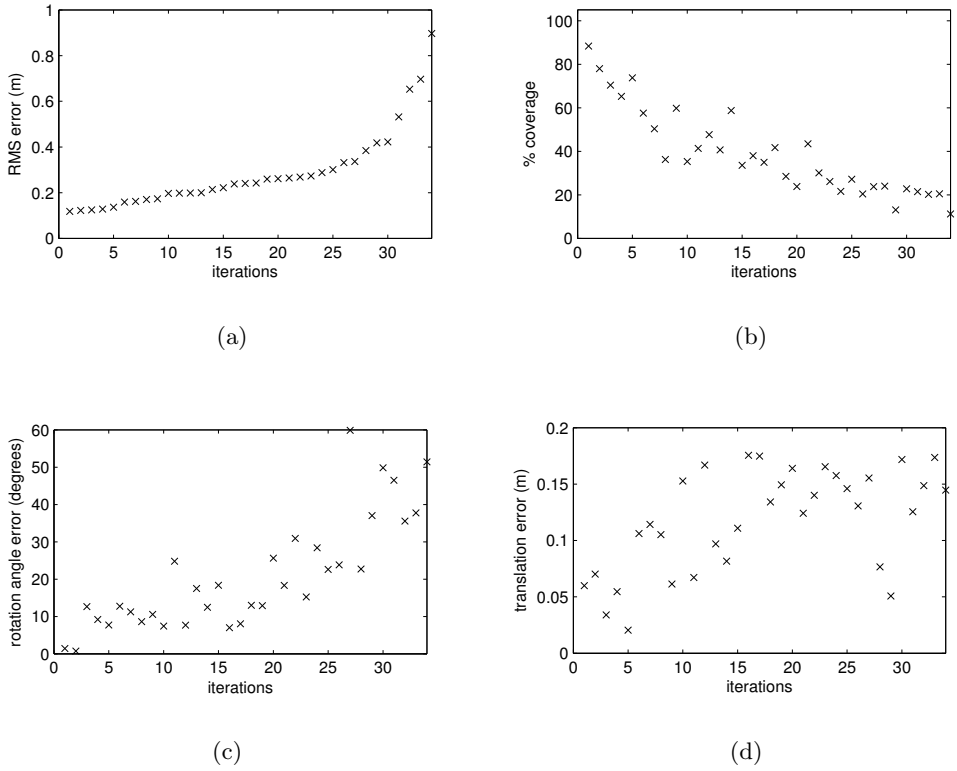


Figure 6.9: mRANSAC success with no patch elimination
RMS error, coverage, rotation angle error and translation error for each iteration. All plots are sorted by RMS error. Note: these are the best of out of the 1000 iterations. All other iterations produced less accurate results.

rotation error, and 0.03 m translation error, respectively, for each of sub plots (a), (b), (c) and (d). Only those iterations resulting in a rotation error of no more than 60° and translation error of no more than 0.2 m are shown.

The number of successful iterations to produce a rotation error no more than 30° and a translation error no more than 0.1 m is 11 for each of no patch elimination and eliminating patches with less than 2000 faces. The number of successful iterations to achieve an RMS error no more than 0.2 m is the same for those two schemes. Elimination of patches with less than 1000 faces resulted in slightly worse success rates:

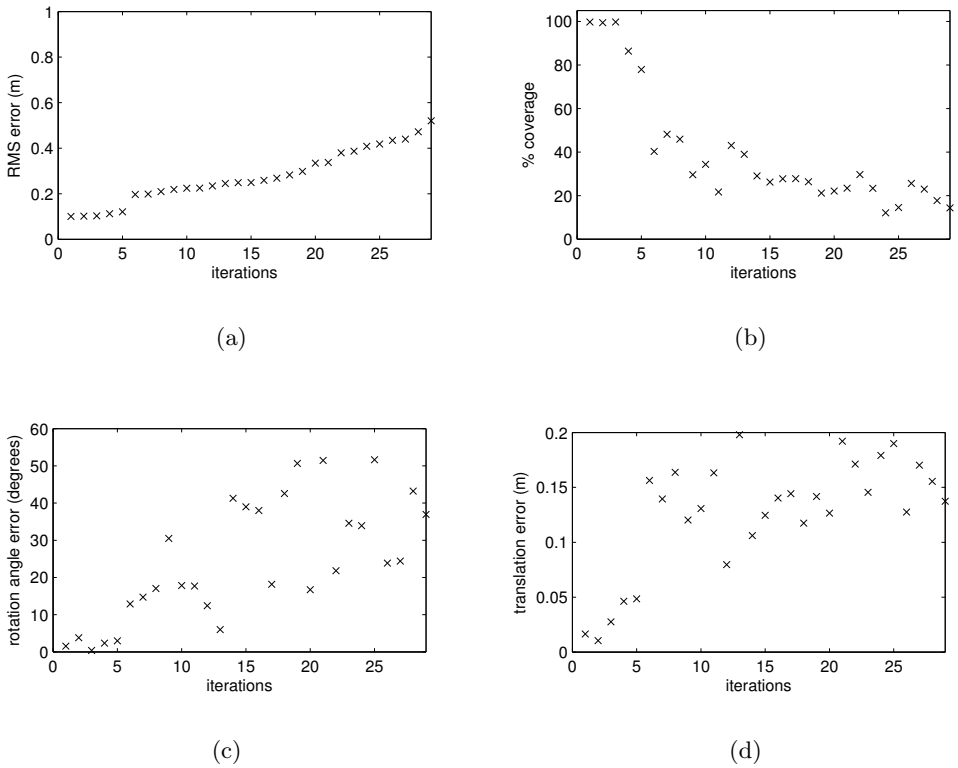


Figure 6.10: mRANSAC success after eliminating patches with smaller than 1000 faces

Patch elimination scheme	Successes within RMS error bounds #	%	Successes within rot. and tran. error bounds #	%
none	11	1.1	11	1.1
less than 1000 faces	7	0.7	6	0.6
less than 2000 faces	11	1.1	11	1.1

Table 6.10: Number of successful mRANSAC iterations
Number of successful iterations (within either set of error bounds) and percentage of total number of iterations for each patch elimination scheme.

6 were within rotation and translation error bounds, and 7 were within the RMS error bound. See Table 6.10 for a summary of these results, and the corresponding percentage of successful solutions.

6.4 Registration Method Two – DPSA

DPSA attempts to register surfaces using only the centres of patches in each surface. A number of questions must be examined:

1. How effective is DPSA for registering surfaces using patch centres?

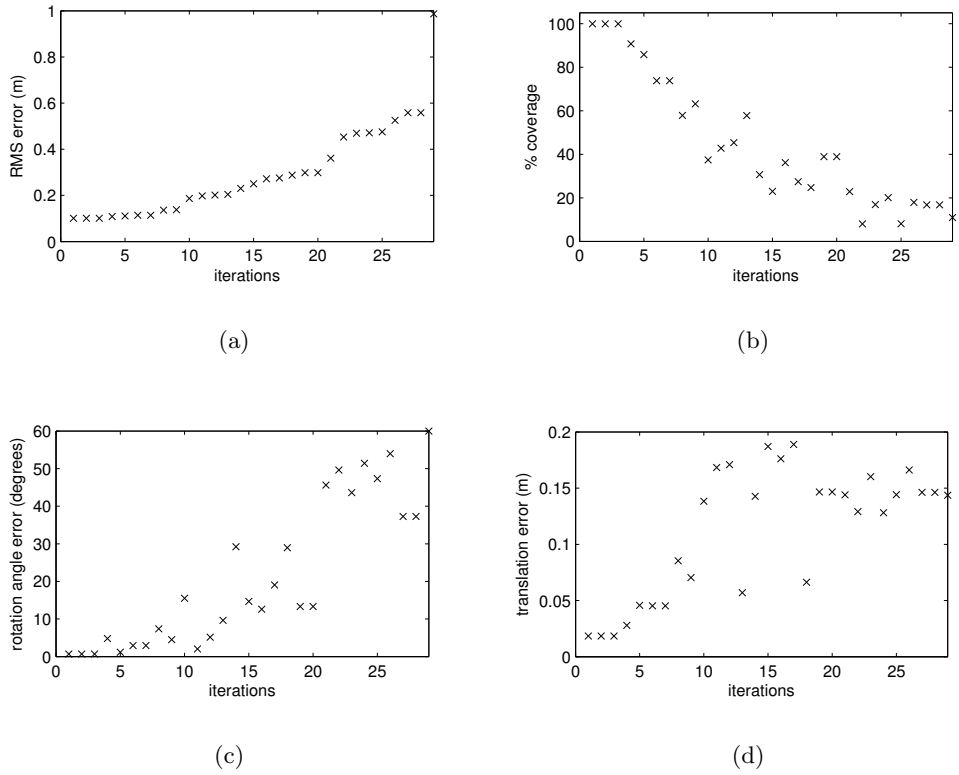


Figure 6.11: mRANSAC success after eliminating patches with smaller than 2000 faces

2. Are the centres of patches suitable to use for feature point positions? Patches can be very irregular and consequently their shape very inconsistent across different surfaces of the same physical region. Their centres may not be consistent across different surfaces either.

3. If patch centres are not used, what is a more suitable alternative?

4. Can comparison of patch shape be included to aid in the search for the best registration using the methodology of DPSA?

Unfortunately, not all of these questions can be fully answered without an extensive investigation. The focus is on the first two questions in the following investigation.

The registration accuracy of DPSA using patch centre data is examined in the first two experiments below. The first experiment uses the patch elimination results of Section 6.3.2 to select only the largest patches for registration. The second experiment considers the effects of patch elimination more closely. The third experiment attempts to address question (2) by discarding patch information altogether and performing a more naïve method of randomly selecting some pre-set number of points.

Input surfaces:	
current surface	*NoisyScene*
database surface	*SmoothScene*
Extended Gold:	
β_0	0.00001 / (mean point distance)
β_f	2000 / (mean point distance)
β_r	1.53
τ_0	100
τ_1	30
δ_0	0.01 m
δ_1	0.0001
# attempts	10

Table 6.11: Parameters used in DPSA experiments

(See Table 4.1[p. 54] for explanation of parameters for Gold *et al.*'s method.)

6.4.1 Patch Centres

Gold *et al.*'s algorithm runs very slowly when the number of input points is large. In a MATLAB implementation of the extended Gold version, using 10 attempts, registering two point sets of size 400 points each, it took about 230 seconds to execute. Whereas, registering point sets of 50 points each took only about 10 seconds. The decomposition process resulted in approximately 1100 to 1300 patches for the data set used. For this reason, the first experiment using DPSA eliminates all but the largest patches and uses the centres of those remaining patches.

Experimental Setup

Surfaces *NoisyScene* and *SmoothScene* are decomposed. The centres of each patch is calculated as the average of the centre points of all faces within the patches.

Two patch elimination schemes are used: (i) eliminate all patches with less than 1000 faces, and (i) eliminate all patches with less than 2000 faces. The number of patches remaining after this process, and the percentage of correctly matched patches can be seen from Table 6.8[p. 100]. The extended Gold method is applied to the centres of the remaining patches from the surfaces of *NoisyScene* and *SmoothScene*.

The parameters used for DPSA are given in Table 6.11. The values for the parametres β_0, β_f and β_r correspond to very conservative and slow estimates of the point set registration. This should ensure an accurate result, however it may cause an unnecessary number of iterations of Gold *et al.*'s method in order to find the registration.

Results

Table 6.12 lists the RMS, rotation and translation error for each of the patch elimination schemes. Note that only one experiment for each patch elimination scheme

Patch elim. scheme	RMS error	rotation error	translation error
less than 1000 faces	0.176 m	15.6°	0.121 m
less than 2000 faces	0.194 m	6.4°	0.164 m

Table 6.12: Results for DPSA on largest patches
RMS, rotation and translation error of applying DPSA to centre points of the largest patches, eliminating all patches with less than 1000 and 2000 faces.

is performed. DPSA achieved registration with less than 0.2 m RMS error, and only 15.6° rotation error in one case and only about 6.4° rotation error in another. Both of these registration accuracies are within the benchmark goals. However, DPSA did not meet the requirements on translation error, having 0.12 m translation error in the best case. Can DPSA do better than this?

6.4.2 Patch Centres with Patch Size Selection

The results of the previous experiment are not convincing. Accepting patch sizes of 1000 or greater yielded slightly better results than accepting only patches of size 2000 or greater, however it is not entirely clear how much patch elimination affects the success rate of DPSA. Neither is it clear how many patches is needed by DPSA to function successfully. A further test is to set a limit of the number of patches used and to apply DPSA to sets of similarly sized patches. For example, 50 patches in the patch size range from 500 faces to whatever upper limit causes approximately 50 patches to be selected. By using an upper and a lower limit in this manner, we may be able to determine some patch-size dependent pattern. Perhaps a particular size of patch gives rise to the best registration accuracy.

Experimental Setup

The extended Gold method is applied a number of times, each time using a different set of patches by applying upper and lower limits to the patches based on patch size. The lower limit is set to a number of pre-set values and the upper limit is calculated so as to accept a certain number of patches. Five different approximate patch counts are compared: 10, 20, 50, 100, and 150 patches.

For each trial, the registration accuracy is measured by the RMS error, the rotation angle error, and the translation error. To test the stability of the results, each trial is repeated 5 times. However, since the same data is used in each repeated trial, little randomness exists.

The lower patch size limit, the calculated upper patch size limit, and the number of patches actually selected from each of the data set surfaces is listed in Table 6.13 and Table 6.14 for each of the different patch counts. The same parameters for the extended Gold method are used as described in Table 6.11.

Patch size thresholds		# patches found	
min	max	*N.S.*	*S.S.*
1	1	28	23
5	5	26	20
10	10	26	19
20	20	12	8
50	52	14	8
100	103	11	5
200	207	11	7
500	541	11	15
1000	1121	11	12
2000	3906	11	16

(a)

Patch size thresholds		# patches found	
min	max	*N.S.*	*S.S.*
1	1	28	23
5	5	26	20
10	10	26	19
20	22	32	17
50	54	22	18
100	109	24	15
200	219	21	23
500	560	21	20
1000	1387	21	25
2000	9562	21	23

(b)

Patch size thresholds		# patches found	
min	max	*N.S.*	*S.S.*
1	2	152	107
5	6	51	41
10	12	61	65
20	25	58	33
50	65	51	51
100	120	54	37
200	235	51	42
500	723	51	47
1000	5840	51	56
1075	14625	51	55

(c)

Patch size thresholds		# patches found	
min	max	*N.S.*	*S.S.*
1	2	152	107
5	9	120	107
10	15	109	105
20	31	101	72
50	82	101	91
100	141	103	62
200	288	101	94
500	1481	101	97
609	9639	101	98

(d)

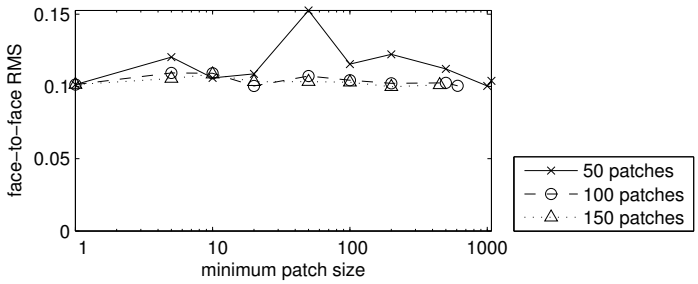
Table 6.13: Patch size limits for DSPA patch centre experiment
Patch size upper and lower limits and the numbers of patches actually selected for (a) 10, (b) 20, (c) 50, and (d) 100 patches. *NoisyScene* is shortened to *N.S.* and *SmoothScene* is shortened to *S.S.*.

Results

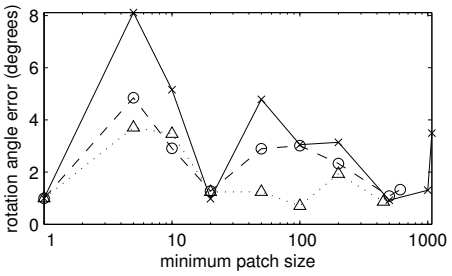
Figure 6.12 shows the results of applying the extended Gold method to the patches selected by the process described above for each of 50, 100 and 150 patches. The results for 10 and 20 patches are substantially different to 50 or more patches and so the RMS error results for them are listed in Table 6.15. On the whole, the results were very similar for 50, 100, and a 150 patches. Using 50 patch centres tended to produce less accurate results than 100 or 150 patch centres, and 150 patches sometimes produced slightly better results than using only 100 patches. All instances of using 50 or more patches were within 0.15 m RMS error and were less

Patch size thresholds		# patches found	
min	max	*N.S.*	*S.S.*
1	2	152	107
5	11	161	148
10	20	160	141
20	39	152	100
50	103	156	127
100	171	154	104
200	355	151	135
450	14625	151	143

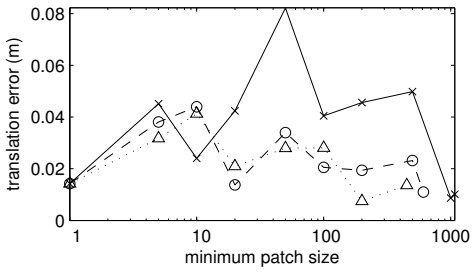
Table 6.14: Patch size limits for DSPA patch centre experiment: 150 patches



(a)



(b)



(c)

Figure 6.12: Results of DPSA on patch sizes of 50, 100 and 150
RMS error, rotation error and translation error for applying extended Gold method to sets of approximately 50, 100, and 150 patches, using different ranges of patch sizes.

than 10° rotational error and 0.1 m translation error. In contrast, DPSA performed poorly using 10 and 20 patches. Only three instances of using 10 patches produced less than 0.2 m RMS error, and four succeeded using 20 patches.

Some patch sizes produces slightly better results than others for 50 or more patches. For example, patch sizes starting at 1, 20, and 500 patches produced rotational error less than 2° and translation error no greater than about 0.04 m

Min patch size	RMS error	
	10 patches	20 patches
1	0.665	0.665
5	0.154	0.154
10	2.109	2.109
20	0.810	2.082
50	0.349	0.170
100	0.288	0.225
200	0.198	0.221
500	2.191	0.140
1000	2.148	2.156
2000	0.183	0.099

Table 6.15: Results of DPSA on patch sizes of 10 and 20

(40 mm). However, this trend is to be taken very lightly considering the limited nature of experiments on only one basic data set. Despite this, it is interesting that patches between one and two faces only in size yield such successful results.

Each trial was repeated five times with the same parameters, in order to measure stability, and all repetitions of the trials produced the same results. This is not particularly surprising since the same patch centre coordinates are used in each repetition.

6.4.3 Random Point Selection

In order to determine the effectiveness of selecting patch centres, the previous results are compared to a more naïve approach whereby points are randomly selected from the centres of all faces within the whole of the surfaces, instead of using decomposition to control where to select points from. The results of this test will indicate the effectiveness of the decomposition approach for feature extraction.

Note that randomly selecting points in this fashion is naïve because it will fail considerably when the area of the two surfaces are very different. For example, if the current surface covers a quarter of the area covered by the database surface and 40 points are selected in each, then approximately only 10 points in the database surface will be covering the same area as the current surface. DPSA has this same problem too, of course, since it must limit the number of points. However, DSPA can handle this problem better because it can set a threshold on the size of patches and thus the density of feature points will be independent of the surface area.

Experimental Setup

The original data set surfaces are used, without decomposition, and a fixed number of points are randomly selected from either surface. The extended Gold algorithm is then applied to the extracted point sets and the RMS error, the rotational error

and the translation error measured on the resultant registration. 100 trials of this approach are performed, using a different randomly selected set of points for each trial. Furthermore, several different point set sizes are used, 100 trials for each, ranging from 10 points to 150 points.

The values of parameters for DPSA are given in Table 6.11.

Results

The results for this experiment are examined in three ways. Firstly, the percentage of successful trials are measured for each number of points by comparing against the RMS error benchmark goal and against the rotation and translation error goals. The results for this are shown in Table 6.16. Secondly, the RMS error values for all trials of four different point set sizes are shown in Figure 6.13. Finally, the rotation and translation error values for all trials of four different point set sizes are shown together in Figure 6.14.

# of points	% within RMS bound	% within rot. & trans. bounds
10	17	17
15	33	26
20	53	43
30	83	79
40	92	90
50	97	97
60	97	97
70	100	100
80	100	100
90	100	100
100	100	100
150	100	100

Table 6.16: Success rate of DPSA with random point selection
The percentage of 100 trials for which the extended Gold method achieved error of no worse than (a) 0.2 m RMS error, (b) 30° rotation or 0.1 m translation error, using different numbers of randomly selected points.

Table 6.16 shows a definite trend in the percentage of successes achieved by the extended Gold method on randomly selected points. When selecting only 10 points, 17% of trials succeed. When selecting only twice that many points, 53% succeed in terms of the RMS error benchmark and 43% succeed in terms of the rotation and translation benchmark. Selecting 50 points allows for 97% success of all trials, and by selecting 70 points 100% of trials succeed.

Figure 6.13 shows these results in a different way by examining the RMS error for every trial over four different numbers of points. Each of these curves have been sorted in ascending order of RMS error so that it is easy to see what proportion of trials give different ranges of RMS errors. As can be seen in Figure 6.13(a),

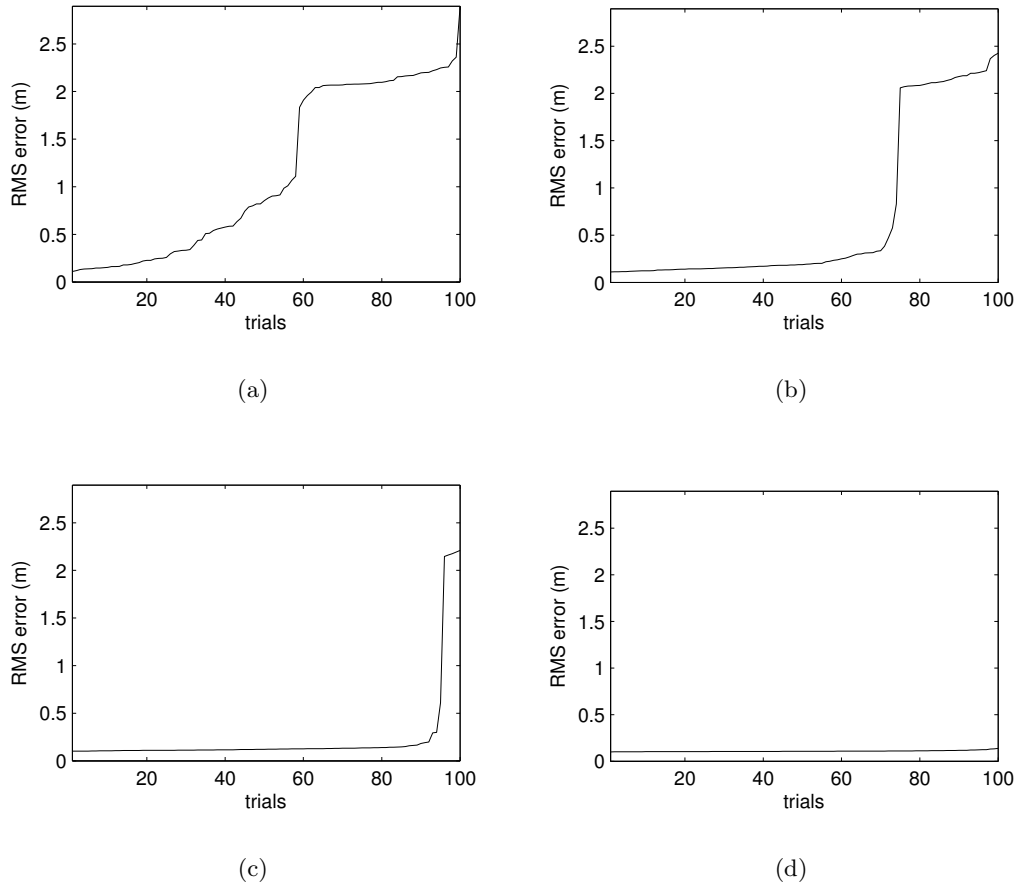


Figure 6.13: RMS error of DPSA with random point selection
RMS error for each of 100 trials using different numbers of randomly selected points: (a) 10, (b) 20, (c) 40, (d) 80. The plots are sorted for sake of clarity, and consequently the individual points form a curve.

using only 10 points is very unreliable. Approximately 40% of trials using only 10 randomly selected points resulted in RMS errors greater than 2 m. Figure 6.13(b) shows that using 20 points reduces this to about 30%. In fact, using 20 points, the majority of trials are below 0.4 m RMS error. Figure 6.13(d) shows the RMS error for the trials when 80 points are used. None of the trials failed in this case. Another pattern to notice is that the curve of RMS errors for successful trials gets increasingly flatter as the number of points are increased. This indicates that the stability of the result is more consistent and is due to the fact that with more points being sampled the alignment error is reduced because the corresponding points have a greater chance of being close to each other.

Figure 6.14 shows the same general trend when examining the rotation and translation error. The plots within this figure show the rotation and translation error combined on an scatter plot. A noticeable trend is that, as the numbers of points increases, the dots cluster together more towards the corner of zero error,

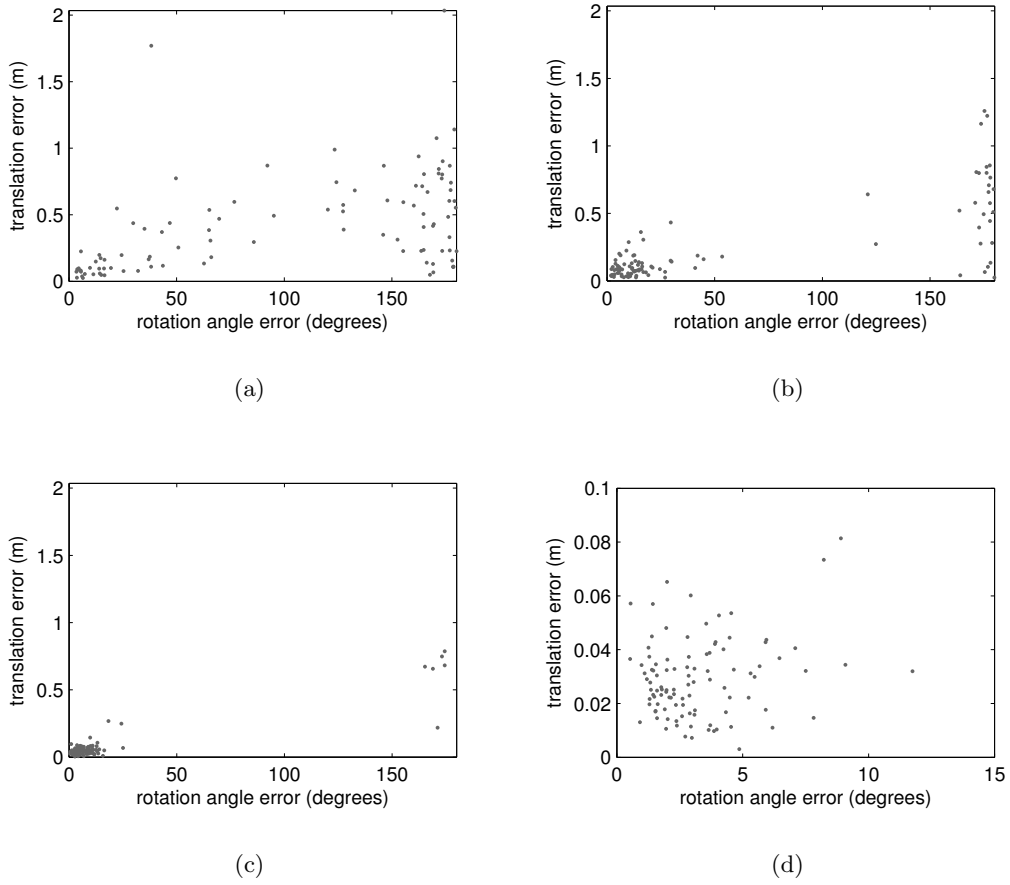


Figure 6.14: Rotation and translation error of DPSA with random point selection
Rotaton and translation error scatter plots for 100 trials using different numbers of randomly selected points: (a) 10, (b) 20, (c) 40, (d) 80.
Note the different x-axis and y-axis scale in plot (d), no results are observed outside the limits shown.

with only a few outliers appearing with large errors. Notice the range of values for Figure 6.14(d), all trials were within 12° rotation error and 0.08 m translation error.

The effectiveness of DPSA using randomly selected points is very similar to using patch centres, for the same number of points. This indicates two things. Firstly, Gold *et al.*'s method is very effective at coping with noisy input data. Using only a small number of randomly selected points, it is still able to find the correct registration quite often. Secondly, DPSA on patch centres is no worse than when using random selection on this data. This is good, because it means that using patch centres is likely to be better than randomly selecting points when applying DPSA to registering surfaces when one covers a much larger area than the other, due to the sampling density problem mentioned earlier.

6.5 Algorithm Efficiency of DMARA

The DMARA method takes a database surface, with all pre-processing already performed, and a raw triangular mesh for the current surface. It is assumed that a filter has been applied to the current surface to remove noise. If the current surface has F_C faces, the database surface has F_D faces, and the database surface is already decomposed into P_D patches, having shape distribution representations, DMARA involves the following major processes:

- The current surface is decomposed into P_C patches.
- Shape distribution representations are generated for each of each of the P_C patches from the current surface.
- Matches are found from each of the P_C patches to the most similar of the P_D patches in the database surface.
- The mRANSAC algorithm is applied to find the final registration.

Before examining the time complexity of the individual parts of DMARA, an analysis of the number of iterations required by mRANSAC will be given. This is followed by examining the time complexity, and finally giving some typical execution times.

6.5.1 Number of Iterations of mRANSAC

The probability of successfully registering two surfaces can be calculated from the measurements presented within Section 6.3.4. Those measurements show that, for the data set and parameters used, 1.1% of iterations of mRANSAC successfully register the surfaces within the benchmark goals. Eliminating small patches does not appear to improve this.

Given a probability, p, for an individual iteration of mRANSAC to successfully register two surfaces, Equation (6.2)[p. 89] calculates the number of iterations, n, required to achieve a desired probability of success, P_{success}. The results of Section 6.3.4 suggest that p is 1.1%. To achieve 90% probability of succeeding at least once, 209 iterations are required. For 95% and 99% probability of a single success, 271 and 417 iterations are required, respectively. If p is only 0.6%, the number of iterations required are 383, 498 and 766 for 90%, 95% and 99% success probability, respectively. For DMARA to be suitable for real-time robotic navigation, mRANSAC must be capable of being executed using around 500 to 1000 iterations in a fraction of a second.

6.5.2 Complexity Analysis

Decomposition of surfaces, using the Watershed algorithm, has a computational complexity of $O(F_C)$. The merging algorithm can be implemented very efficiently

by using the right data structures, allowing $O(F_C)$ complexity also. The result of the decomposition is P_C patches, where $P_C \leq F_C$.

The results of the decomposition process affect somewhat the efficiency of the remaining processes. If the data is not sufficiently filtered to remove noise, the decomposition process will result in many tiny patches, even after merging, resulting in the remaining processes to perform very poorly. However, with care, this problem can be avoided. A solution is to apply an algorithm before decomposition which measures the noise and filters the surface data until some pre-set level is attained. On the assumption that some filtering algorithm is used, the remaining process are relatively efficient.

Calculation of patch distributions, accumulated over all patches for the current surface, require $O(F_C)$ for setting up the lookup table of face coordinates (used to index the faces by cumulative area value). Calculation of the distributions for P_C patches is dependent on the number of samples taken, N, and independent of the number of faces. The whole of the patch distribution calculation thus has computational complexity of $O(F_C)$.

Tentative patch matching requires $O(P_C P_D)$ to find matches from all patches in the current surface to the most similar of all patches in the database surface.

The mRANSAC algorithm involves a number of separate parts. Two grids are constructed using some number of cells. The number of cells, N_{cells1} and N_{cells2}, are dependent on the range of coordinates within the database surface and the values of $cellSize_1$ and $cellSize_2$. The computation time to set up the grids is: $O(F_D + N_{\text{cells1}} + N_{\text{cells2}})$. The computation time to perform lookups on the grid depends on the type of grid optimisation used. Closest Face Selection requires at worst $O(F_C F_D)$ per iteration, but with sufficiently small *cellSize*, the computation cost can be approximated by $O(F_C \log_b F_D)$, where $b \geq 1$ is some value dependent on the size of the grid cells and the density of faces within the database surface mesh. With very small *cellSize*, b can be close to 1. All other methods require only $O(F_C)$ per iteration, and both Average Face Coordinate and Random Face Selection have been shown to perform better than Closest Face Selection. The final computational complexity of mRANSAC is $O(nF_C + F_D + P_C P_D + N_{\text{cells1}} + N_{\text{cells2}})$, where n is the number of iterations of mRANSAC.

The overall complexity of the whole DMARA algorithm is summarised in Table 6.17.

6.5.3 Execution Time

DMARA is implemented partly in MATLAB and partly in C, on a Dual core Intel(R) Xeon(TM) CPU, 2.40GHz, with 512 KB L1 cache, and enough RAM to store all data necessary for calculations without swapping to secondary memory.

The decomposition process is implemented efficiently in C for all but the merg-

Decomposition	$O(F_C)$
Patch distributions	$O(F_C)$
Patch matching	$O(P_C P_D)$
mRANSAC	$O(F_D + nF_C + N_{\text{cells1}} + N_{\text{cells2}})$
total	$O(nF_C + F_D + P_C P_D + N_{\text{cells1}} + N_{\text{cells2}})$

Table 6.17: Computational complexity of DMARA
where:
F_C and F_D – number of faces in the current surface and database surface, respectively.
P_C and P_D – number of patches in current surface and database surface, respectively.
N_{cells1} and N_{cells2} – number of cells generated for calculation of closest faces and coverage.
n – required number of iterations of mRANSAC.

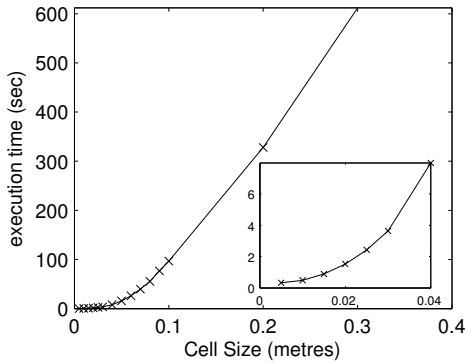
ing part. Typical execution time for the decomposition process (excluding merging) is 1.02 seconds. Shape distribution calculation is implemented in C. Taking 10 000 samples for each patch, the shape distribution calculation requires 5.9 ms per patch. For *NoisyScene*, having 1386 patches, it requires 8.16 seconds to calculate all shape distributions. The tentative patch matching process is not implemented in an efficient manner, and so execution time cannot be given. Each iteration of mRANSAC, using the parameters given in previous experiments, requires 1.22 seconds to execute when using Average Face Coordinate for face correspondence calculation.

The execution time to calculate coverage for a single iteration of mRANSAC, using Closest Face Selection for face correspondence, is given in Figure 6.15(a) for different values of $cellSize_1$. The inset shows the detail with small $cellSize_1$ up to only 0.04 m, and shows that for 0.04 m it requires about eight seconds to find the closest database surface faces to all faces within the current surface. From this plot it can be seen that any $cellSize_1$ greater than about 0.02 m causes the execution time mRANSAC to be prohibitively large if using Closest Face Selection. Figure 6.15(b) shows the execution times to calculate coverage using the alternative methods of Random Face Selection, Average Face Coordinate, and Grid Cell Centre. It can be seen that Closest Face Selection is considerably less efficient than the alternative methods.

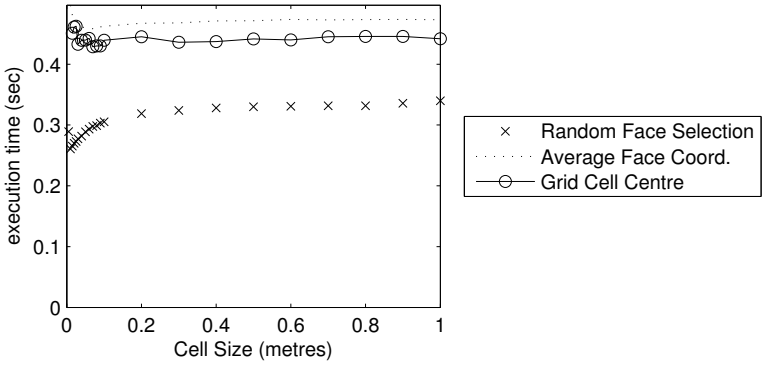
The execution times for the different parts of DMARA are summarised in Table 6.18. The execution time per iteration of mRANSAC is measured using Average Face Coordinate.

Decomposition*	1.02 seconds (0.07)
Patch distributions	8.16 seconds (0.02)
Patch matching	*unknown*
iteration of mRANSAC	1.22 seconds (0.142)

Table 6.18: Execution times for DMARA
Time to execute each part of DMARA. The standard deviation is given in brackets.
unknown is given when execution time could not be measured.
* – excludes execution time of merging process.



(a)



(b)

Figure 6.15: Execution times for coverage calculations
Time to compute coverage measure using different values for $cellSize_2$:
(a) Closest Face Selection
(b) other calculation methods

6.6 Algorithm Efficiency of DPSA

The DPSA method uses decomposition followed by applying the extended Gold method. The computational complexity is examined below, followed by actual execution times.

6.6.1 Complexity Analysis

The same decomposition process is used by DPSA as by DMARA. That has a computational complexity of $O(F_C)$, where F_C is the number of faces in the current surface. Gold *et al.*'s method requires $O(lm)$ for l and m points in either point set respectively. In DPSA, l and m are approximately equal to the number of patches, P_C and P_D, in the current and database surfaces, respectively. Thus, Gold *et al.*'s method, applied to patch centres has $O(P_C P_D)$ computational complexity. The

Decomposition	$O(F_C)$
Gold *et al.*'s method	$O(P_C P_D)$
total	$O(F_C + P_C P_D)$

Table 6.19: Computational complexity of DPSA

where:
F_C and F_D – number of faces in the current surface and database surface, respectively.
P_C and P_D – number of patches in current surface and database surface, respectively.

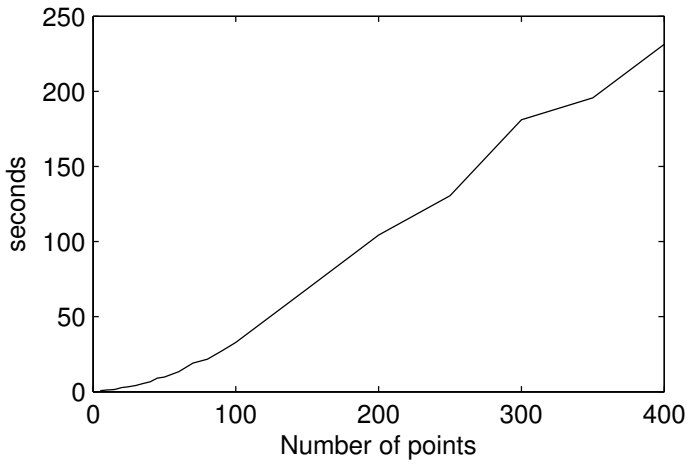


Figure 6.16: Execution time of extended Gold method

The execution time required by the extended Gold *et al.*'s method (using 10 attempts) for different numbers of points. Standard deviation of time at each point is approximately equal to 14% of the plotted execution time

overall complexity of DPSA is thus $O(F_C + P_C P_D)$, as summarised in Table 6.19.

6.6.2 Execution Time

DPSA is implemented in MATLAB. Typical execution times for the extended Gold method, using 10 attempts and selecting the best, are given in Figure 6.16 for different numbers of points. The control parameters used are the same as given in Table 6.11. For 10, 20, 50, and 100 points, the extended Gold method executes in approximately 1.2 seconds, 2.7 seconds, 9.8 seconds and 33 seconds, respectively. This shows how Gold *et al.*'s method increases rapidly in execution time for increased numbers of points.

Chapter 7

Conclusions and Further Work

This thesis set out to examine whether decomposition can be used as a basis for feature extraction in the hope that registration, and consequently recognition, can be performed both more efficiently and with greater accuracy than existing methods. Decomposition was tested by introducing two new methods for surface registration, DMARA ("Decomposition, Match Assignment, and RANSAC Alignment") and DPSA ("Decomposition and Point Set Alignment").

Decomposition for feature extraction has a major advantage over selecting feature points randomly because it controls much better where feature points are chosen. Two reasons are given for this. Firstly, there is a greater chance of feature points being chosen at positions corresponding to the same physical point when selecting, say, the patch centres, than when picking points randomly. Secondly, feature density is independent of the size of the surface.

The idea of feature point density was alluded to within Chapter 6. A problem encountered with randomly selecting feature points is when the areas covered by the current and database surfaces are substantially different. If a fixed number of points is chosen in either set, then the density of feature points is different. This can lead to feature points being too sparse in the important area within the database surface for registration to be possible. Decomposition partly overcomes this because any limits applied on the number of feature points can be applied in a manner which depends on the size of the patches. For example, DPSA was too slow to be performed on all 1300 patches, and so various upper and lower limits on patch size were used to control the number of feature points extracted (in this case patch centres). These limits adjust the number of feature points in a more dynamic fashion. In fact, this method controls the number of feature points in a fashion which is almost independent on the area of the surfaces and so the density of feature points is the same within both surfaces (assuming the two surfaces are smoothed to similar levels of noise).

Patches can be matched by defining a patch descriptor to describe the overall shape of the patches and to perform a search using some similarity measure. Shape distributions were used for their simplicity of implementation and efficiency for cal-

culating similarity. However, the decomposition process is affected badly by noise in the input surfaces and so any shape based patch matching method will result in a large number of patches which are incorrectly matched.

Elimination of patches was investigated as a way of increasing the probability of successful registration, particularly for DMARA. However, results using DMARA showed no benefit from patch elimination.

Benefit from patch elimination was found when using DPSA. It was shown that setting upper and lower limits on the number of faces within a patch is an effective way of reducing the number of patches and that DPSA gave very favourable results using this approach. It was also shown that the size of patches used does not considerably affect the success rate of DPSA. Using 50 small patches, for example, produced roughly the same success rate as using 50 medium sized or large sized patches. This would seem to indicate that the patch size alone is a suitable means for increasing the proportion of correctly matched patches, and that it can be used to help the efficiency of DPSA. Perhaps patch size alone is sufficient enough for use as a patch descriptor, at least for use with DPSA.

DMARA has been shown to be capable of registering two surfaces. However, experiments so far have shown it be too inefficient as it stands. The main problem is that the raw surface meshes were used for calculation of the updated registration and the coverage measure. A more sensible approach is to sub-sample the surface meshes before using within DMARA so that only a small fraction of points are used to calculate the registration parameters and coverage. Of course, the sub-sampling process must be carefully applied in order to not loose important structural information from the surfaces. Another problem with DMARA is that it requires a large amount of memory to store the grid of face coordinates. However, one advantage DMARA has over DPSA is that it can handle large numbers of patches.

DPSA is very efficient, but only for small numbers of patches. Experiments showed that DPSA requires around 50 patches or more in order to successfully register surfaces. Experiments also showed that, provided enough patches are used, DPSA is very stable. DPSA consistently achieved high success rates in a number of different situations. The stability of DPSA stems from the nature of Gold *et al.*'s method, which directly searches for a set of correspondences satisfying the geometric constraint. Satisfying the geometric constraint means that all matched patches lead to the same rotation and translation. In this way, DPSA makes up for not considering how similar the shape of patches are.

Like DMARA, DPSA also requires a large amount of memory. Its memory usage is $O(P_C P_D)$ at worst, where P_C and P_D are the number of patches in each of the current surface and database surface, respectively.

These results indicate that the approaches given in this thesis have the potential to be effective for 3D surface recognition. DPSA is shown to be more efficient than DMARA for small numbers of patches (around 50 to 100), however DMARA

can handle any number of patches. Suggestions are that patch elimination can be very beneficial as a way of selecting patches for use in DPSA, helping to aid in its efficiency. However, patch elimination does not appear to benefit DMARA. Decomposition is shown to have good potential for feature selection.

7.1 Further Research

The results of experiments on these new methods has raised a number of questions, and like all research, the specific approaches used can be chosen differently or varied slightly and different results will be achieved. Furthermore, much more testing is required. Particularly, a much larger database of surfaces needs to be used in order to accurately measure the various characteristics of these methods. An examination of some potential areas for future research is now given.

7.1.1 Decomposition

Decomposition has been used because it naturally gives rise to a controlled feature selection. However, a major drawback of decomposition is that the number of result patches, and their shape, is very dependent on the noise inherent within the surface being decomposed. The solution employed here is to manually smooth the data and perform a post merging process where patches which are approximately co-planar are joined. An extension is to perform the smoothing step automatically. Gregor and Whitaker [37] perform smoothing on range data in an iterative fashion using an edge preserving diffusion technique, and then apply a watershed decomposition algorithm. A smoothing algorithm, such as used by Gregor and Whitaker, could be used and repeatedly applied until a pre-set maximum level of noise is achieved.

7.1.2 Improvements and Further Research on DMARA

The efficiency of DMARA is affected greatly by the proportion of correctly matched patches. However, elimination of all but the largest patches failed to improve the efficiency despite successfully increasing the proportion of correctly matched patches. A probably cause is that the largest patches are also most inconsistent between surfaces due to noise, and so their centres do not align well. While examining DPSA it was discovered that limiting the patches to a small band of sizes produced good results. This should be examined more closely to discover whether it can improve the efficiency of DMARA as well.

Furthermore, different patch descriptors should be examined. Patch distributions handle small levels of noise well, however they are not effective to match patches when one patch is clipped due to the decomposition process. In other words, patch distributions don't handle partial matches well. Other methods need to be examined, such as considering the use of Turning Angle Functions [6].

DMARA uses the Coverage measure as a heuristic to determine which registrations are better. Currently, coverage only measures the number of faces which are corresponded. Two things can be changed. Firstly, not all faces are the same size, it may be more accurate to use the cumulative area of the corresponding faces rather than the number of faces. Secondly, some capture methods provide a means of determining a measure of certainty or uncertainty about the measurements at different points. For example, when only a single view is used, surfaces which are in line with the incident line from object to focal point will not have any data captured, but these surfaces will instead by extrapolated as part of the capture process. Such surfaces should be considered to be very uncertain in their positions, whereas surfaces which are perpendicular to the camera will have high accuracy. The coverage calculation could be extended to incorporate such measures of data accuracy by giving low importance to measurements which have low accuracy, and high importance to measurements of high accuracy.

7.1.3 Improvements and Further Research on DPSA

It was discovered in Section 6.4.2 that the size of the patches used doesn't appear to affect the registration accuracy of DPSA. It was noted earlier in this chapter that this could be because the patch size elimination effectively reduces the search to similar patches (like a simple form of patch matching). To test this theory, the registration accuracy should be tested when a fixed number of patches are randomly selected, without limiting the patch size. If this performs worse than when using upper and lower patch size limits, then the theory may be correct.

Both mRANSAC (DMARA) and Gold *et al.*'s algorithm (DPSA) require point coordinates as inputs. However, patch centres give a very poor likelihood of correct registration due to the affects of noise on the decomposition process. Gold *et al.*'s algorithm is particularly affected by this because it does not consider the similar between patches. However, Gold *et al.*'s algorithm can be extended to incorporate the patch correspondence information calculated as described in Section 5.2.2.

The error function used by Gold *et al.*'s algorithm currently attempts to minimise the RMS error between corresponding points. Equation (4.15)[p. 53] can be extended to bias towards correspondences which are similar to the correspondences found using the shape descriptors. A possible alternative error function is:

$$E(m, \vec{T}, \mathbf{R}) = \sum_{j=1}^{J} \sum_{k=1}^{K} \left((1-\beta) c_{jk} + \beta m_{jk} \| \vec{X}_j - (\vec{T} + \mathbf{R}\vec{Y}_k) \|^2 \right) \tag{7.1}$$

where c_{jk} is the patch correspondence matrix (of same size as m) produced from the

patch matching algorithm by the following equation

$$c_{jk} = \begin{cases} 1 & \text{if } p_j = k \\ 0 & \text{otherwise,} \end{cases} \tag{7.2}$$

$\beta \in [0,1]$ determines the relative importance of the match matrix and the patch correspondence matrix, and $\vec{p} = (p_1, p_2, \ldots, p_K)$ is an *any-to-one* mapping from each of K patches in the current model onto patches within the database model as computed by Equation (5.1)[p. 64].

7.1.4 Further Research into Heuristics

The methods used within this thesis contain many control parameters. Further work needs to address how these parameters are set. A list of such parameters, and some suggestions, follows:

- Section 6.1.1 described very briefly the post-decomposition process of merging close-to co-planar patches. This process involves deciding on what "close to co-planar" means. Fixed values were used within all experiments within this report, without examining in detail an effective way of selecting such values. Much research needs to be done to find the best method for merging patches.
- Elimination of small patches is currently based on the number of faces within the patches. This gives very little physical interpretation, and is affected by the density of the surface tessellation (the size of the faces). A better approach is to apply thresholding based on the area covered by the patches. This has a much easier connection to real-world measurements, perhaps indicating the typical size of objects which are valid for recognition purposes.
- Gold *et al.*'s method, used within DPSA, has many control parameters. These affect the ability to overcome local minima solutions, the efficiency, and the accuracy. Research needs to examine how to set these parameters for the purposes of using within DPSA.

7.1.5 Partial Surface Matching

The experiments of Chapter 6 only considered registration when identical surfaces were used (with different levels of noise). The effectiveness of registering surfaces captured from different view points must now be measured. In particular, further research needs to address the following questions.

- How effective are these methods when handling occlusions?
- What proportion of overlap between the input surfaces is required for accurate registration. For example, can these methods find the registration when only 20% of the surface area present in one surface is also present in the other?

- How effective are they when the current surface is substantially smaller than the database surface? Decomposition is used because it enables a way of selecting feature points with approximately equal density, regardless of total surface area. How well does decomposition achieve this goal?
- How effective are they when the current surface is substantially larger than the database surface? This can happen in a number of situations. One example is when a robot only partly enters a room the first time, but moves well inside on the second occasion.

7.1.6 A Curious Result

An interesting result is achieved when using the tentative patch matching combined with Sinkhorn normalisation (described as part of Gold *et al.*'s algorithm). Section 7.1.3 suggested combining the patch match results with the point-to-point distances within the error function used by Gold *et al.*'s method. What if the point-to-point distance was excluded from the equation entirely, and only the result from computing the Shape Distribution matches is used?

Construct a matrix, $\{q_{jk}\}$, as follows:

$$q_{jk} = patchMatchError_{jk} \tag{7.3}$$

where: $patchMatchError_{jk}$ is the patch match error calculated between patch j from the first model and patch k from the second model. This is to be used in a similar fashion to matrix $\mathbf{Q}$ in Algorithm 5.

For some large β, take the exponent:

$$m_{jk} = \exp\left(-\beta q_{jk}\right) \tag{7.4}$$

and then apply Sinkhorn normalisation to matrix $\mathbf{m}$. This produces a matrix with entries close to 0 or 1 and can be used to extract a consistent patch correspondence matrix. Now use the extended version of Walker *et al.*'s method, described in Section 4.5, to calculate the rotation matrix, $\mathbf{R}$, and translation vector, $\vec{T}$.

Applying this method to the task of registering surface *NoisyScene* onto surface *SmoothScene*, yielded a fit with RMS error of 0.116 m. This is comparable to the best results achieved using DMARA and , better than any of the results found using the RANSAC approach, as are shown in Figure 6.7(a). This is a curious result, perhaps worthy of further investigation. However, caution must be taken because, as it stands, this approach will not handle partial matches well.

References

[1] Large geometric models archive. Internet WWW page, `http://www-static.cc.gatech.edu/projects/large_models/index.html` (last accessed 21 Feb 2006). Georgia Institude of Technology.

[2] Data archive. Internet WWW page, `http://www.graphics.stanford.edu/data/` (last accessed 21 Feb 2006). Stanford Computer Graphics Laboratory.

[3] M. Agrawal and L. Davis. Trinocular stereo using shortest path and the ordering constraint. *International Journal of Computer Vision*, 47:43–50, 2002.

[4] M. Agrawal, A. Motilal, and L. S. Davis. A probabilistic framework for surface reconstruction from multiple images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[5] R. Allègre, A. Barbier, E. Galin, and S. Akkouche. A hybrid shape representation for free-form modeling. In *International Conference on Shape Modeling (SMI'04)*, 2004.

[6] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms (SODA '90)*, pages 129–137, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[7] K. Arun, T. Huang, and S. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:698–700, 1987.

[8] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume 1, pages 705–708, 2002.

[9] J. T. Baker. Towards solving the correspondence problem with 3d visual landmarks. Master's thesis, School of Computing and Mathematical Sciences, University of Waikato, New Zealand, 2004.

[10] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pat. Anal. and Mach. Intel*, 14(2):239–256, February 1992.

[11] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematics Society*, 35:99–110, 1943.

[12] I. M. Boier-Martin. Domain decomposition for multiresolution analysis. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP '03)*, pages 31–40, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[13] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An atlas framework for scalable mapping. In *International Conference on Robotics and Automation*, 2003.

[14] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 211–217, San Mateo, CA, 1990. Morgan Kaufmann.

[15] R. Campbell and P. Flynn. Eigenshapes for 3d object recognition in range data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, June 1999.

[16] R. J. Campbell and P. J. Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81:166–210, 2001.

[17] D. Carnegie, M. Cree, and A. Dorrington. A high resolution full-field range imaging system. *Rev. Sci. Instr.*, 76:083704, 2005.

[18] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: an experimental study. In *Proceedings of the eleventh annual symposium on Computational geometry (SCG '95)*, pages 297–305, New York, NY, USA, 1995. ACM Press.

[19] B. Chazelle, D. P. Dobkina, N. Shourabourab, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry*, 7(5–6):327–342, April 1997.

[20] Chetverikov. The trimmed iterative closest point algorithm. In *International Conference on Pattern Recognition*, 2002.

[21] C. S. Chua and R. Jarvis. Point signatures: A new representation for 3D object recognition. *Int. J. Comput. Vision*, 25(1):63–85, 1997.

[22] V. Chvatal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[23] D. Cobzas, H. Zhang, and M. Jagersand. Image-based localization with depth-enhanced image map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taiwan, May 2003.

[24] G. Cox and G. de Jager. A survey of point pattern matching techniques and a new approach to point pattern recognition. In *Proc. of South African Symposium on Communications and Signal Processing*, pages 243–248, 1993.

[25] F. Dellaert, W. Burgard, D. Fox, and S. Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, June 1999.

[26] A. Dempster, A. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[27] Y. Deng, B. Manjunath, and H. Shin. Color image segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, June 1999.

[28] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300, 2002.

[29] D. Eggert, A. Lorusso, and R. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9:272–290, 1997.

[30] A. Elfes. *Occupancy Grids: A probabilistic framework for robot perception and navigation.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[31] G. Farin. *Curves and Surfaces For CAGD.* Academic Press, San Diego, third edition, 1993.

[32] D. Fender and B. Julesz. Extension of panum's fusional area in binocularly stabilized vision. *JOSA*, 57(6):819–830, June 1967.

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

[34] A. Fitzgibbon. Robust registration of 2D and 3D point sets. In *Proceedings of the British Machine Vision Conference*, pages 662–670, 2001.

[35] S. Gold, C. P. Lu, A. Rangarajan, S. Pappu, and E. Mjolsness. New algorithms for 2D and 3D point matching: Pose estimation and correspondence. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 957–964. The MIT Press, 1995.

[36] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, April 1996.

[37] J. Gregor and R. T. Whitaker. Indoor scene reconstruction from sets of noisy range images. *Graphical Models*, 63:304–332, 2001.

[38] J.-S. Gutmann. *Robuste Navigation Autonomer Mobiler Systeme*. Arkademishe Verlagsgesellshaft Aka, Berlin, Germany, 2002.

[39] J.-S. Gutmann and B. Nebel. Navigation mobiler roboter mit laserscans. *Autonome Mobile Systeme*, 1997.

[40] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[41] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th ALVEY vision conference*, pages 147–151, 1988.

[42] W. Hoff and N. Ahuja. Surfaces from stereo: integrating feature matching, disparity estimation, and contour detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):121–136, February 1989.

[43] M. E. Jefferies, W. Weng, J. T. Baker, M. C. Cosgrove, and M. Mayo. A hybrid approach to finding cycles in hybrid maps. In *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australia, December 2003.

[44] A. E. Johnson and M. Hebert. Surface matching for object recognition in complex three-dimensional scenes. *Image and Vision Computing*, 16:635–651, 1998.

[45] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, 82:35–45, 1960.

[46] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.

[47] B. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(191–233), 2000.

[48] K. N. Kutulakos. Approximate n-view stereo. In *Proceedings of the 7th European Conference on Computer Vision (ECCV'00)*, pages 67–83, Dublin, Ireland, July 2000.

[49] C. Leung, B. Appleton, and C. S. Brian C. Lovel and. An energy minimisation approach to stereo-temporal dense reconstruction. In *17th International Conference on Pattern Recognition (ICPR'04)*, volume 4, Cambridge UK, August 2004.

[50] Z. Lin, J. Jin, and H. Talbot. Unseeded region growing for 3d image segmentation. In *CRPITS '00: Selected papers from the Pan-Sydney workshop on Visualisation*, pages 31–37, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[51] D. Lowe, J. Little, and S. Se. Global localization using distinctive visual features. In *Proceedings of the 2202 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, EPFL, Lausanne, Switzerland, October 2002.

[52] D. Lowe, J. Little, and S. Se. Mobile robotic localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21(8):735–758, August 2002.

[53] D. Lowe, J. Little, and S. Se. Vision-based mapping with backward correction. In *Proceedings of the 2202 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, EPFL, Lausanne, Switzerland, October 2002.

[54] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[55] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[56] I. Mahon and S. B. Willams. Three-dimensional robotic mapping. In *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australia, December 2003. Available at `http://www.araa.asn.au/acra/acra2003/` (last accessed 18 Feb 2006).

[57] A. P. Mangan and R. T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[58] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194:282–287, 1976.

[59] D. Marr and T. Poggio. A computational theory of human stereo vision. In *Proceedings of the Royal Society*, volume 204, pages 342–328, London, 1979.

[60] P. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, Inc., 1979.

[61] B. Messmer. *GMT–graph matching toolkit.* PhD thesis, University of Bern, 1995. Available at `http://iamwww.unibe.ch/~fkiwww/projects/GraphMatch.html` (last accessed 16 Jan 2006).

[62] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[63] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[64] K. Mueller, A. Smolic, P. Merkle, B. Kaspar, P. Eisert, and T. Wiegand. 3D reconstruction of natural scenes with view-adaptive multi-texturing. In *2nd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'04)*, pages 116–123, September 2004.

[65] A. Nüchter, H. Surmann, and J. Hertzberg. Automatic model refinement for 3d reconstruction with mobile robots. In *Fourth International Conference on 3-D Digital Imaging and Modeling*, page 394, October 2003.

[66] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Trans. Graph.*, 21(4):807–832, 2002.

[67] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[68] G. Papaioannou, E.-A. Karabassi, and T. Theoharis. Segmentation and surface characterization of arbitrary 3D meshes for object reconstruction and recognition. In *International Conference on Pattern Recognition*, volume 1, page 1734, 2000.

[69] M. Pollefeys. Visual 3D modeling from images. Internet WWW page, `http://www.cs.unc.edu/~marc/tutorial/tutorial02.html` (last accessed 20 Feb 2006), 2002. University of North Carolina.

[70] M. Pollefeys and L. V. Gool. From images to 3D models. *Commun. ACM*, 45(7):50–55, 2002.

[71] M. Pollefeys, L. V. Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[72] M. Pollefeys, R. Koch, M. Vergauwen, and L. V. Gool. Hand-held acquisition of 3D models with a video camera. In *Second International Conference on 3-D Imaging and Modeling (3DIM'99)*, page 14, October 1999.

[73] J. Porta, J. Verbeek, and B. Kröse. Active appearance-based robot localization using stereo vision. *Autonomous Robots*, 18(1):59–80, January 2005.

[74] T. Rofer. Using histogram correlation to create consistent laser scan maps. In *IEEE International Conference on Intelligent Robotic Systems*, 2002.

[75] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 1998 IEEE International Conference on Computer Vision*, pages 59–66, Bombay, India, 1998.

[76] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1067–1073, 1997.

[77] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, London, 1982.

[78] H.-Y. Shum, M. Hebert, and K. Ikeuchi. On 3D shape similarity. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 526, 1996.

[79] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, 3:876–879, 1987.

[80] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robotic Vehicles*, pages 167–193. Springer-Verlag, 1990.

[81] R. C. Smith and P. Cheeseman. One the representation and estimation of spatial uncertainty. Technical Report TR 4760 & 7239, SRI, 1985.

[82] S. N. Srihari. Representation of three-dimensional digital images. *ACM Comput. Surv.*, 13(4):399–424, 1981.

[83] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[84] G. Taubin. Estimation of planar curves, surfaces, and nonplanar spcace curves, defined by implicit equations with applications to edge and range image segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 13, pages 1115–1138, 1991.

[85] D. Terzopoulos and T. McInerney. Topology adaptive deformable surfaces for medical image volume segmentation. *IEEE Transactions on Medical Imaging*, 18(10), October 1999.

[86] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[87] S. Thrun. Robotic mapping: A survey. Technical Report CMU-CS-02-111, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2002. Available at `http://reports-archive.adm.cs.cmu.edu/cs2002.html` (last accessed 17 Feb 2006).

[88] S. Thrun, D. Haehnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. R. L. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 2003.

[89] S. Thrun, C. Martin, Y. Liu, D. Hähnel, R. Emery Montemerlo, C. Deepayan, and W. Burgard. A real-time expectation maximization algorithm for acquiring multi-planar maps of indoor environments with mobile robots. *IEEE Transactions on Robotics and Automation*, 20(3):433–442, 2003.

[90] N. Tomatis, I. Nourbakhsh, and R. Siegwart. Hybrid simultaneous localization and map building: Closing the loop with multi-hypothesis tracking. In *International Conference on Robotics and Automation*, 2002.

[91] T. Tung and F. Schmitt. Augmented reeb graphs for content-based retrieval of 3D mesh models. In *International Conference on Shape Modeling and Applications*, pages 157–166, 2005.

[92] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of Computer Graphics, Annual Conference Series, ACM SIGGRAPH*, pages 311–318, Orlando, Florida, July 1994. Software available at `http://graphics.stanford.edu/papers/zipper/` (last accessed 18 May 2005).

[93] B. van Wyk and M. van Wyk. A pocs-based graph matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):1526–1530, February 1989.

[94] E. Wahl, U. Hillenbrand, and G. Hirzinger. Surflet-pair-relation histograms: A statistical 3D-shape representation for rapid classification. In *Fourth International Conference on 3-D Digital Imaging and Modeling*, page 474, Banff, Alberta, Canada, 2003.

[95] M. W. Walker, L. Shao, and R. Volz. Estimating 3D location parameters using dual number quaternians. *CVGIP: Image Understanding*, 54(3):358–367, 1991.

[96] S. S. Wong and K. L. Chan. Multi-view 3D model reconstruction: Exploitation of color homogeneity in voxel mask. In *Third International Conference on Image and Graphics (ICIG'04)*, December 2004.

[97] S. Yamany and A. Farag. 3D objects coding and recognition using surface signatures. In *International Conference on Pattern Recognition*, volume 4, page 4571, 2000.

[98] W. Yeap and M. Jefferies. Computing a representation for the local environment. *Artificial Intelligence*, 107(2):265–301, 1999.

[99] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Computers and Graphics*, 26(5):733–743, October 2002.