INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
MULTISENSORY OBJECT RECOGNITION
AND
TRACKING FOR ROBOTIC APPLICATIONS

by

LARS JONAS OLSSON

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Advisor: Dr. Sheldon Gruber

Department of Electrical Engineering and Applied Physics
Case Western Reserve University
May 1995
MULTISENSORY OBJECT RECOGNITION
AND
TRACKING FOR ROBOTIC APPLICATIONS

CASE WESTERN RESERVE UNIVERSITY
GRADUATE STUDIES
Feb. 17, 1995
We hereby approve the thesis of
LARS JONAS OLSSON
candidate for the Doctor of Philosophy
degree. *

Committee Chair

Dr. Sheldon Gruber
Thesis Advisor
Department of Electrical Engineering and Applied Physics

Committee

Dr. Francis Merat
Department of Electrical Engineering and Applied Physics

Committee

Dr. Wyatt Newman
Department of Electrical Engineering and Applied Physics

Committee

Dr. Leon Sterling
Department of Computer Engineering & Science

*We also certify that written approval has been obtained for any proprietary material contained therein.
I grant to Case Western Reserve University the right to use this work, irrespective of any copyright, for the University's own purposes without cost to the University or to its students, agents and employees. I further agree that the University may reproduce and provide single copies of the work, in any format other than in or from microforms, to the public for the cost of reproduction.

[Signature]
Abstract

by

LARS JONAS OLSSON

A vision system which uses 3D sensory information as well as image intensity as input data for object recognition algorithms is a major component of a mobile robot guidance system. The vision system uses a special 3D range-finder that generates registered intensity and range images. From these images the fold, jump, and reflectivity edges in the scene are detected. The different edge types are fused into one edge image, forming the base for an edge-based segmentation technique. The resulting segments in the scene are then grouped into object regions, each likely to come from the same object. Hypotheses for objects are formed for each object region based on constraints on surface features and constraints between surfaces. Each hypothesized object's location and orientation is estimated and the hypotheses are verified. The verification is aided by an accurate simulator of the real range-finder. The simulator, using ray-tracing techniques, provides the range and intensity images that would be generated by the real range-finder. This verification can be done using only the sensed data. Furthermore, the simulator can be incorporated in an object tracking system necessary for a mobile robot. The initial hypotheses are updated as the robot moves through the environment, and the simulator can be used to generate the expected scene from each new mobile robot position. This simulated data can also be used to segment the real range-finder data in a model-driven manner and to select areas of the scene for selective scanning and processing.
ACKNOWLEDGMENTS

First of all I have to thank my advisor Dr. Sheldon Gruber for his support and suggestions during the course of this work.

I also like to thank the other students that have helped in parts of this project: Robert Delvalle, Daniel Karnade, Geoff Langos, Thomas Mueller, Scott Stiefel, and Leda Villalobos.

Thanks to the Center for Automation and Intelligent Systems Research (CAISR) and Electrical Engineering and Applied Physics department for their financial support.

Most of all, however, I have to thank my family whose patience made this work possible.
Contents

1 INTRODUCTION .................................................. 1
  1.1 Problem Statement ........................................... 3
  1.2 Related Work .................................................. 6
  1.3 Approach Developed in this Thesis ......................... 9

2 OUTLINE OF THESIS ........................................... 11
  2.1 Object Representation ...................................... 11
  2.2 Feature Extraction ......................................... 12
  2.3 Object Regions ............................................. 13
  2.4 Model Indexing ............................................. 14
  2.5 Hypothesis Generation ..................................... 16
  2.6 Hypothesis Verification and Pose Refinement ............. 18
  2.7 Scene Prediction ........................................... 19
  2.8 Implementation ............................................ 20

3 SENSORS ..................................................... 23
  3.1 Range-Finder ............................................... 23
    3.1.1 Accuracy ............................................... 27
    3.1.2 Characterization ....................................... 29
  3.2 Range-Finder Simulator ................................... 30
7.2 Scenes .............................................. 77
7.3 Recognition Results .................................. 80
  7.3.1 Real Scenes ....................................... 87
  7.3.2 Evaluation .......................................... 102

8 CONTRIBUTIONS ........................................... 109

9 FUTURE WORK ........................................... 110

BIBLIOGRAPHY ............................................. 111

APPENDIX I: DERIVATIVE CALCULATIONS ................. 116

APPENDIX II: USER'S GUIDE ............................. 119
  The Recognize Program ................................ 121
  The rangeSegm Program .................................. 124
  The RangeCalibrate Program ............................. 128

APPENDIX III: SOFTWARE ................................. 132
List of Figures

1  Edge types for object representation .................... 12
2  Illustration of occlusion ............................... 18
3  Block diagram of system ............................... 21
4  Front view of the range-finder .......................... 24
5  Geometry of triangulation plane ......................... 25
6  Horizontal Motor Compensation for Several Scenes ........ 28
7  The dependency of range-finder's $\sigma_x$ on $x$ .............. 31
8  The dependency of range-finder's $\sigma_x$ on intensity return level .... 31
9  Intensity Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below. ........... 39
10 Jump Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below. ............. 40
11 Curvature Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below. ............. 43
12 Fused Edge Processing using Euclidean Distance Transforms. The steps are indicated on the top line of each box and the class name is in parentheses below. ............. 45
13 Segmentation. The steps are indicated on the top line of each box and the class name is in parentheses below. ....................... 48
Example of Edge Image and edge and vertex point sets used for
Edge Map creation ............................................. 49

Example Edge Map ............................................. 51

Example Loop Map ............................................. 54

Example of a simple Segment Map Hierarchy ................. 56

Segmentation example. (a) intensity image (b) jump edges (c) con-
cave edges (d) convex edges .................................. 58

Segmentation example. (a) fused edges (b) segments .......... 59

Example Model and Object Region ......................... 65

Example of search in hypothesis generation step. Matching the a
cube model and an object region from an octagonal object. .... 67

Simulator Data Processing .................................... 73

The Model library (a) PaperRoll (b) Block .................... 77

The Model library (a) OctCup (b) OctPlate ................. 78

The Model library (a) Bone (b) Arc ......................... 78

The Model library SoapBox .................................... 79

The cylinder scene (a) intensity image (b) range image .... 80

The octCup scene (a) intensity image (b) range image .... 81

The boneArc scene (a) intensity image (b) range image .... 82

The octArc scene (a) intensity image (b) range image .... 83

The octArcBone scene (a) intensity image (b) range image .. 84

The plateCup scene (a) intensity image (b) range image .... 85

The plateSoapArc scene (a) intensity image (b) range image .. 86

The cylinder scene (a) fused edges (b) closed edges .......... 88
The segments and object regions in the cyledge scene. Each object region is shown in a different stipple pattern.

The octCup scene (a) fused edges (b) closed edges

The segments and object regions in the octCup scene. Each object region is shown in a different stipple pattern.

The boneArc scene (a) fused edges (b) closed edges

The segments and object regions in the boneArc scene. Each object region is shown in a different stipple pattern.

The octArc scene (a) fused edges (b) closed edges

The segments and object regions in the octArc scene. Each object region is shown in a different stipple pattern.

The octArcBone scene (a) fused edges (b) closed edges

The segments and object regions in the octArcBone scene. Each object region is shown in a different stipple pattern.

The plateCup scene (a) fused edges (b) closed edges

The segments and object regions in the plateCup scene. Each object region is shown in a different stipple pattern.

The plateSoapArc scene (a) fused edges (b) closed edges

The segments and object regions in the plateSoapArc scene. Each object region is shown in a different stipple pattern.

Main window for Recognize program

Main window for rangeSegm program

Oct.l scene before (a) and after (b) calibration

Average distance to Surface for Calibration Region for oct.l scene

Horizontal Compensation Profile for oct.1 scene

Directory tree for the Object Recognition software
# List of Tables

1. Execution times using an i486DX-33 processor with 16MB RAM and 256kB cache running 386BSD. .......................... 60
2. Example Object Hypothesis generated in Indexing step ............ 65
3. Recognition results for the cylledge1 scene ..................... 90
4. Recognition results for the octCup scene ........................ 92
5. Recognition results for the boneArc scene ....................... 93
6. Recognition results for the octArc scene ........................ 96
7. Recognition results for the octArcBone scene .................... 101
8. Recognition results for the plateCup scene ...................... 102
9. Recognition results for the plateSoapArc scene ................. 107
Chapter 1

INTRODUCTION

Self-guided mobile robots are an important goal for many potential applications, such as hazardous waste cleanup, service robots in hospitals, homes, and factories as well as for planetary exploration. In order to achieve mobile robot autonomy the environment in which the robot must navigate must be perceived. This thesis focuses on the use of computer vision applied to this problem and tries to define and evaluate algorithms that will lead to this capability.

Computer vision can be differentiated from image processing in that it uses image data to generate a symbolic description; whereas image processing uses image data to generate other image data. Image processing methods such as filtering and edge detection, however, are used in the early stages of processing for computer vision.

An important area in computer vision is image understanding which can be divided into scene analysis and object recognition. The goal of image understanding is to describe the scene in a symbolic form.

Scene analysis typically uses color cameras in order to recognize houses, streets, sky, rocks, ravines, bushes, and trees: natural objects in general. These objects are only described by typical characteristics such as: the sky is blue and
at the top of the image. The scenes are restricted to outdoor scenes where the
distance from the camera to the objects is large. In this case gross assumptions
about positions of objects are valid, e.g. sky is at the top of the image. One good
example of scene analysis is by Ohta [1], in which various color segmentation
algorithms and rule based reasoning systems are used to analyze images of houses,
trees, sky, and streets. One of the main research efforts in scene analysis is
the Strategic Computing Computer Vision project funded by DARPA (Defense
Advanced Research Program Agency) which included as a research goal the use
of autonomous land vehicles on roads and in open terrain.

Object recognition is more restricted than scene analysis. The models of the
objects are more specific and in general a model only matches one particular
instance of an object category, e.g. a particular kind of screw driver. Typically,
projects use black and white cameras and/or range-finders to recognize tools,
castings, toys, etc. The objects are either described by volume occupancy or their
bounding surfaces. A review of representations is given by Henderson [2]. This
more detailed description, however, makes it possible to recognize similar objects,
reason about geometry, and check constraints such as dimension, non-intersecting
parts etc. While much research has been done in object recognition, the problem
area is far from being resolved. Some problems encountered are:

- Noisy data and the relative unavailability of range data. Range finders have
  traditionally been very slow or expensive.

- The complexity of the programs for implementing the algorithms. This al-
  most eliminates the possibility of repeating previous work; therefore, makes
  it difficult to compare different systems. This also makes improving on one
  step in the process difficult because the whole system has to be rebuilt.

- Computational complexity, run-times range from minutes to hours.

- The research procedures in computer vision often make it impossible to
  compare different algorithms. See Haralick [3]. It is common to test the
  routines on only 1–10 scenes. Mostly algorithms are not described in detail
  and depend on thresholds that need to be set manually.
The goal of this project is the development of flexible and modular object recognition and tracking system. The object recognition technology used employs diverse methods derived from image processing, computer vision, computer science, pattern recognition and artificial intelligence. The image processing methods include edge detection, curvature estimation, connected region labeling, and segmentation. Methods from computer vision are surface fit, classification of regions based on curvature, matching between image and model features. Artificial intelligence techniques are used to guide the sequential matching of features. Finally while most systems for object recognition tend to be very large and have a complicated control structure, an effort has been made to keep the system manageable. It was deemed desirable to build the system in a manner that it can be run on a parallel computer. Effort was put into finding algorithms that makes parallelism possibly.

1.1 Problem Statement

In general the problem of object recognition is characterized by the questions: What objects are in the scene and where are they? This hardly can serve as a definition, so some more specific definitions are given below.

Fisher [4, chapter 2] states that:

Three dimensional object recognition is the identification of a model structure with a set of image data, such that geometrically consistent model-to-data correspondences are established and the object’s three dimensional scene position is known. All model features should be fully accounted for—by having consistent image evidence either supporting their presence or explaining their absence.

In the above definition only “a set of image data” is associated with a modeled object.

Besl [5] defines the 3-D object recognition problem as:
(1) Given any collection of labeled rigid solid objects, these objects may be examined in a way desired (automatically or manually) as long as the objects are not deformed.

(2) Models for the labeled objects may be formed using information from this examination in any way desired and given the appropriate object’s label.

(3) (i) Given digitized sensor data corresponding to one particular, but arbitrary, field of view of the real world as it existed at the time of data acquisition;
(ii) given any data stored previously during the model formation process; and
(iii) given a list of labels of distinguishable solid objects;
answer the following questions for each object in the list using only the capabilities of a single autonomous processing unit:
a) Does the given labeled object appear in the digitized sensor data?(That is, does it permit a consistent interpretation of the sensor data?)
b) If it does, how many times does the object occur?
c) For each occurrence of a given object, determine the location of that object within the sensor data and, if it is possible using the particular type of sensor data, determine the three-dimensional location of that object with respect to some convenient coordinate system.
d) Also, if possible, determine the three-dimensional orientation of each occurrence of a given object with respect to some convenient coordinate system.

(4) Finally, if there exist regions within the sensor data that do not correspond to any of the objects in the list, characterize these regions in a way that they might be recognized if they occur again in any subsequent images.

Another definition is by Feng [6]:

The identity of two objects requires relational isomorphism between the two relational data structures. If, however, an object has been partially observed, as is inevitable in three dimensions, then its recognition corresponds to the relational sub-isomorphism of its data structure with that of a member of the library. The problem of object recognition is therefore that of finding relational sub-isomorphism between
data structures. For a single pair of relations this is an example of a more general problem, the consistent labeling problem.

The definition of the problem in this thesis is based on that of Fisher. However Fisher's definition neither takes into account unmodeled objects nor the possibility that more than one model structure can be consistent with the image data. In general there are multiple objects in the scene which may create occlusion of one object by another. Self-occlusion due to non-convex object surfaces may also occur. The scene image may have missing data as a result of sensor limitations.

The problem domain for this thesis is limited to relatively simple but not strictly polyhedral objects. The objects can have creases and corners but should also have some curved surfaces. The surfaces do not have to be quadric, but there should be a mathematical model for each surface that accurately describes it. Also the mathematical models can be of different kinds for different surfaces, e.g. quadric, polygonal approximation, and interpolation models. Furthermore the system is intended for situations where the scenes are changing i.e. with moving robot where successive scenes are different due to object and/or robot movements, or industrial robot application where successive scenes change as new objects arrive and other objects are removed.

The goal is to generate a solution to the above problem in such a way that it can benefit from the redundancy of image sequences where the difference between successive images is small; and the source of the changes can be partly explained, e.g. if the change in viewpoint is approximately known or the velocities of the objects in the scene are known or may be reasonably estimated.

Identification of pairs between image data sets and object models in the database must be obtained. All instances of object models, that are detectable according to the sensor model in the image data, should be found. Each such image data set should be consistent with the object model and the sensor model.
The pairs should also be mutually consistent.

1.2 Related Work

Thorough reviews of object recognition are given by Besl and Jain [5], and Chin and Dyer [7]. Besl and Jain [5] deal with range image understanding, define the problem in detail, and have reviews of 3D object and surface representations, 3D object and surface rendering, range image formation and processing, 3D object reconstruction, and 3D object recognition. Chin and Dyer [7] cover model-based recognition in robot vision similar to Besl and Jain, but focus more on different matching techniques for various representations. A more recent review of progress in object recognition from range data is by Brady et. al. [8]. Related reviews are: 3-D representations [2], and use of multi-sensory images for object recognition [9].

The main divisions in object recognition are the dimensionality of the object models. There are 2 D, 2 1/2 D, and 3 D models. 2 D models can be used when there are only a few distinct views of the objects; each distinct view is represented as a separate model. One example is objects with only a few stable positions on a light table observed by a stationary camera. A 2 1/2 D model describes the geometry of visible surfaces from a particular viewpoint; and each object usually needs several models corresponding to significant viewpoints. There is a particular class of 2 1/2 D models that has been called characteristic views or aspects, Each aspect corresponds to a set of viewpoints from which the same surfaces of the object are visible. There is a so-called visual potential or aspect graph that is associated with the aspects and that describes the transitions between different aspects when the viewpoint is altered. Aspects have been automatically derived for general polyhedra under perspective projection. The most general model is the 3 D which describes the geometry of the whole object. One problem with 3 D models is
the following. The image data only give a 2 1/2 D description of the object; and it is computationally hard to determine from the 3 D model what parts of the objects surface should be visible from different viewpoints. 3 D models, see [2], can either describe the volume occupied by the object, e.g. constructive solid geometry (CSG), oct-trees, and generalized cylinders, or describe the bounding surface of the object (B-rep). As the image data describe the surface of objects, the B-rep models are more compatible than the volume descriptions.

The matching between object and model data is an NP-complete sub-isomorphism problem. Heuristics are therefore needed to guide the search and prune the search tree. Graph matching is the method most frequently used. The method uses different constraints to prune the search. Typical constraints are topological (connectedness), property (size, surface shape), and geometrical (the consistent viewpoint constraint; the transforms between scene and model features should be the same for all features). A tree search for matches is described by Grimson [10]. Another method is a Hough transform that computes the transformation between scene and model features. The most prominent cluster in the transform space is taken to imply the proper match. This method has the advantage that all data is treated equally and the ease of implementation on a parallel processor. The disadvantage is its complexity if there is a complex scene or a large object library. A good example of this method is by Silberberg et. al. [11].

Fan, Medioni, and Nevatia [12, 13] have developed curvature based surface segmentation routines which are used successfully for object recognition using range data. A graph is constructed from the scene with surface patches as nodes and relations between them as links. This graph is then matched with multi-view models of different objects. The matching is controlled by a best first graph search. The limitations of the method are the multi-view model and the scene is directly matched with all models. Multi-view models means that each object in the model
data base is described as a number (4 to 6) of completely independent models describing the surfaces seen from different viewpoints. The advantage of this model is that it is directly compatible with the scene. With large model databases it is important to have an intermediate step that selects a few candidate models out of the database. This intermediate step has been called model invocation [4] or prediction hierarchies [14].

The system described by Fisher [4] is interesting in that it is very rigorous and verifies several constraints during the recognition. It also incorporates the model invocation step which sorts out a few interesting models. The models used are surface based descriptions of the whole object. Its also possible to specify degrees of freedom in the model such as in the model describing a puma robot. Visibility groups which describe what surfaces are visible from different viewpoints (compare aspects, characteristic views) are associated with each object model. However, the system starts with hand-segmented artificial range data. The model base only contains three models (robot, trash can, and chair); and the system is only tested with one test image.

An interesting approach to object recognition for a bin-picking system has been developed at Carnegie Mellon [15]. This differs from an object recognition system in that it seeks a single object model and the distance from the sensor to the object is always approximately the same. The system uses sensor models to predict the detectability and reliability of features. This information is used to automatically generate recognition programs. The recognition has two stages: determining the correct aspect and determining the precise pose. The first step is performed by an interpretation tree that uses surfaces found in range data. No occlusion is tolerated. The second step uses edges from both range and intensity images to refine the pose.

The use of both range and intensity data has been described by Duda, Nitzan,
Barrett [16]. Their system is not a range fusion system, but rather a system that uses range and intensity for different purposes. An object recognition system was described in [17]. Another system is described by Hu and Stockman [18]. A review of using multi-sensor images to derive the structure of 3D objects is given by Magee and Aggarwal [9]. There have also been object recognition systems using several cameras to eliminate some occlusion and to get range data.

1.3 Approach Developed in this Thesis

Intensity and range data is used to increase the flexibility of the system, to speed up the model indexing, and to increase the reliability of the derived features. Both the scene and the object models are described in terms of the bounding surfaces (B-rep).

There are several significant differences between the system developed here and what has been done by others:

- New type of auto-synchronized 3D range-finder.
- New efficient calculation of range derivatives used to find jump and fold edges.
- Characterization of the range-finder and logical sensors based on range-finder data.
- New way of fusing edge images using Euclidean distance transforms.
- New efficient way of edge map tracking using hashed sets.
- The system is specifically designed to use the redundancy in sequential images. At the present, update time is likely to be 1–10 minutes; but with refined technology and algorithms, it should soon be possible to perform updating in real time.
- Scene prediction is used to facilitate sequential processing. Given a first interpretation of the scene and the movement of the sensor the appearance of the scene can be predicted. Then the new aspect for each object which takes into account the new viewpoint is predicted and matched.
• Modularity of implementation: This is to be important as implementations of object recognition and scene interpretation projects are very large and often specific to particular sensors and other conditions. It is easier to implement this system and to change the conditions under which it works since it uses a modular design that has specific modules for sensor model, environment model, and each object model.

• The best parts of previous projects [13, 4] have been integrated after eliminating their weaknesses such as: poor models [13] and little or no reliance on real data [4].
Chapter 2

OUTLINE OF THESIS

This thesis concerns itself with computer recognition of man-made objects; specifically industrial parts, boxes, canisters, walls, doors, tools, and other relatively simple objects that have dimensions suitable for the current range finder. It uses multi-sensor data, namely range and intensity. A general discussion of the various steps in the recognition process is given in the following sections. Details are then presented in later chapters.

2.1 Object Representation

In systems that use range data, the main features are surfaces and the edges of these surfaces. The latter fall into one of the following classes: (see figure 1.)

Class 1: Jump edges—these are found at range discontinuities;

Class 2: Creases, folds, and corners which are at discontinuities of first order partial derivatives of range; and

Class 3: Smooth transitions which occur between planes and cylinders, etc. which exist at discontinuities of second order partial derivatives of range.
Figure 1: Edge types for object representation

The most appropriate class of edges for the modeling of the object is class 3. The object can be modeled with quadratic surfaces and there are edges at the points where the surface equation changes. However, for feature extraction, the easiest to determine are class 1 edges and the hardest are class 3 edges. This is due to the higher noise sensitivity of the higher order derivative estimations. The extraction of edges can be improved by fusing range and intensity information.

Class 2 edges are selected as the most appropriate for the data driven feature extraction. However, class 3 edges can be extracted in the matching and refinement stages if necessary.

2.2 Feature Extraction

In order to reason and recognize objects from the sensory data, features have to be extracted. The features used are surfaces and properties and relations between
them. For segmentation of the scene into regions corresponding to surfaces there are two main strategies:

1. Find the bounding edges of a region and declare the interior as the region.
2. Use the smoothness of the interior to grow the region out to its edges.

The most intuitive method is strategy 1, but since edges are hard to detect this often leads to gaps in the edge bounding a surface. Strategy 2 is only indirectly related to the definition of the surfaces. This leads to a problem as the edges produced by this method are somewhat displaced from the real edges. This occurs as the growth is stopped when some error rises above a threshold, at which time the edges have been passed. The best practical implementation is believed to be a combination of the two methods. Based on the fusion of range and intensity data, edge detection extracts the main part of the surfaces’ boundaries. The edge map can then be cleaned up using sensor model derived rules, very short gaps in edges can be filled in, and very short isolated edge segments can be deleted. The edge map still may contain breaks.

Future improvements could include a region-growing algorithm based on the scene and the edge map to close these edges. An example of integration of edge detection and region growing is described by Pavlidis et. al. [19]. The region growing should not be allowed to proceed across edges in the edge map. Edge detection and refinement of the edge map has been described by Fan et. al. [12] who only used range images.

2.3 Object Regions

To reduce the complexity of the problem it is helpful to initially group segments that are part of the same object into larger regions called object regions. However.
this requires that the objects be already recognized. Some simple rules can yield an approximation. These are:

- Jump edges are assumed to separate different objects,
- Convex edges are very unlikely to separate different objects, and
- Concave edges are likely to be object boundaries.

With these rules it is possible to derive a grouping where each segment has high probability of being formed by surfaces from only one object, however it is also likely that some objects correspond to multiple object regions.

The advantages of using object regions are:

1. The features in one object region are likely to have come from only one object, and as such are unlikely to be spurious in regard to that object's model. This means that the matching of the features in the object region does not have to be matched with NULL (no match), which would greatly increase the complexity of the search.

2. A smaller number of features is used greatly reducing the complexity of the search.

2.4 Model Indexing

Model indexing is a data driven algorithm to sort out the most promising models for a set of image data. Then one of these models is invoked to determine if the data match the model. The model invocation step can be eliminated by sequentially trying all models in the database, however, this is not practical if the number of models is large. Indexing can be seen as the main recognition step, the others being used for either preprocessing or verification.

Recently a number of articles concerning object recognition from large model data bases have been published in which either an explicit or an implicit model invocation process is involved.
Fisher [4] uses an evidence calculating network. Property evidence and spatial configuration are used to form plausibility values for the image data corresponding to the different models. When a plausibility value exceeds a fixed threshold, the corresponding model is invoked.

Bolle et. al. [20] describes a concurrent and layered parameter network approach. This approach is inspired by connectionist methods. The network extracts planar patches, quadrics of rotation patches, and the edges between these patches, and performs the model indexing step. The claim is made that the method is suitable for sensor fusion.

Burns and Kitchen [14] use prediction hierarchies for 3D object recognition from intensity scenes. The prediction hierarchies generate a 2D model so the scene data can be directly matched with compatible data.

Swain [21] used decision trees for recognition. The decision trees were based on the information theory concept of entropy. This approach can be compared to the ID3 method of Quinlan [22]. Errors in the decisions are handled by calculating a confidence for the state. The branch can be terminated when the plausibility falls below a plausibility threshold or upon arrival to a leaf node. The search then resumes in another part of the tree.

The approach for indexing proposed here is different from the methods mentioned above, it is divided into two steps. The method first uses indexing to find the most likely object models, and then finds the most likely aspect. This simplifies the generation of indexing algorithms and reduces the computational complexity. The first step generates a set of possible model objects. The second step selects one of these object models and generates a set of aspect models. The sets are implicitly represented as the open nodes in the interpretation tree. Selection of what object or aspect model to evaluate is based on an evidence measure.

Due to the complexity of calculating the object aspects, the aspect indexing
was never implemented. Instead a simpler method based on individual surface properties was implemented for the testing of the system, this method will be described in chapter 5.

2.5 Hypothesis Generation

For both computational and implementation reasons the matching is divided into two steps: hypothesis generation and hypothesis verification. The first step is to find a matching between object and model features with the unary and binary constraints satisfied. This generates the hypothesis that is later verified in the hypothesis verification step. The division is created since unary and binary constraints are easy to evaluate and can prune incorrect matches. Hypothesis generation is implemented as a depth first tree search which is guided by property and topological constraints. Global constraints are not incorporated during the hypothesis generation since they require geometric reasoning which is computationally intensive. Therefore evaluation is deferred until there is substantial evidence for the model. The reasoning is further simplified when a pose for the object has already been determined. This later step can also merge object regions that are separated due to segmentation or occlusion.

An interesting matching procedure is described by Wu [23]. The procedure uses surface equations for matching. It also calculates the feasible centroid set, which is the set of object centroids that are consistent with the match. This is different from geometric reasoning methods which calculate the feasible transform set. The feasible transform set includes the feasible centroid set and the feasible rotations of the object. Wu's system calculates general quadric surface parameters, but only a subset of the features (sphere, circle, point, cylinder, line, and plane) can be used to calculate the feasible centroid set. A more specific feasible centroid
set is calculated for combinations of matched surface pairs.

For complex feature types, such as curves and curved surfaces, the abstract representation should be deferred until the curve or surface is used in matching. This eliminates the heuristic choice of what kind of representation should be used and also improves the computational efficiency because a complex surface representation may not be necessary at all, e.g. when matching a curve with a spline function, it may not be necessary to calculate the curve’s spline representation. It corresponds to making one of the following decisions:

1. The curve matches the spline function; or
2. The curve is an instance of the spline function.

The difference between 1 and 2 is that given the sensor data and the sensor models the decision 1 can be done without any uncertainty, whereas the decision 2 involves uncertainty and might later have to be undone. Therefore by restricting the answers to “Taking the sensor uncertainty into account, it might be this object”, instead of saying “It is probably this object”. This also makes the reasoning monotonic instead of non-monotonic. This works if the probability density functions have local support, e.g. uniform distribution and not normal distribution.

Therefore, the problem can be posed as monotonic by using uniform distribution in the sensor model and having model driven feature extraction. The effect is that the system answers the question “What objects could it possibly be?” instead of “What object do you think it is?”. This means that there might be multiple solutions. The sensor model therefore needs to be accurate enough so:

1. no feasible solutions are considered infeasible, and
2. not too many infeasible solutions are considered feasible.

Occlusion in the scene can, in most cases, be detected from the range discontinuities; the surfaces at the larger distance at a range discontinuity is marked as
Figure 2: Illustration of occlusion

occluded so the surface can still be used for matching based on local features. For instance when a polyhedral face \{e1, e2, e3, e4\} is occluded by the edge e6, the representation for the occluded face will be: \{ e1, e5, e6, e7, e4\}, e6 is an occluding edge, edges e5, e7 are occluded. In this case it is still possible to match the surface, see figure 2.

2.6 Hypothesis Verification and Pose Refinement

The constraints that were not evaluated in the hypothesis generation step have to be evaluated in this step. Geometric reasoning using global constraints is necessary in this step. Basically the process is to see if a solution exists that satisfies the coordinate system transformation constraints generated by each feature in the hypothesis. Constraints used by Fisher [4] are: two scalars close in value, two points are close in location, two vectors in nearly the same direction, a transformation links a pair of points, a transformation links a pair of vectors, etc. The constraints are mostly non-linear; and the methods used to solve them are:
sup-inf method [24, 4], constraint networks [4], and Gröbner basis [25]. A more restrictive method which is a more efficient way is described by Wu [23]. In the sup-inf method, the feasible region is expressed as a polyhedron. A simpler region expression is the ellipsoid used in the Soviet ellipsoid method[26].

Also in the geometric reasoning process the result is that there exists a coordinate system transformation but there is no guarantee that this is the best solution. For some robotic applications there is a need to calculate the best estimate. This can be done in a least square sense using quaternions, see Faugeras[27].

### 2.7 Scene Prediction

The system is able to use the redundancy in sequential images. Using aspects and aspect graphs makes it easy to access information about when the appearance of an object will change and in what way it will change. The transition from one aspect to a neighboring aspect is called a visual event. Neighboring aspect means that the aspect can be reached without passing any intermediate aspect by continuously changing the viewpoint. The aspects contain information about the set of viewpoints from which this aspect can be seen. In the aspect graph, nodes correspond to aspects and arcs to visual events.

After the initial interpretation of the scene, there are pairs between image data sets and object models. Each pair can be incorrect, as objects can be perceived as identical when there is missing data. The new aspect can be predicted by using the estimated pose, the change in sensor position, the change in object position, and the model aspect. As each aspect corresponds to a region of viewpoints the prediction can tolerate some errors in the estimate of sensor position and object pose. Then the predicted aspect is matched with the image data. If the match fails, this can be because:
• the object model is not the right one,
• the aspect model is not correct, and
• the errors in sensor and object positions are too large.

New aspects may be generated to improve upon the possible errors by:
• Using the previously generated set of object models to generate a new aspect;
• Choosing the current object model to generate a new aspect;
• Choosing an aspect from the aspects neighboring to the predicted aspect; and
• Doing a complete indexing to find a new aspect.

2.8 Implementation

The software component of the implementation is client/server based, which facilitates the communication between different algorithms and makes the system modular. There are currently two servers, the Range Finder server and the Range Finder Simulator Server, see figure 3.

The main modules of the system are:

1. Sensor.
   Laser Spot Range Finder. The range finder is controlled by a Zenith 386/20 with controller cards for the stepping motors, and a TI 320C25 DSP board with custom interface to a line camera for acquisition and processing of the intensity data. The range-finder is controlled by the Range Finder Server.

2. Feature Extraction.
   The main features used for object recognition are surfaces, edges, and points. These features are derivable from the scene and are also derivable from the object models. The object model surfaces correspond to regions without discontinuities in the first derivative of range.

3. Object indexing module for determining the candidate objects for a scene region.

4. Graph matching, this module determines if the candidate object matches the scene region.
Figure 3: Block diagram of system
5. Global hypothesis evaluation.
   This module assures that the different local hypotheses are consistent.

   Based on the sensors used and information about pose and orientation and
   possibly speed the appearance of the new scene is predicted. The features
   that are detectable for of each object are predicted as well as the position
   where they appear.

   To facilitate the modularity needed for the distribution of tasks to the multi-
   processors the software is written in object oriented programming language. The
   chosen language is C++.
Chapter 3

SENSORS

The system uses a range-finder that generates registered intensity and range images. This range-finder, its accuracy, and characterization are described in this chapter. To support the verification and tracking procedures a range-finder simulator has been developed, and is briefly described. Finally, the derivation of the logical sensors are given.

3.1 Range-Finder

The range-finder is based on the triangulation principle in which the length of one side and the adjoining angles are known. This is enough information to calculate the 3D coordinates of the opposing vertex. In practice the equations are not quite this simple and will be derived here. The front view of the range-finder is shown in figure 4.

The baseline of the triangle coincides with the axis of the vertical deflection rod, and the two end points of the baseline are the intersection of the laser beam and the scanner’s vertical mirror and the point where the ray from the laser spot in the scene to the camera’s front focal point intersects the camera’s vertical deflection
mirror. Both of the base-line’s end-points move. The point at the laser end moves with different horizontal deflection of the scanner, and the point at the camera end moves with different object distances. The geometry of the triangulation plane is shown in figure 5.

The formula from which the range calculation is derived is

\[ B_0 = x_r \tan(\beta_c) - x_r \tan(\beta_l) + d_c \tan(\beta_c) - d_l \tan(\beta_l) \]  

(1)

Where \( B_0 \) is the nominal baseline, \( \beta_c \) is the camera angle, \( \beta_l \) is the horizontal deflection scanner angle\(^1\), and \( d_c \) and \( d_l \) are the camera and scanner focal point offsets respectively. From this equation the coordinate \( x_r \) in the triangle plane can be found.

\[ x_r = \frac{B_0 - d_c \tan(\beta_c) + d_l \tan(\beta_l)}{\tan(\beta_c) - \tan(\beta_l)} \]  

(2)

When rotating the triangulation plane around the z axis the following equations are obtained, where \( \alpha_l \) is the vertical scanner angle.

\[ x = x_r \cos(\alpha_l) \]

\(^1\)The deflection angle is twice the motor shaft angle
Figure 5: Geometry of triangulation plane
y = x_r \sin(\alpha_l) \\
z = (x_r + d_l) \tan(\beta_l) \tag{3}

The camera angle $\beta_c$ is divided into a fixed portion $\beta_{0c}$ defined by the viewing angle of the camera and the relative angle $\beta_{rc}$ measured by the camera.

$\beta_c = \beta_{0c} + \beta_{rc}$

To facilitate the on-line computation of the coordinates the following equation can be used:

\[
x_r = \frac{n_{x00} + n_{x01} \tan(\beta_l) + n_{x10} \tan(\beta_{rc}) + n_{x11} \tan(\beta_l) \tan(\beta_{rc})}{d_{00} + d_{01} \tan(\beta_l) + d_{10} \tan(\beta_{rc}) + d_{11} \tan(\beta_l) \tan(\beta_{rc})}
\]

Where the coefficients can be pre-computed and $\tan(\beta_l)$, $\cos(\alpha_l)$, and $\sin(\alpha_l)$ are stored in a lookup tables.

The tangent of the relative camera angle $\tan(\beta_{rc})$ is calculated from the camera pixel coordinate $u_c$ and the camera focal length $f$.

\[
\tan(\beta_{rc}) = \frac{u_c}{f} \tag{4}
\]

The camera coordinate $u_c$ is estimated by calculating the centroid of a window around the pixel with maximum intensity. The intensity return, $I$, is defined as the sum of pixel intensities within this window. A line camera with 256 pixels is used, which makes the data acquisition time considerably less than that of an area array camera.

The particular dimensions for this range-finder are:

$B_0 = 0.555$ m, $d_l = 0.0543$ m, $d_c = 0.055$ m, $\beta_{0c} = 0.1554$ radians, and $f = 644.86$ (in pixels where each pixel is $11$ $\mu$m wide).

With a sustained speed of 1300 pixel/s and peak speed of 2500 pixel/s, the time for data collection is about 50 s for 256 by 256 resolution. Both range and
intensity pictures are generated and the angular resolution is programmable up to 750 horizontal by 3000 vertical. The range data is generated in "stripe" format, an image in which the value for each pixel is the angle measured by the CCD camera. A PC compatible with a DSP board is dedicated for the control of the range finder.

3.1.1 Accuracy

As in most system the accuracy of the system can be divided into systematic and random accuracy. In this particular system, high accuracy is needed in the differential measurements used for jump and curvature edge detection. The detection of these edges uses the first and second derivatives of the range coordinates.

The main source of systematic error is from the horizontal deflection motor. This is a micro-stepped stepping motor. The motor has 200 real steps per revolution and a micro-stepping driver generating 25000 steps per revolution. This gives about 125 micro-steps per real step. The problem with the motor/driver is that the angular position of the motor shaft is not linearly dependent to the micro step position. A special procedure described in the appendix is used to compensate for this problem. A plot of the compensation is shown in figure 6. The distortion corresponds to approximately 12 micro-steps peak to peak.

The random errors are introduced by:

- random errors in the horizontal deflection angle \( \beta_l \).
- random errors in the vertical deflection angle \( \alpha_l \).
- random errors in the estimated camera angle \( \beta_c \).

\[
\sigma_{x_r}^2 = \left( \frac{\delta x_r}{\delta \beta_c} \right)^2 \sigma_{\beta_c}^2 + \left( \frac{\delta x_r}{\delta \beta_l} \right)^2 \sigma_{\beta_l}^2
\]

\[
\sigma_{z_r}^2 = \left( \frac{\delta z_r}{\delta \beta_c} \right)^2 \sigma_{\beta_c}^2 + \left( \frac{\delta z_r}{\delta \beta_l} \right)^2 \sigma_{\beta_l}^2
\]  

(5)
Figure 6: Horizontal Motor Compensation for Several Scenes
\[ \frac{\delta x_r}{\delta \beta_c} = \frac{B_0 + (d_l - d_c) \tan(\beta_l)}{(\cos(\beta_c)(\tan(\beta_c) - \tan(\beta_l)))^2} \]

\[ \frac{\delta x_r}{\delta \beta_l} = \frac{B_0 + (d_l - d_c) \tan(\beta_c)}{(\cos(\beta_l)(\tan(\beta_c) - \tan(\beta_l)))^2} \quad (6) \]

\[ \frac{\delta z_r}{\delta \beta_c} = \frac{\delta x_r}{\delta \beta_c} \tan(\beta_l) \]

\[ \frac{\delta z_r}{\delta \beta_l} = \frac{\delta x_r}{\delta \beta_l} (1 + \tan(\beta_l)) \quad (7) \]

As in this range-finder the \( d_l - d_c \) distance is much shorter than the baseline \( B_0 \) the expressions can be simplified as:

\[ \frac{\delta x_r}{\delta \beta_c} \approx -\frac{x_r^2}{B_0 \cos^2(\beta_c)} \]

\[ \frac{\delta x_r}{\delta \beta_l} \approx \frac{x_r^2}{B_0 \cos^2(\beta_l)} \quad (8) \]

### 3.1.2 Characterization

In the previous section the accuracy for the range-finder was derived in terms of \( \delta \beta_{rc} \) and \( \delta \beta_{rl} \). The camera accuracy \( \delta \beta_{rc} \) is not easy to accurately model. Instead experiments were performed to study this inaccuracy term when the distance and light return levels were changed.

This was performed using the range-finder and scanning a small flat white surface at different distances. Instead of having surfaces with different reflectivity the light return levels were altered with neutral density filters in front of the camera and by scanning at different speeds. Note that the light return level could also have been changed with the aperture of the lens, but that would change the focus and therefore influence the centroid calculation method that is used to find the peak on the CCD array.
The scanned data was fitted to a plane and the average distance to the plane for each column was found. As the scanner was scanning vertically the angle $\beta_{rl}$ was constant for each column and the influence from $\delta \beta_{rl}$ was calibrated away for each column. Then a new plane was fitted to the calibrated data and the standard deviation from that plane was calculated. This standard deviation is due to the inaccuracies in $\beta_{rc}$ and $\alpha_l$, the later term is highly correlated in the vertical direction and after the median filtering its influence on the standard deviation is negligible.

In figure 7 the effect on the standard deviation of the x coordinate, $\sigma_x$, on distance is shown, it can be seen that it fits well with the quadratic function theoretically derived (the shown quadratic functions have the constant multipliers experimentally set, to fit with the data). In figure 8 the effect of varying intensity return levels in shown. The steep increase in standard deviation at low intensities occurs as the signal level becomes comparable to the noise level.

When deriving $\sigma_{ue}$ from the median filtered images it gives values between 1.8 $\mu$m and 6.1 $\mu$m at the different intensity values, corresponding to 0.16–0.55 pixel widths.

To verify that this was an accurate way of characterizing the range-finder the tests were also done with a few non-white surfaces and surfaces placed at varying orientation. It was shown that this data agreed well with the other data and therefore that the procedure was sound.

3.2 Range-Finder Simulator

The input data to the simulator is the set of objects that are assumed to be present in the scene and their positions and orientations, as well as the height, width, and starting position of the scan. The simulator generates images in the same format as
Figure 7: The dependency of range-finder’s $\sigma_x$ on $x$

Figure 8: The dependency of range-finder’s $\sigma_x$ on intensity return level
the range-finder. These images have as coordinate axes the horizontal and vertical
deflections of the scanner, and have the camera pixel coordinate $u_c$ and the intensity
return level $I$ stored at each pixel. The simulator also stores the object and the
surface which are hit by the laser, which is useful for algorithm development and
scene prediction. These are saved as surface and object ID-numbers using 16-bit
integers.

The objects in the scene are described by models using a boundary representa-
tion scheme. The models are hierarchically built up from points, edges, loops, and
surfaces. The edges can be straight edges, ellipse segments, or can be implicitly
defined by the neighboring surfaces, but currently only straight edges and circu-
lar arcs have been implemented. The loops are composed of a list of edges and
the direction along the edges\(^2\). The loops represent the external and any internal
boundaries of the surfaces. The surfaces that are currently implemented are planar
(with any polygonal external and internal boundaries), cylindrical, and spherical
(ubbounded).

The simulator was originally designed and implemented by Stiefel [28] and
has been enhanced to better suit the current object recognition system.

### 3.2.1 Simulator Working Principles

The simulator accurately mimics the unusual perspective distortion in the image
finder images. This is achieved by using a ray-tracing method [29]. It calculates
the origin and direction of the laser beam at each image pixel and then intersects
this ray with the objects in the scene. It finds the object, if any, that is struck by
the laser and calculates the 3D coordinates for the intersection point.

The laser ray, $\vec{r}_i$, starts at a moving scanner focal point that rotates a distance

\(^2\)Each edge is part of two loops, one in the forward direction and one in the backward direction.
\( d_t \) behind the vertical deflection axis. It then hits the vertical deflection axis at a point that moves with the horizontal deflection. The ray is defined to originate at the moving focal point of the scanner and its direction is the vector between the two points, see equation 9.

If the laser ray hits an object at the 3D point \( \vec{r} \), this point and the camera’s front focal point form the camera ray \( \vec{r}_c \). (See equation 10.) This ray is again intersected with the different objects and, if the first object that is hit and the intersection point for this ray are not the same as that of the laser ray, that point must be in the shadow of the object hit by the camera ray and is therefore not detectable.

\[
\vec{r}_l = d_l \begin{pmatrix} -\cos(\alpha_l) \\ -\sin(\alpha_l) \\ 0 \end{pmatrix} + tN_i \begin{pmatrix} \cos(\alpha_l) \\ \sin(\alpha_l) \\ -\tan(\beta_l) \end{pmatrix} \tag{9}
\]

\[
\vec{r}_c = \begin{pmatrix} -d_c \cos(\alpha_l) \\ -d_c \sin(\alpha_l) \\ B_0 + d_c \tan(\beta_{0c}) \end{pmatrix} + tN_c \begin{pmatrix} p_x + d_c \cos(\alpha_l) \\ p_y + d_c \sin(\alpha_l) \\ p_z - B_0 - d_c \tan(\beta_{0c}) \end{pmatrix} \tag{10}
\]

\( \alpha_l \) and \( \beta_l \) are affine functions of the pixel coordinates \( u \) and \( v \) respectively. \( N_i \) and \( N_c \) are used to normalize the direction normals. The parameter \( t \) is the distance along the ray.

For the camera the limited field of view also has to be taken into account. This is done by transforming the 3D point into a coordinate system with \( y_c \)-axis along the focal axis and \( x_c \)-axis along the focal plane of the camera, see figure 5. The equation:

\[
u_c = f \frac{x_c}{y_c}
\]

then gives the camera pixel coordinate and if it is between the limits of the CCD of the camera it is detectable, otherwise the point is non-detectable due to limited field of view.
The intensity return level is calculated for detectable points. This calculation is based on the angles of the two rays, the normal of the object surface at the intersection point, and the reflectance model for that surface. The surface model is lumped together with the laser power and camera properties. The only surface model used is due to Phong [29] and given here as equation 11.

\[
I = \frac{G_c \Delta t A P (k_d (\hat{n} \cdot \hat{r}_c) + k_s (\hat{n} \cdot \hat{r}_{r,l})^n)}{r^2}
\]

where \(G_c\) is the lumped gain of the A/D converter and camera, \(\Delta t\) is the integration time of the camera, \(A\) is the area of the lens aperture, \(\hat{n}\) is the normal of the object at the point, \(\hat{r}_c\) is the normal of the camera ray, \(\hat{r}_{r,l}\) is the laser ray normal after reflection at the point, \(P\) is the laser power, and \(r\) is the distance from the camera focal point to the point. The surface parameters are the diffuse and specular reflection coefficients \(k_d\) and \(k_s\), and the surface specularity, \(n\).

For non-detectable points as well as points where the laser ray does not hit any object, the intensity return of the pixel, \(I\), and the camera coordinate, \(u_c\), are set to zero.

It is also possible to simulate the image that the range-finder can generate in its intensity only mode\(^3\). In this mode the range-finder can quickly (\(\approx 1\ s\)) generate an intensity image of the scene. This is done using ambient light and a horizontal resolution equal to that of the camera (in this case 256 pixel). The vertical resolution is controlled by the vertical scanning mirror. The advantage of this mode over using another CCD camera is that the camera is already available and that the location of the camera in the range-finder coordinate system is well-known. This mode of the range-finder is useful in the scene prediction described in the next chapter.

\(^3\)This mode is not yet implemented
The limitations of the current simulator are:
- atmospheric effects are ignored,
- only opaque surfaces are handled,
- secondary rays as produced by very glossy mirror-like surfaces are not taken into account, and
- A pin-hole model is used for the camera, which ignores the effects from defocusing and lens aberrations.

These limitations play a negligible role under the conditions in which the range-finder and mobile robot are expected to work.

### 3.2.2 Simulator Support in Model Library

The models currently used are described by boundary representations (B-rep) and the surface types used are planar surfaces, cylinders, and spheres. The planar surfaces are delimited by arbitrary polygons as external and internal (holes) boundaries. The cylinders are delimited by planar and circular edges. Other surfaces such as quadrics are easy to add as the system is built in an object oriented fashion using C++. Each new surface type is derived from the general surface class and need to specify methods for intersection and normal calculation.

The objects are stored with lists of the surfaces, the external and internal loops, the edges that make up the loops, and the vertices and other points that define the location of points and specify the surfaces. The edges are used in the object recognition system to find the neighbors of surfaces (this is usually not calculated for standard CAD systems.)

### 3.3 Logical Sensors

The segmentation scheme uses the concept of real and logical sensors, the real sensors in this case are range and intensity that are generated by the range-finder.
The logical sensors [30] are abstract models for either real sensors or for "sensors" derived from the real sensors. In this system these derived sensors are edge detection methods that generate edge images from the range and intensity images. The different edge images are jump, concave fold, convex fold, and reflectivity edges. A final logical sensor combines these edge images into one image.

The concept of logical sensors is used to emphasize the importance of accurate characterization of not only the real sensors, but also of the processing applied to these. This is often lacking in computer vision literature, but is a very important step.

The work is related to that of Fan, Medioni, and Nevatia [31] who also combine different edge images, they are, however, not using intensity data and do not derive any sensor characteristics.

3.3.1 Development of Logical Sensors

The edge detection is done in two phases:

- Independent detection of the jump, concave, convex, and reflectivity edges.
- Fusion of the edges

Using the previously derived inaccuracy equations for the range-finder and the effects of the edge detection methods on the standard deviation in each measurement (jump gradient, curvature magnitude, intensity gradient) is estimated at each edge point. For each of the different edges an evidence image is generated where the measurements are divided by their standard deviation, as estimated on each individual edge point. This is a rough approximation of "evidence", as it assumes that the mean values of the different distributions are zero and they are clearly not as the magnitude of the gradient is strictly positive and the minimum and
maximum curvatures are also biased from zero. A better approximation would be to estimate the mean values of the noise also.

The edge detection processes all start with median filtering, Gaussian smoothing, and derivative estimation by linear filtering. The Gaussian smoothing and derivative estimation are convolutions of the image with finite filter masks. If the noise in the image is considered to be white, the effects of these convolutions on the standard deviations are easy to calculate. This is not true for the median filtering however and the effect of this was experimentally evaluated.

The edge images use “crack edges”. These edges are in between the pixels, and therefore called “crack edges”. This kind of edge eliminates the problems associated with having the edges occupying space of the image, e.g. the sum of segment sizes is equal to the image size.

Approximate Gaussian smoothing using a simple separable filter [32] is used; this method is very efficient as it only requires additions. The convolution mask can be described by a two-dimensional Z-transform, with variable $p$ in the horizontal direction and $q$ in the vertical direction, the equation for the mask is:

$$G_n(p,q) = 4^{-(n-1)}(1 + p)^{n-1}(1 + q)^{n-1}$$ (12)

Where $n$ is the size of the filter. It can be shown that this filter approximates a Gaussian filter with variance $(n - 1)/2$.

The derivatives are then calculated using the Sobel filters.

$$\frac{\delta}{\delta u} = \frac{1}{8}(1 + p)(1 + q)^2(1 - p)$$

$$\frac{\delta}{\delta v} = \frac{1}{8}(1 + p)^2(1 + q)(1 - q)$$

$$\frac{\delta^2}{\delta u^2} = \frac{1}{4}(1 + q)^2(1 - p)^2$$
\[
\frac{\delta^2}{\delta u \delta v} = \frac{1}{4} (1 + p)^2 (1 - q)^2 \\
\frac{\delta^2}{\delta v^2} = \frac{1}{4} (1 + p) (1 + q) (1 - p) (1 - q)
\] (13)

The effect on the variance of these filters can be found by the equation

\[
\sigma_{out}^2 = \sum_{i,j} c_{ij}^2 \sigma_{in}^2
\] (14)

This assumes that the noise is un-correlated and has uniform variance within the filter mask. This equation is currently evaluated using a small image with one impulse, on which the different linear filters are applied, the resulting image contains the filter coefficients and these are then summed up. This is done for all the different filters and at the different sizes of the smoothing filter used for the different detection processes.

In the jump and intensity edge detection the edges are found as the zero-crossings in the Laplacian, and the edge strength at the zero-crossings is estimated with the magnitude of the gradient. This is a non-linear function, but it is differentiable everywhere, except at the origin, and it can be shown not to influence the standard deviation. In the jump edge detection the input image is the x-coordinate image of the range image. The intensity and jump edge processing is shown in figures 9 and 10.

For the convex and concave fold edges the edges are detected as the points that form minima or maxima respectively in any direction (one dimensional extrema) of the minimum curvature \(k_{min}\) and maximum curvature \(k_{max}\) respectively. The equations for calculating the minimum and maximum curvature are given below. Note that they calculate the true curvatures and are therefore also useful for classifying the segments obtained, if the goal of the system was only to find the fold edges a simpler approximation could be used. \(\vec{x}\) is the 3D physical coordinate
Figure 9: Intensity Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below.
Figure 10: Jump Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below.
of one pixel, $\vec{x}_u$ is the $u$ (horizontal) derivative of $\vec{x}$, $\vec{n}$ is the normal at the pixel. $H$ and $K$ are the mean and Gaussian curvatures.

$$\vec{n} = \frac{\vec{x}_u \times \vec{x}_v}{\|\vec{x}_u \times \vec{x}_v\|}$$

$$\beta = G^{-1} B$$

Where the elements of the $G$ and $B$ matrices are:

$$g_{11} = \vec{x}_u \cdot \vec{x}_u \quad g_{12} = g_{21} = \vec{x}_u \cdot \vec{x}_v \quad g_{22} = \vec{x}_v \cdot \vec{x}_v$$

$$b_{11} = \vec{x}_{uu} \cdot \vec{n} \quad b_{12} = b_{21} = \vec{x}_{uv} \cdot \vec{n} \quad b_{22} = \vec{x}_{vv} \cdot \vec{n}$$

$$K = \det(\beta)$$

$$H = \frac{1}{2} \text{trace}(\beta)$$

The minimum and maximum curvatures are the eigenvalues of the $\beta$ matrix. As the matrix is only $2 \times 2$ the characteristic equation is only second dimensional and the eigenvalues have algebraic solutions.

$$k_{\text{min}} = H - \sqrt{H^2 - K}$$

$$k_{\text{max}} = H + \sqrt{H^2 - K}$$

The directions of the eigenvectors are also calculated.

The strength of the fold edges is estimated as the absolute value of the minimum and maximum curvature respectively at the edge points. Again the minimum and maximum curvature estimation is non-linear, so its influence on the standard deviation is hard to calculate. By using the same method as was used in section 3.1.1 the standard deviations for $k_{\text{min}}$ and $k_{\text{max}}$ could also be computed, but the equations would be much more complicated. Instead they were experimentally measured and a simple model was derived for intensity and range dependency within the 1-2 m working range.
The range finder's "stripe" image is converted to a true 3D range image where the x, y, and z coordinate values are stored for each pixel. The derivatives can then be calculated on this image generating 15 derivative values ($\frac{\delta}{\delta u}$, $\frac{\delta}{\delta v}$, $\frac{\delta^2}{\delta u^2}$, $\frac{\delta^2}{\delta v^2}$, and $\frac{\delta^2}{\delta u\delta v}$ for each of the three coordinates). This makes the program use a very large data space and also makes the program slower due to all the memory accesses. An alternative to this is to derive the needed derivatives directly from the "stripe" image derivatives (see the appendix). The curvature edge extraction is depicted in figure 11.

These evidence images are then thresholded using a hysteresis thresholding technique. The hysteresis helps in avoiding short breaks in the detected edges. This technique uses two thresholds (normally set to 3 and 1). The high threshold is first applied to the image and the lower is used to follow edges out from edges detected by the high threshold. The edge-following continues until the edge evidence falls below the low threshold.

### 3.4 Edge Fusion

The fusion of the different edges is based on models of the types of edges and on the interdependency of the different detection schemes. The types of edge are jump, concave fold, convex fold, and reflectivity. The intensity image is also thresholded to classify pixels as shadow/valid-data. The models of the interaction between edges are expressed in the following rules:

- any edges inside shadow regions are spurious,
- reflectivity edges close to jump edges are due to the discontinuity in range,
- reflectivity edges close to fold edges are due to the discontinuity in surface normal,
- fold edges close to jump edges are due to the detection scheme.
Figure 11: Curvature Edge Processing. The steps are indicated on the top line of each box and the class name is in parentheses below.
• fold edges close to shadow regions are due to the detection scheme,
• jump edges close to shadow regions are due to the detection scheme, and
• reflectivity edges close to shadow regions are due to the detection scheme.

In the above rules the degree of closeness is different for the different rules, and is based on the size of the filter masks used in their detection.

When evaluating these rules the distance from the current edge point to the closest edges in the other edge images are evaluated using the Euclidean distance transform [33]. The process is to compute the five distance transforms for the shadow, jump, concave fold, convex fold, and reflectivity images. The distance transform stores the squared distance, and the vector to the closest edge point in each pixel. When evaluating the rules for each edge pixel the distance to the closest point in the other images can be directly found as well as the position of that point. The position is used in the fusion of fold and reflectivity edges where the reflectivity edge is more accurate in estimating the location of the fold edge. The dependencies between the thresholded evidence images and the fused image is shown in figure 12.
Figure 12: Fused Edge Processing using Euclidean Distance Transforms. The steps are indicated on the top line of each box and the class name is in parentheses below.
Chapter 4

FEATURES

The object recognition algorithm uses aggregates (edges, loops, segments, and object regions) for its reasoning. These are extracted from the input scene using various segmentation routines described here. After the different aggregates have been extracted various descriptors/features are calculated for them. Some of these features are only calculated if/when they are required by the recognition algorithm.

In this chapter a new method of tracking the edge image to create an edge map of connected edges is described. Also methods to go from this edge map to loop maps and segment maps are described. A new hierarchical description of the segment map is used.

4.1 Segmentation

In order to have some aggregate features for object reasoning the pixel level images have to be converted. This is done in steps where the pixel level edge image is converted to an edge map which describes complete edges and the connections between edges. A loop map is then generated by linking edges that form loops. The relations between loops are included in the loop map. Finally a segment map
is generated that describes segments, segment groups, and their relations.

As the edge image contains gaps and missing edges an edge closing routine is used to attempt to find and fill these gaps with new edges. The overall process of segmentation and calculation of segment features is shown in figure 13. The following sections will explain the process.

4.1.1 Edge Map

The fused edge image is converted to an edge map where the edges are linked into lists of edge coordinates\(^1\). The edges have vertex points and/or end-points at the start and end of the lists.

This tracking is done by first creating the sets \(E\) and \(V\) of edge fragments. The edge point set, \(E\), is created from the point coordinates that have have two edges connected. The vertex point set, \(V\), is created from the coordinates that have one, three, or four edges originating from them. Both the \(E\) and \(V\) sets are described by triplets of the \(X\) and \(Y\) coordinates of the point and an integer describing in what directions edges are attached. An example image and the edge point \((E)\) and vertex point \((V)\) sets are shown in figure 14.

The tracking consists of taking a starting point from the set \(V\), and following an edge out from it. This direction is then removed from the starting point, and if there are no more edges attached to the starting point it is removed from \(V\). The point and the selected direction leads to a new point. If this point is in the \(V\) set the end of the edge has been reached. Otherwise the point is found in the \(E\) set. Based on what direction we came to the point we determine in what direction to take from the point and we go to this new point and delete the old point from the \(E\) set. This is repeated until a point in \(V\) is found. When an end point in \(V\) has

\(^1\)The edge coordinate system is not the same as the pixel coordinate system, see figure 14.
Figure 13: Segmentation. The steps are indicated on the top line of each box and the class name is in parentheses below.
Figure 14: Example of Edge Image and edge and vertex point sets used for Edge Map creation
been found the direction that we came to that point is deleted from the end point and if there are no more edges attached it is deleted from $V$. At this time a new starting point is taken from $V$ and the process is repeated until there are no more points in $V$.

This can be illustrated by tracing one edge from figure 14. Assume that the edge map creation starts with the point (5, 1, RIGHT) from $V$, there is only one direction to go from this point and after deleting this direction no more edges are attached and the point is deleted from $V$. After taking the RIGHT direction from (5, 1) we are at (6, 1). This point can not be found in the $V$ set so the edge point (6, 1, LEFT | DOWN) is found in the $E$ set. As we came to this point from the RIGHT we take the DOWN direction out from the point and delete the (6, 1, LEFT | DOWN) point from $E$. The tracking ends at the point (6, 5, UP | LEFT | DOWN) from the $V$ set. The UP direction is deleted from this point, but the point remains in $V$ as there are two more edges attached. The same procedure is repeated two more times to track the other two edges connected to this point.

To handle closed edges (Like the one in the top-left corner of the image is figure 14), without any vertex or end points, a final stage is done if there are edge fragments left in $E$ and no vertex/end points left in $V$. These remaining edges are tracked by picking the first edge fragment in $E$ and tracking this until this position is reached again.

This method of edge tracking is quite efficient due to use of hashed implementations of the sets. There is only one scan of the complete edge image to generate the $E$ and $V$ sets. The rest of the algorithm works only on these much smaller data sets.
Figure 15: Example Edge Map
4.1.2 Loop Map

The edge map is traced on the edge level to generate the loop map. First a set of edges with directions is created from the edge map, this is a set of tuples (Edge, direction). Initially the set will contain all edges in forward and backwards directions.

The tracing is done by taking one edge and direction from the set. The edge is then followed in that direction (the end-points can be directly accessed, so the intermediate points are not accessed) and the left-most edge at this point is then selected in the direction going out from the point. If there is no edge at this point the routine backtracks to the previous edge’s end-point. The tracking ends when reaching the edge that the loop started with.

This can be illustrated by the creating of the loop map in figure 16 from the edge map in figure 15. Suppose that the tracking is started by picking edge 1 in the backward direction. When going in the backward direction we reach vertex 4 and the left-most edge out from this point is edge 5 in the backward direction. Then edge 6 in forward direction is followed, etc. For the edges 6 and 8 the tracking will backtrack as these edges are open. Finally loop 1 will be: (e1, -) -> (e5, -) -> (e7, +) -> (e9, -) -> (e3, +).

The result is that all edges are either part of zero or two loops. For the edges that belong to no loops the loop the edge is inside is found and the tuple of the edge’s pointer and the loop’s pointer are stored in the loop map’s internal edge set. For the edges that are part of two loops the two loops are found and triples of the edge’s pointer and the two loop’s pointers are stored in the loop-edge set. the internal edge and loop-edge sets are used in the edge-closing to find edges that inside an particular loop, and also to find what loops are broken if an edge is removed.
The areas of the loops are calculated using:

\[ area = \frac{1}{2} \sum_{k=1}^{n} u_k u_{k-1} - u_{k-1} u_k \]  

(20)

where \( u_k \) and \( v_k \) are the coordinates of the \( k^{th} \) edge-point in the loop. The sum is split up into the contributions from each edge in the loop, and the area for the loop is then only a sum of the individual edges partial "area" with sign corresponding to edge-direction.

The generated loops are also stored in two lists depending on the areas of the loops, positive area loops are stored in the external-loop list and negative area loops in the internal-loop list. The relations between the loops are then calculated to build up an hierarchy of loops, the internal loops and internal edges are marked as internal to the next larger external loop enclosing. The base of the hierarchy is the bounding loop that encloses the whole image.

### 4.1.3 Segment Map

The segment maps contain information about each segment in the scene. This information is hierarchically ordered in segments and segment-groups. A segment is enclosed by a counter-clockwise loop and a segment-group is enclosed by a clockwise loop. With this description it is easy to find neighboring segments and what segment this segment is inside, e.g. segment corresponding to the area inside a hole of another segment. It is also easy to compute the area of each segment, see below.

From the loop map the segment map is generated simply by finding external and internal loops, and creating segments and segment-groups using these. For each counter-clockwise loop a segment is created and for clockwise loop a segment-group is created. Note that counter-clockwise loops have positive area and clockwise loops have negative area.
The segments are then grouped into segment-groups enclosed by internal loops and a hierarchy of segments and segment-groups is built. Each segment has a pointer to its parent segment-group and a set of pointers to child segment-groups (holes in the segment). And in the same way a segment-group has a pointer to the parent segment and a set of child segments.

After the segment map is created the area of each segment is calculated. This is done simply by summing of the areas of the external loop of the segment and the external loops of the child segment-groups.

A comparison of the loop map in figure 16 and the corresponding segment and segment-group hierarchy in figure 17 helps in understanding this. The first step is to create segments for all loops with positive areas (counter-clockwise loops), and segment-groups for all loops with negative areas (clockwise loops). For the example this will be three segments, 1, 2, and 3, with the loops 1, 2, and 3 as their external loops. Segment-group 1 is created with external loop 4. Next the segment-group that is the parent of each segment is found, in this case all three segments are children of the segment-group 1.

4.1.4 Edge Closing

The resulting segmentation is typically incomplete due to missing edge segments; the process of closing these gaps is also based on the loop concept. For each segment over a certain size an edge path that divides the segment into two segments is sought. This can be done by adding edges to split the external loop of the segment, or by generating a new internal loop. Currently the strategy only implements splitting the external loop, and this is done using a best first search where the cost is the total length of the new edge segments needed for the new edge path.
The formulation of the edge closing in terms of independent loop splitting operations makes the method applicable for parallel processing.

### 4.1.5 Object Regions

To reduce the complexity of the object recognition routines an attempt is made to find groups of segments that are likely to be from the same object. This is done by evaluating the types of the edges between segments. As stated before the types of edges can be jump, concave, convex, intensity, shadow. The only type that is almost certain not to bound different objects is the concave edge type. By grouping all the segments that are connected by concave edges of sufficient length into object regions we have larger regions to reason about in the object recognition routines. Furthermore the search can be divided into independent subtasks.
4.1.6 Experiment

The test scene presented here\(^2\) consists of a 100 mm diameter cylinder lying on top of a square base block. The block is oriented with one edge facing toward the range-finder and another edge supporting the cylinder, forming a concave fold. To the left in the image is a jump edge to a more distant vertical plane. In this scene there are no shadow regions and there are no detectable intensity edges.

The results on the scene are shown in figure 18 and 19. The execution times are given in table 1. As can be seen from the example, spurious fold edges are removed close to the jump edge, but otherwise it is hard to illustrate the efficiency of the evidence calculations and thresholding.

4.2 Feature Calculation

The features for the edges and segments are used in the evaluation of constraints in the object recognition. The features calculated are described in chapter 5.

\(^2\)This is the scene called "clyedge".
Figure 18: Segmentation example. (a) intensity image (b) jump edges (c) concave edges (d) convex edges
Figure 19: Segmentation example. (a) fused edges (b) segments
Table 1: Execution times using an i486DX-33 processor with 16MB RAM and 256kB cache running 386BSD.

<table>
<thead>
<tr>
<th>Process</th>
<th>Sub-process</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Region Extraction</td>
<td>Total</td>
<td>1.6</td>
</tr>
<tr>
<td>Intensity Edges Detection</td>
<td>Median</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Derivatives</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>Detection</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>Evidence Thresholding</td>
<td>2.3</td>
</tr>
<tr>
<td>Range Image</td>
<td>Stripe Median</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>Conversion</td>
<td>1.6</td>
</tr>
<tr>
<td>Jump Edge Detection</td>
<td>Derivatives</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Detection</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Evidence Thresholding</td>
<td>2.0</td>
</tr>
<tr>
<td>Fold Edge Detection</td>
<td>Derivative Calculation</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td>Normal Calculation</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>Curvature Calculation</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>Evidence Thresholding</td>
<td>7.1</td>
</tr>
<tr>
<td>Edge Fusion</td>
<td>Distance Transforms (5)</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>Fusion</td>
<td>1.4</td>
</tr>
<tr>
<td>Edge Closing</td>
<td>Edge Map</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Loop Map</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Segm Map</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Loop Splitting</td>
<td>2.3</td>
</tr>
</tbody>
</table>
Chapter 5

RECOGNITION

The recognition is divided into three different phases. First, the indexing phase selects possible object models for each object region. The different object region/model pairs are then evaluated in the second step, hypothesis generation, which finds pairs between individual features in the object region, such as segments, edges, points and corresponding features in the object model. The pairs are evaluated using unary and binary constraints. If a solution fulfilling all these constraints can be found in the hypothesis generation step, this solution is then evaluated using global constraints in the third, hypothesis-testing phase. In this phase, a location and orientation of the object is calculated. Again, if the solution fulfills all constraints, the last test is to use the simulator to generate a simulated view of the object and comparing this with the actual scene.

The approach is similar to those of Grimson [10] and Faugeras [34]. The contributions are mainly in using new constraints for the indexing and recognition and the use of the simulator to verify the hypotheses.
5.1 Indexing

The indexing step as implemented is quite simple. It involves finding segment/surface matches for all segments in an object region to surfaces in one object model. Only the unary constraints are evaluated and these usually are not very discriminating.

For each segment a set of object hypotheses are built. Each object hypothesis assumes that the object region matches one object model. It contains for each segment in the object region a set of matching surfaces in the object model. If there are one or more possible surfaces for every segment in the object group the indexing step this object hypothesis is passed to the next step of recognition. Otherwise the indexing has failed for the model.

The unary constraints are:

- Number of neighbors. A segment in the scene must have not more neighbors, \( N_s \), than does the surface in the model, \( N_m \). In the scene the neighbors are counted as the number of connected segments, and two surfaces are not connected if they are separated by concave fold edges, or jump edges.

\[
N_s \leq N_m
\]  

(21)

- Surface match. The surface match depends on the type of model surface.

The surface match constraints for planar surfaces are:

- Planar fit. The distance between the fitted plane and 95 % of the pixels inside the segment, \( \varepsilon_{95} \), have to be less than threshold, \( \Delta_{95} \). This allows a few pixels to have a large deviation to account for segmentation inaccuracies.

\[
\varepsilon_{95} < \Delta_{95}
\]  

(22)
• Planar area fit. The area of the segment projected on the fitted plane, $A_s$, must not be greater than the area of the surface, $A_m$, plus a tolerance, $\Delta_A$, for occluded segments. The area of the segment has to be within symmetric tolerances for non-occluded segments.

\[
\begin{align*}
A_s - A_m &\leq \Delta_A & \text{for occluded segments.} \\
|A_s - A_m| &\leq \Delta_A & \text{for non-occluded segments.}
\end{align*}
\]  

(23)

The tolerance is based on segment area and segment orientation, it is larger for large segments and for segments that are viewed at an oblique angle. The normal of the plane is $N_s$, and the centroid is $C_s$. The dot product between the normal and the ray is then approximately $N_s \cdot C_s$, and $\Delta_A$ is given by the following equation, where $\Delta_{A0}$ is a constant.

\[
\Delta_A = \frac{\Delta_{A0}}{|N_s \cdot C_s|}
\]

(24)

• Planar convex hull area fit. The area of the convex hull of the segment, $A_{c,s}$, must be smaller than the area of the convex hull of the surface, $A_{c,m}$, plus a tolerance, $\Delta_A$, for occluded segments. The area of the convex hull of the segment has to be within symmetric tolerances for non-occluded segments. The tolerances are based on segment area and segment orientation as described above.

\[
\begin{align*}
A_{c,s} - A_{c,m} &\leq \Delta_A & \text{for occluded segments.} \\
|A_{c,s} - A_{c,m}| &\leq \Delta_A & \text{for non-occluded segments.}
\end{align*}
\]  

(25)

• Maximum Diameter. The maximum diameter, is defined as the maximum distance between two points on the external loop of the segment. The segment’s maximum diameter, $D_s$, must be smaller than the maximum
diameter of the surface, $D_m$, plus a tolerance, $\Delta_D$, for occluded segments. The maximum diameter of the segment has to be within symmetric tolerances for non-occluded segments

$$\begin{cases} D_s - A_m \leq \Delta_D & \text{for occluded segments.} \\ |D_s - D_m| \leq \Delta_D & \text{for non-occluded segments.} \end{cases}$$  \hspace{1cm} (26)$$

For cylindrical and spherical surfaces the modified Levenberg-Marquardt algorithm (see appendix) is used to find the best orientation/translation to fit the given segment. A similar method has been used by Flynn [35].

For cylindrical surfaces the equation for the cylinder is described by:

$$F_c(x, y, z; r, \nu, \varphi, x_0, y_0) =$$

$$z \cos \varphi + \sin \varphi (y \sin \nu + z \cos \nu) - x_0^2 + (y \cos \nu - z \sin \nu - y_0)^2 - r^2$$  \hspace{1cm} (27)$$

For spherical surfaces the equation is:

$$F_s(x, y, z; r, x_0, y_0, z_0) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2$$  \hspace{1cm} (28)$$

The modified Levenberg-Marquardt algorithm uses the above functions and their Jacobians to find a solution that minimizes the least mean square system of the above non-linear functions. The fitting is done on up to 200 points inside the segment, creating the same amount of functions in the system.

The maximum distance between the fitted surface and any point in the segment has to be below a threshold for match.

This method is much more robust than fitting quadrics as the number of parameters is less. The fitting of quadrics using an eigenvector method was tested but it could not cope with the level of noise in the data. The modified Levenberg-Marquardt method had no problems fitting surfaces to the range-finder data.
Figure 20: Example Model and Object Region

Table 2: Example Object Hypothesis generated in Indexing step

<table>
<thead>
<tr>
<th>Segment</th>
<th>Surface Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{1, 2, 3, 4, 5, 6}</td>
</tr>
<tr>
<td>b</td>
<td>{1, 2, 3, 4, 5, 6}</td>
</tr>
<tr>
<td>c</td>
<td>{1, 2, 3, 4, 5, 6}</td>
</tr>
</tbody>
</table>

An example indexing between a cubical model object and an object region containing three surfaces will be described here. The model and object region are shown in figure 20.

In this case all six surfaces of the cubic model object have the same features and each of the three segments of the object region match all six model surfaces. The resulting object hypotheses passed to hypothesis generation will be as shown in table 2.
5.2 Hypothesis Generation

Each object hypothesis is evaluated using the binary constraints between segments/surfaces. The binary constraints are:

- The type of edge, if any, between segments and corresponding surfaces has to match. If there is no edge between the surfaces in the model there can be no convex, concave, or reflectivity edge between the segments in the scene. Otherwise there must be a scene edge of the same type as the edge between the surfaces in the model.

- The angles between segments and angles between corresponding planar surfaces have to be within some threshold.

- The distance between segments and distance between corresponding planar surfaces have to match. The scene distances are computed based on the fitted planar surfaces. The distances calculated are the perpendicular distances of the convex hull of one surface to the other surface.

- Visibility constraint. The angle between surfaces and segments have to be less than 2.6 radians\(^1\) as the range-finder cannot see two segments that have widely different normals.

A global constraint which is also tested during this step is that each surface can match only one segment.

The algorithm is a depth-first constraint satisfaction routine with back-tracking. The depth of the tree is the same as the number of segments in the object region. The maximum number of nodes in the tree, \(N_{\text{max}}\), will be dependent on how many

---

\(^1\)In general the angle has to be less than \(\pi\) radians, and for our particular range-finder geometry the value is 2.6 radians
Figure 21: Example of search in hypothesis generation step. Matching the a cube model and an object region from an octagonal object.

segments, $M$, there are in the object region and the number of surfaces that each segment matches, $K_j$. This leads to a maximum number of nodes,

$$N_{max} = 1 + \sum_{i=1}^{M} (\prod_{j=1}^{i} K_j).$$  \hspace{1cm} (29)

With discriminative constraints the number of nodes will be much smaller. Also for most applications the search can be stopped after one satisfactory solution has been found.

For the model and scene from figure 20 and the corresponding object hypothesis from table 2 the maximum number of nodes in the tree will be $1 + 6 + 36 + 216 = 259$ and the actual number of nodes in the tree is $1 + 6 + 36 = 43$. There will not be any solution and the matching will fail as several of the binary constraints do not match, e.g. the angles between surfaces (90 degrees) and between segments (45 degrees) do not match. The tree is shown in figure 21.
5.3 Hypothesis Testing

After successful hypothesis generation we have an object region with each segment paired with a matching model surface. However, this is only a partial solution to the problem since only local constraints have been evaluated. The next step finds the orientation and location of the object and uses this to evaluate the global constraints.

The orientation of each object is found using a technique by Faugeras [27]. This technique uses the directions of surfaces (normals of planar surfaces, the axis of cylindrical surfaces). The method minimizes the function:

$$\sum_i \| \hat{n}_i - R\hat{N}_i \|^2$$  \hspace{1cm} (30)

Where \( \hat{n}_i \) is the measured direction of a scene segment and \( \hat{N}_i \) is the direction of the corresponding model surface. The \( R \) Matrix is the rotation matrix.

The least square equation is solved using quaternions. These are defined as tuples \((s, w)\) where \( s \) is a scalar, and \( w \) is a three-dimensional vector. There is a new multiplication defined for quaternions:

$$ (s, w) \times (s', w') = (ss' - \langle w, w' \rangle, w \times w' + sw' + s'w) $$  \hspace{1cm} (31)

The conjugate quaternion for a quaternion is:

$$ \bar{q} = (s, -w) $$  \hspace{1cm} (32)

An rotation can be described by an rotation axis \( \hat{r} \) and an angle \( \theta \) to rotate around this axis. If a quaternion is formed as:

$$ s = \cos \frac{\theta}{2} $$  \hspace{1cm} (33)
\[
w = (\sin \frac{\theta}{2}) \hat{r}
\]  

(34)

The rotation of a vector \( u \) with a rotation matrix \( R \) is \( Ru \) or with quaternion description:

\[
q \ast (0, u) \ast \bar{q}
\]  

(35)

Using these rules for quaternions the rotation can be found by minimizing:

\[
\sum_i \| q \ast (0, \tilde{N}_i) \ast \bar{q} - (0, \tilde{n}_i) \|^2
\]  

(36)

It was observed by Faugeras that this could be rewritten to:

\[
\sum_i \| q \ast (0, \tilde{N}_i) - (0, \tilde{n}_i) \ast q \|^2
\]  

(37)

This turns out to be a quadratic form and is rewritten as:

\[
\sum_i q^t A_i q = q^t B q
\]  

(38)

Where \( A_i \) and \( B \) are symmetric four by four matrices. The \( A_i \) are found by applying the rules for quaternions. The \( A_i \)'s are then summed up into the \( B \) matrix.

The minimization is then done by finding the eigenvector with minimum eigenvalue for the \( B \) Matrix. This gives the quaternion which is then converted to a rotation matrix \( R \).

After the rotation for the object has been found a test of whether all the rotated scene directions fall close enough to the model directions. This eliminates most incorrect hypotheses that were found using only the unary and binary constraints.

The translation of each object is then found using a technique described by Grimson[10].
5.4 Verification

In the recognition system, features such as segments and edges are extracted from the range-finder images and a feature-based matching algorithm is used to hypothesize which objects are in the scene and their orientations. To verify that this hypothesis is correct we compare the simulated scene data with the observed data. This can be done by comparing the observed and predicted features or by directly comparing the observed and predicted images. This system uses the first approach during the hypothesis generation and the second approach after a hypothesis is made.

As part of this system, a range-finder simulator has been developed that can generate highly accurate range and intensity images identical in format to those generated by the range-finder. The accuracy of the simulator is such that the discrepancies between real and simulated data are almost completely due to inaccuracy in model orientation estimates and incorrect object hypotheses.

A similar idea was proposed by Bolles and Horaid [36] in which a range-prediction algorithm was also used, but it was based on a simpler technique that did not simulate an intensity image and cannot detect the no-data regions in the images. Furthermore, their system creates images using a “patch” method which results in an image with missing data making them impossible to process in ways which will be described in section 6.1. Moreover their simulated range data is used only to verify the hypothesis of one scene. In the system used in this work the range-finder simulator is also used to predict future scenes and plan new scene acquisitions (described in section 6.2).

The simulator differs from the VANTAGE sensor modeler of Ikeuchi [37] in that the latter creates simulations which are intended for algorithm development and lack the accuracy and detail necessary for verification and prediction.
Chapter 6

OBJECT TRACKING

This research integrates the recognition and tracking processes by forming and updating hypotheses of object models and poses. The forming of the hypotheses has already been described in chapter 5. In this chapter the updating of the hypotheses will be described.

6.1 Simulated Image Processing

The simulator generates images with the peak position, $u_e$, intensity, $I$, and the surface and object ID-numbers at each pixel. As these images are not directly suitable for use, a feature-based description is calculated.

The edges of the objects are extracted from the images by finding the “crack edges”. This is simply done using the information about which object and surface is seen at each pixel. For each pixel the one above and to the left is checked to see if the object and surface are the same. If they are, there is no edge, otherwise there is.

The obtained edge image is then converted to an edge map where the edges are linked into lists of edge coordinates. The edges have vertex points and/or
end points at the start and end of each list. The edges are also classified into the
different edge classes: concave and convex fold edges, jump edges, and reflectivity
edges.

This edge image is then traced in the same way as described in section 4.1.1
and the loop map and segment map are constructed from this edge map.

From the simulated range-finder image the features of the different segments
are then calculated. This is done in exactly the same way as for the true range-finder
images.

The result of this process is an almost perfect segmentation of the scene,
and the edges of the object surfaces are also found correctly (except for pixel
quantization errors). Using this segmentation the features for each surface and
edge are calculated. For surfaces, the features include: 2D area, 3D area, curvature
properties, average normal, centroid, and average intensity. The process and the
dependencies between the different objects are shown in figure 22.

6.2 Use of Simulator

The scene description derived in the previous section can be used for several
purposes:

- When developing the recognition system it is very valuable as it is then
  possible to separate the development of the segmentation process (of the
  real data) from the recognition process.

- After the recognition procedure, the results can be verified by comparing
  the real and the simulated data.

- When a scene description has been obtained from one scene this description
  can be used to predict the appearance of new scenes from different positions
  (this is useful for mobile robots.)

The simulator may also be used for testing the recognition process using well-
controlled input data. In addition, the simulated scene description can be used
Figure 22: Simulator Data Processing
to verify that the hypothesis derived from the real range-finder images match the data. This can be done by directly comparing the images as done in 3DPO [36]; comparing the predicted distance to an object with the sensed data. If the sensed data is close to the predicted data this adds positive evidence to the hypothesis; if the sensed data is further away than predicted this adds negative evidence; and if the sensed data is closer than predicted this is neutral evidence as this could be due to the presence of another object. This method’s disadvantage is that it is sensitive to errors in the estimates of the objects’ pose; small orientation and/or translation errors create large errors at the edges of the objects.

The method chosen extracts surfaces and edges from the predicted data and compares these to the real data. This is a comparison, that is based on the true data, that avoids the problems due to orientation estimation errors as the comparisons are done at the feature level. Note that the segmentation used to extract the features in this case is very easy and not as prone to errors as are the segmentation of the real range-finder data [38].

The feature-level comparison can either be done directly using the features that were extracted by the initial segmentation of the real range-finder scene, or by using the predictions to guide the extraction of the features directly from the data and then comparing the extracted features.

Finally, for a mobile robot application, given a scene hypothesis, the simulator can be used to predict the appearance of the scene from different locations and orientations of the mobile robot. This prediction can be used to selectively scan regions of the scene to update and verify:

- the object’s locations and orientations,
- the hypothesis about what objects are in the scene, and
- the robot’s location and orientation, given objects (landmarks) with known locations.
All of these capabilities are important for the robot's path planning. As the types of surfaces and other features are known, a model-driven approach to segmentation can be used, making the segmentation faster and more accurate.
Chapter 7

Experimental Results

In this chapter the results of applying the recognition algorithm to real scenes will be shown. First the models that were used will be shown in section 7.1. Various scenes containing these models are shown in section 7.2. Finally in section 7.3 the results of the recognition algorithm are presented.

7.1 Model Library

Each model in the library is described by an object description file. These files contain enough information to be used for both simulation and recognition of the object.

The models we will present here are:

**PaperRoll**  A simple cylinder with radius of 28 mm.

**Block**  A block with square base. Each side of the square is 40mm and the length of the block is 80mm.

**OctCup**  An octagonal cup.

**OctPlate**  An octagonal bowl.
Figure 23: The Model library (a) PaperRoll (b) Block

**Bone** A toy block somewhat resembling a bone.

**Arc** A toy block in the shape of an arc.

**SoapBox** An object with a cylindrical surface blending smoothly into two planar surfaces.

The different models are shown in figures 23 to 26. Note that they are not to scale.

### 7.2 Scenes

The experiments were done both with simulated scenes and real scenes using the range-finder data. The simulated scenes are generated from scene description files. These files describe what objects are in the scene and what orientations they are in.

The scenes obtained from the range-finder are:
Figure 24: The Model library (a) OctCup (b) OctPlate

Figure 25: The Model library (a) Bone (b) Arc
cylinder A scene that consists of the objects PaperRoll on top of the Block oriented so that there is a concave edge between an edge of the Block and the PaperRoll. A region to the left of the PaperRoll shows the background wall. The scene is shown in figure 27.

octCup The OctCup laying on a flat surface with a vertical wall behind the OctCup. The scene is shown in figure 28.

boneArc The Bone standing up to the left of the Arc on a horizontal surface. The Bone casts a shadow on a vertical wall behind the two objects. The Bone is only partially visible due to the limited field of view of the camera of the range finder. The scene is shown in figure 29.

octArc The Arc standing up at an angle relative to the range-finder's z-axis. The Arc is in front and to the left of the OctCup and casts a shadow on it. Both objects are on a horizontal surface and have a vertical wall behind them. The scene is shown in figure 30.
Figure 27: The cylinder scene (a) intensity image (b) range image

octArcBone The octArc scene with an extra Bone object placed behind the Arc and to the left of the OctCup. The scene is shown in figure 31.

plateCup The octCup placed on top of the OctPlate. The octPlate is placed upside down. The scene is shown in figure 32.

plateSoapArc The Arc placed in front of the octPlate. The SoapBox object placed on top of the OctPlate’s bottom surface. Most of Arc is not visible due to the limited field of view for the range-finder’s camera. The scene is shown in figure 33.

7.3 Recognition Results

The results of the algorithm applied to real scenes are found in section 7.3.1.
Figure 28: The octCup scene (a) intensity image (b) range image
Figure 29: The boneArc scene (a) intensity image (b) range image
Figure 30: The octArc scene (a) intensity image (b) range image
Figure 31: The octArcBone scene (a) intensity image (b) range image
Figure 32: The plateCup scene (a) intensity image (b) range image
Figure 33: The plateSoapArc scene (a) intensity image (b) range image
7.3.1 Real Scenes

The experiments for the real scenes were hampered by the deficiencies of the current range-finder. The distorted range data caused by the non-linearity of the horizontal deflection motor cause spurious edges at the peaks and valleys of the basically sinusoidal distortion, see figure 6. The calibration procedure for this eliminates most of this problem. Some artifacts still had to be manually edited away, in particular for the boneArc and plateCup scenes where the method cannot be applied as there is no planar surface that extends from side to side in these scenes.

The segmentation of the cyledge-l scene (see figures 34 and 35) contains three main segments: the segment, 1, at the top corresponds to the PaperRoll object, and the two large segments, 2 and 3, below correspond to two sides of the block object. Segment number 1, forms object region number 1 and the segments 2 and 3 form object region number 2. The results for this scene are shown in table 3. The table shows for each object region the results of indexing, matching, verification for each object in the model library. The number of solutions found after the matching and verification steps and number of nodes in the matching process are also given.

The algorithm as evaluated finds all solutions for the given object region and corresponding model. There will often be multiple solutions because segments match several surfaces in the model.

The segmentation algorithm does not have any problems with this scene as the edges are not very close to each other and there are no segments at large angles to the laser ray. The recognition algorithm also has few problems with the scene as the cylindrical surface only matches one cylindrical surface in the library, also the large areas of the planar surfaces of the block and the fact that two surfaces can
Figure 34: The cylinder scene (a) fused edges (b) closed edges

be seen makes the block easy to recognize. In this case the indexing identifies the true model as the only possible model. Segments 1 is small enough to match the two end surfaces of the block, this leads to the total number of 16 solutions. There are six surfaces of the block, number 1 to 4 are the sides and 5 and 6 are the ends of the block. The solutions found are sets of pairs between segments and surfaces (segment, surface), for the block the solutions are: \{1, 1\}, \{(1, 2), \{(2, 3)\}, \{(1, 3), (2, 4)\}, \{(1, 4), (2, 1)\}, \{(1, 2), (2, 1)\}, \{(1, 3), (2, 2)\}, \{(1, 4), (2, 3)\}, \{(1, 1), (2, 4)\}, \{(1, 5), (2, 1)\}, \{(1, 5), (2, 2)\}, \{(1, 5), (2, 3)\}, \{(1, 5), (2, 4)\}, \{(1, 6), (2, 4)\}, \{(1, 6), (2, 2)\}, \{(1, 6), (2, 3)\}, and \{(1, 6), (2, 4)\}. The first four corresponds to the block in one orientation and the four rotations. The next four is for the block in the opposite direction, and the last eight is for the smaller segment matching each of the ends of the block.
Figure 35: The segments and object regions in the cyledge scene. Each object region is shown in a different stipple pattern.
Table 3: Recognition results for the cyledge_1 scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The segmentation of the octCup scene (see figures 36 and 37) contains one main object region corresponding to four segments (2, 3, 4, and 5) of the octCup. There are also three object regions corresponding to the wall behind the octCup and the floor under it. The results for this scene are shown in table 4. For this scene the segmentation algorithm again has few problems. The individual segments from the object region are not very discriminating; the object models for Arc and Block also pass the indexing step. However, the binary constraint used in the matching step eliminates these models as solutions.

The segmentation of the boneArc scene can be seen in figures 38 and 39. There are two interesting object regions, the first corresponds to the Bone model and contains four segments (6, 7, 8, and 9), the second (segment 2) contains one surface from the Arc model. The recognition results for this scene are shown in table 5.
Figure 36: The octCup scene (a) fused edges (b) closed edges
Figure 37: The segments and object regions in the octCup scene. Each object region is shown in a different stipple pattern.

Table 4: Recognition results for the octCup scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>16/16</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5: Recognition results for the boneArc scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>8/8</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>0/0</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Pass</td>
<td>Fail</td>
<td>Pass</td>
<td>8/8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td>Fail</td>
<td>Pass</td>
<td>2/2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td>2/2</td>
<td>2</td>
</tr>
</tbody>
</table>

For this scene the segmentation routine has problems with the two jump edges on the front of the Bone object. The edge detection routines find combinations of concave and convex edges instead of jump edges. This is due to the low-pass filtering necessary for edge detection. There is also no calibration file for the range-finder's horizontal motor problem, which generates spurious edges. These problems were corrected by manually editing the fused edge image.

The Arc in the scene only gives one usable segment, and this in turn makes the object region match several more objects than the correct Arc model. This is due to not being able to use any of the more powerful binary constraints in the matching.

The segmentation of the octArc scene can be seen in figures 40 and 41. There are two interesting object regions, the first corresponds to the octCup model and contains four segments (5, 6, 7, and 8), the second contains two surfaces (segments...
Figure 38: The boneArc scene (a) fused edges (b) closed edges
Figure 39: The segments and object regions in the boneArc scene. Each object region is shown in a different stipple pattern.
Table 6: Recognition results for the octArc scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>16/16</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 and 3) from the Arc model. The recognition results for this scene are shown in table 6.

The segmentation of the octArcBone scene can be seen in figures 42 and 43. There are three interesting object regions: the first corresponds to the octCup model and contains four segments (14, 15, 16, and 17), the second contains two surfaces (segments 2 and 3) from the Arc model, and the last contains one surface (segment 11) from the Bone model. The recognition results for this scene are shown in table 7.

The segmentation of the plateCup scene can be seen in figures 44 and 45. There are two interesting object regions, the first corresponds to the octCup model and contains three segments (11, 12, and 13), the second contains six surfaces (segments 4, 5, 6, 7, 8, and 9) from the octPlate model. One interesting thing about this scene is that the matching finds 16 solutions that satisfy the unary and
Figure 40: The octArc scene (a) fused edges (b) closed edges
Figure 41: The segments and object regions in the octArc scene. Each object region is shown in a different stipple pattern.
Figure 42: The octArcBone scene (a) fused edges (b) closed edges
Figure 43: The segments and object regions in the octArcBone scene. Each object region is shown in a different stipple pattern.
Table 7: Recognition results for the octArcBone scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>16/16</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td></td>
<td></td>
<td>0/0</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>octCup</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>10/10</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>24/24</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>octCup</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>8/8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>8/8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>3/3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>2/2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>6/6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8: Recognition results for the plateCup scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>16/16</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The recognition results for this scene are shown in table 8.

The segmentation of the plateSoapArc scene can be seen in figures 46 and 47. The plateSoapArc scene consists of two interesting object regions, region one (segment 13) is the curved surface of the SoapBox model, and region 2 contains five surfaces (segments 6, 7, 8, 9, and 10) from the octPlate model. There is also an object region (segment 3) from the Arc model, but this region only contains a heavily occluded surface and can not be interpreted.

7.3.2 Evaluation

The experiments show that the system is working. There are some problems caused by the horizontal motor problem of the range-finder. There are also problems in some other areas of the segmentation algorithm.
Figure 44: The plateCup scene (a) fused edges (b) closed edges
Figure 45: The segments and object regions in the plateCup scene. Each object region is shown in a different stipple pattern.
Figure 46: The plateSoapArc scene (a) fused edges (b) closed edges
Figure 47: The segments and object regions in the plateSoapArc scene. Each object region is shown in a different stipple pattern.
Table 9: Recognition results for the plateSoapArc scene

<table>
<thead>
<tr>
<th>Object Region</th>
<th>Model</th>
<th>Index</th>
<th>Match</th>
<th>Verify</th>
<th>Solutions</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>octCup</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>1/1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>1/1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SoapBox</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>octCup</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>16/8</td>
<td>784</td>
</tr>
<tr>
<td></td>
<td>octPlate</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>16/8</td>
<td>784</td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Bone</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
<td>0/0</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Block</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PaperRoll</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SoapBox</td>
<td>Fail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The recognition algorithm works very well for object regions containing more than one segment. For object regions only containing one segment the unary surface constraints are not powerful enough to uniquely determine one matching model object. This could be corrected by having more powerful constraints for individual segments, e.g. by matching the segment boundary. Also the problem is partially due to the wide baseline of the triangulation range-finder.

The computational complexity of the algorithm is quite low due to the use of object regions. The maximum size of the search trees is about 2000 nodes which makes the time for the search negligible compared to the times required for segmentation and feature calculation.
Chapter 8

CONTRIBUTIONS

Object recognition systems involve algorithms from the whole spectrum of computer vision: from low-level processing to high-level reasoning. Research in object recognition therefore give a broad knowledge in computer vision. This dissertation resulted in a complete object recognition system. By having a complete system it is possible to demonstrate the capabilities and facilities of CAISR and CWRU in computer vision and object recognition. The system can also in the future serve as a starting point and comparison for new projects. The system is largely based on methods currently used, but there are refinements of these methods as well as new algorithms. The new contributions have been in the following areas:

- Auto-synchronized range-finder.
- Sensor models for edge detectors
- Edge fusion using Euclidean distance transform.
- Structured segmentation technique.
- Combination of best techniques for object recognition
- Range-finder simulator integrated into the object recognition algorithm.
Chapter 9

FUTURE WORK

Future improvements to the system are especially needed in the following areas.

- Range-finder. A new range-finder needs to be built. The inaccurate deflection motors should be replaced with more linear actuators with little back-lash, e.g. voice coil actuators. The baseline needs to be shorter and to compensate for the accuracy loss due to this a CCD array with more pixels is needed. The speed of the range-finder is normally limited by the signal levels of the CCD array, to avoid this a higher power laser and higher sensitivity camera/lens combinations could be used.

- Segmentation. The current segmentation technique uses median and Gaussian filtering to reduce noise levels. The Gaussian filtering in particular smears the images. Different noise removal techniques that better preserve edges should be investigated. A combination of edge-detection and region growing should also be investigated.

- Recognition. More and better constraints are needed for the recognition. The current algorithm works very well when two or more segments from the same object are seen, but often works quite poorly for single segment object
regions. To improve this, the recognition should also match the borders of the segments.

- Tracking. Much of the work has been done to integrate the range-finder simulator to the system. Some work is still needed in interfacing the object recognition results with the simulator input. Model based scene segmentation should also be investigated.
Bibliography


APPENDIX I: DERIVATIVE CALCULATIONS

In this appendix the equations for computing the 3D derivatives from the "stripe" image derivatives are derived.

The calculation of the 15 derivatives is done on demand from 5 stored intermediate derivatives $\frac{\delta x_r}{\delta u}$, $\frac{\delta x_r}{\delta v}$, $\frac{\delta^2 x_r}{\delta u^2}$, $\frac{\delta^2 x_r}{\delta u \delta v}$, and $\frac{\delta^2 x_r}{\delta v^2}$.

\[
\frac{\delta x}{\delta u} = \cos(\alpha_l) \frac{\delta x_r}{\delta u}
\]

\[
\frac{\delta x}{\delta v} = -\beta_{ures} \sin(\alpha_l)x_r + \cos(\alpha_l) \frac{\delta x_r}{\delta v}
\]

\[
\frac{\delta^2 x}{\delta u^2} = \cos(\alpha_l) \frac{\delta^2 x_r}{\delta u^2}
\]

\[
\frac{\delta^2 x}{\delta u \delta v} = -\beta_{ures} \sin(\alpha_l) \frac{\delta x_r}{\delta u} + \cos(\alpha_l) \frac{\delta^2 x_r}{\delta u \delta v}
\]

\[
\frac{\delta^2 x}{\delta v^2} = -\beta_{ures} \cos(\alpha_l)x_r - 2\beta_{ures} \sin(\alpha_l) \frac{\delta x_r}{\delta v} + \cos(\alpha_l) \frac{\delta^2 x_r}{\delta v^2}
\]

\[
\frac{\delta y}{\delta u} = \sin(\alpha_l) \frac{\delta x_r}{\delta u}
\]

\[
\frac{\delta y}{\delta v} = \beta_{ures} \cos(\alpha_l)x_r + \sin(\alpha_l) \frac{\delta x_r}{\delta v}
\]

\[
\frac{\delta^2 y}{\delta u^2} = \sin(\alpha_l) \frac{\delta^2 x_r}{\delta u^2}
\]

\[
\frac{\delta^2 y}{\delta u \delta v} = \beta_{ures} \cos(\alpha_l) \frac{\delta x_r}{\delta u} + \sin(\alpha_l) \frac{\delta^2 x_r}{\delta u \delta v}
\]

\[
\frac{\delta^2 y}{\delta v^2} = -\beta_{ures} \sin(\alpha_l)x_r + 2\beta_{ures} \cos(\alpha_l) \frac{\delta x_r}{\delta v} + \sin(\alpha_l) \frac{\delta^2 x_r}{\delta v^2}
\]

\[
\frac{\delta z}{\delta u} = -\frac{\beta_{ures}}{\cos^2(\beta_l)} (x_r + \delta z) - \tan(\beta_l) \frac{\delta x_r}{\delta u}
\]

\[
\frac{\delta z}{\delta v} = -\tan(\beta_l) \frac{\delta x_r}{\delta v}
\]
\[
\frac{\delta^2 z}{\delta u^2} = -\frac{2 \beta \text{ures}}{\cos^2(\beta_l)}(\tan(\beta_l)(x_r - d_l) + \frac{\delta x_r}{\delta u}) - \tan(\beta_l) \frac{\delta^2 x_r}{\delta u^2}
\]

\[
\frac{\delta^2 z}{\delta u \delta v} = -\frac{2 \beta \text{ures}}{\cos^2(\beta_l)} \frac{\delta x_r}{\delta v} - \tan(\beta_l) \frac{\delta^2 x_r}{\delta u \delta v}
\]

\[
\frac{\delta^2 z}{\delta v^2} = -\tan(\beta_l) \frac{\delta^2 x_r}{\delta v^2}
\]

(41)

The above \(x_r\) derivatives with respect to \(u\), and \(v\) are in turn calculated from the \(x_r\) derivatives with respect to \(u_c\) and \(\beta_l\). In the above equations \(\beta \text{ures} = \beta \text{ures} = \frac{2 \pi}{12800}\) for the current range-finder.

\[
\frac{\delta x_r}{\delta u} = \frac{\delta x_r}{\delta u_c} + \frac{\delta x_r}{\delta \beta_l} \beta \text{ures}
\]

\[
\frac{\delta x_r}{\delta v} = \frac{\delta x_r}{\delta u_c} \frac{\delta u_c}{\delta v}
\]

\[
\frac{\delta^2 x_r}{\delta u^2} = \beta \text{ures} \frac{\delta^2 x_r}{\delta \beta_l^2} + \frac{\delta^2 x_r}{\delta u_c^2} (\frac{\delta u_c}{\delta u})^2 + 2 \beta \text{ures} \frac{\delta^2 x_r}{\delta u_c^2} \frac{\delta u_c}{\delta u} + \frac{\delta x_r}{\delta u_c} \frac{\delta^2 u_c}{\delta u^2}
\]

\[
\frac{\delta^2 x_r}{\delta u \delta v} = \beta \text{ures} \frac{\delta^2 x_r}{\delta u_c^2} \frac{\delta u_c}{\delta v} + \frac{\delta^2 x_r}{\delta u_c^2} \frac{\delta u_c}{\delta u} \frac{\delta u_c}{\delta v} + \frac{\delta x_r}{\delta u_c} \frac{\delta^2 u_c}{\delta u \delta v}
\]

\[
\frac{\delta^2 x_r}{\delta v^2} = \frac{\delta^2 x_r}{\delta u_c^2} (\frac{\delta u_c}{\delta v})^2 + \frac{\delta x_r}{\delta u_c} \frac{\delta^2 u_c}{\delta v^2}
\]

(42)

In the above derivatives are the \(u_c\) derivatives can be estimated using median and Gaussian smoothing followed by the Sobel operators, or by fitting surfaces.

The equations for the multipliers are:

\[
\frac{\delta x_r}{\delta u_c} = -\frac{f \cos^2(\beta_l)(B_0 + (d_l - d_c) \tan(\beta_l))}{\cos^2(\beta_l - \beta \text{oc})(u_c - f \tan(\beta_l - \beta \text{oc}))^2}
\]

\[
\frac{\delta x_r}{\delta \beta_l} = \frac{\cos^2(\beta \text{oc})(f - u_c \tan(\beta \text{oc}))(f B_0 + u_c(d_l - d_c) - (B_0 u_c + (d_l - d_c) f) \tan(\beta \text{oc}))}{\cos^2(\beta_l - \beta \text{oc})(-u_c + f \tan(\beta_l - \beta \text{oc}))^2}
\]

\[
\frac{\delta^2 x_r}{\delta u_c^2} = \frac{2 f \cos^2(\beta_l)(B_0 + (d_l - d_c) \tan(\beta_l))}{\cos^2(\beta_l - \beta \text{oc})(u_c - f \tan(\beta_l - \beta \text{oc}))^3}
\]
\[
\frac{\delta^2 x_r}{\delta \beta_c \delta u_c} = \frac{B_0(-f + u_c \tan(\beta_l - \beta_0))}{\cos^2(\beta_l - \beta_0)(u_c - f \tan(\beta_l - \beta_0))^3} - \frac{\cos(\beta_l + \beta_0)}{\cos(\beta_l - \beta_0)} \frac{((B_0f + (d_l - d_c)u_c) - (B_0u_c - (d_l - d_c)f) \tan(\beta_l + \beta_0))}{\cos^2(\beta_l - \beta_0)(u_c - f \tan(\beta_l - \beta_0))^3}
\]

\[
\frac{\delta^2 x_r}{\delta \beta_c^2} = -2 \cos^2(\beta_0)(f + u_c \tan(\beta_l - \beta_0))(f - u_c \tan(\beta_0)) \frac{(B_0f + (d_l - d_c)u_c - (B_0u_c + (d_c - d_l)f) \tan(\beta_0))}{\cos^2(\beta_l - \beta_0)(-u_c + f \tan(\beta_l - \beta_0))^3}
\]
APPENDIX II: USER’S GUIDE

This appendix describes the main programs used in this thesis. The programs include. Note that the programs do not contain a lot of source code. The bulk of the implementation is in the ObjRec library.

rangeSegm Program that does complete segmentation on scenes generated by the range finder. It detects shadow regions and intensity edges from the intensity image. Jump and fold edges from the range image. It fuses the different edge images and shadow mask into one edge image. This edge image is then converted to an edge map which is then converted to loop map and segment map. The edges are closed using the loop splitting technique and small segments are deleted. After all of this is done the recognition algorithm can be applied.

The program is a client of the range finder and simulation servers.

The program has a graphical user interface using the XView libraries.

Recognize Program for segmentation and recognition of simulated scenes. First the simulated scene is generated then the scene is segmented and recognition algorithm can be applied.

The program has a graphical user interface using the XView libraries.

RangeCalibrate Program is used to compensate for the non-linear horizontal deflection stepping motor used in the range finder. By fitting a plane to a planar region of the scene the compensation can be estimated. The program generates for a scene 'scene' that contains the intensity file 'scene.pgm' and stripe file 'scene.str' a compensation file 'scene.cal'. This compensation file is automatically used by the other programs if found.
**RangeFinder**  The range finder program. The current program also contains the range finder server interface by Thomas Mueller [39].

In manual mode the program asks for a region to scan, the resolution to use for horizontal deflection and the scanning speed. The program then asks for scene name and generates two files that have the scene name as base and .pgm and .str extensions. These files contain the intensity and stripe images.

In server mode a client (rangeSegm) can connect to the server and directly get the scene data into memory. This is convenient as the range finder runs on another computer than the segmentation and recognition programs.

**ReSample**  Program to compensate for the non-linearity of the horizontal deflection stepping motor. This program is used to generate a new scene which has a corrected stripe file. It uses a scene which has been processed by RangeCalibrate as input. It then resamples the stripe image based on the compensations calculated by RangeCalibrate. The new scene should contain the intensity image from the old scene and the new stripe image.

This method of compensation normally gives somewhat better results than just using RangeCalibrate.

**SimServer**  Simulation server. Clients (rangeSegm) can connect to this server to get simulated scenes. Program by Thomas Mueller [39].

**Simulate**  Program based on the original simulator program by Scott Stiefel [28]. Generates simulated scenes.

**genPolygon**  Program to automatically generate object files of 'polyhedral cylinders'.

The Recognize Program

The Recognize program will be described in more detail in this section. This program simulates the scene it is given on the command line. The different scenes are looked up from the /usr/local/objRec/scenes directory.

It then analyzes the objects in the scene to come up with bounding boxes (in the image plane) for the different objects. The user then selects what region in the image plane to simulate. An example of the input is shown below:

```
># ./Recognize soap.scn
Reading soapbox.obj
Do you wish to move the range-finder ? (1 = yes) 0
Object 1 is at min (u, v) -877 246 max -603 535
Input the left border of the scan in motor steps: -900
Input the top border of the scan in motor steps: 240
Input the right border of the scan in motor steps: -600
Input the bottom border of the scan in motor steps: 540
```

The simulation is done with ray-tracing techniques and takes 1-60 seconds depending on scene complexity. After the scene is simulated the main window is displayed, as shown in figure 48.

The main window has parts:

**Image Window** This is the big window that displays different results based on display mode.

**Information area** This is the big area below the Iterator Controls. It displays different information based on the display mode. The information is updated
Figure 48: Main window for Recognize program

when the user clicks with the mouse on a new position in the Image Window
or when selecting a new item with the Iterator Controls.

**Iterator Controls** This is the item that has label either "Edge" or "Segm" depending on display mode and contains a identity number and up and down arrows. The identity number can be changed to display the information for another edge or segment.

**Quit** Push button that quits the program.

**Save** Push button that saves the simulated scene into scene files as generated by the range-finder. The user is prompted for scene name to save into in the window that Recognize was started from. Two files are created, both with file names with base as the user gave and extensions .pgm for the intensity file and .str for the stripe file.
**Sim. Recog** Push button that applies the recognition algorithm to the scene. The results are displayed in the window Recognize was started from. It will also save the solutions into the file "SOLUTION".

**Display** Exclusive choice buttons to select what should be displayed in the image window. The required computations will be done as needed before the results are displayed. The choices are:

- **Surface** Draws the different surfaces from the models in different colors. This result is directly obtained from the ray-tracing as the object and surface that the ray for each pixel hits.
- **Intensity** Draws the simulated intensity image.
- **Edges** Draws the edge map generated from the scene. The edges are drawn in different color depending on type.
- **Segments** Draws the segment map. The identity number for each segment is drawn at the centroid of the segment. Note that this is not always inside the segment.
- **Features** Draws the segment map as for "Segments" the information area now also display segment information.
- **Fused** Draws the fused edge image.
The rangeSegm Program

In this section the rangeSegm program will be described in more detail. This is the main program of the program package and implements mostly everything described in this thesis. This program can start with raw (or corrected using the RangeCalibrate/ReSample programs) and generate a complete recognition of the objects in the scene.

The input to the program can either come from range finder files, e.g. oct.1.pgm and oct.1.str, with an optional oct.1.cal for the oct.1 scene. It can also be used directly connected to the range finder server, in which case the rangeSegm program can be used to control the range finder and the range-finder’s data is fed directly to rangeSegm.

The final solutions are written to a file “SOLUTION”. This is a simple text file that lists the possible solutions for the different object regions in the scene.

Control of the program is via an XView interface, see figure 49. The main elements of the interface are described below.

At the top left corner there is a panel containing these items:

**Quit** Quit push button. Pops up a confirmation window.

**Load File** Loads range finder files to load new scene. Pops up a window where the user can type in the scene name. The images are loaded from the directory that the user’s environment variable IMAGE_PATH is set to.

**R-Finder** Uses the range finder server as source for new scene. Pops up a window where the user can control how to scan the scene.

**Load** Loads in existing segmentation results from files. Not completed.
Figure 49: Main window for rangeSegm program
Save  Saves segmentation results to files.

Inten  Selects the size of the Gaussian filter that is applied to the intensity image.

Range  Selects the size of the Gaussian filter that is applied to the stripe image.

Below this panel are four individual panels for control of the segmentation.
   The first controls the Data Mask, a.k.a Shadow Regions. The controls are the high and low thresholds to use in the extraction. The OK button recalculates the regions with the new values.
   The next three panels are for Intensity, Jump, and Fold Edges all have the items:

OK  Applies the new values and recalculates the edges.

Detect  The high threshold used in hysteresis thresholding algorithm.

Trace  The low threshold used in hysteresis thresholding algorithm.

   The Fold Edges panel also have additional Eps H and Eps K items, these are used in one of the Curv display modes.
   The next panel that says “Function” in its top-left corner has the following items:

Fuse  Calculates the fused image edges.

Close  Calculates the edge map, loop map, and segment map from the fused image edges. Also closes the edges (splits the loops).

Clean  Cleans up the segment map. This removes the very small segments. Also calculates the segment features used for recognition.
**Simulator**

**Gap** Controls how big gaps in edges can be closed in the edge closing (loop splitting) algorithm.

**Recognize** Calls the recognition algorithm.

The bottom panel on the left side controls the display modes.

The top row has buttons to select display of intensity image, the data mask and the range image. The range image display is done by shading based on the range normal. Below this is a line of buttons to select display of the edge evidence images. Associated with this is a slider to select a evidence threshold. The new row of buttons is used to display the thresholded edge images. These are the edge images used as input to the edge fusion. Finally there are two rows of buttons for display of fused edges, loop map, segment map and various displays of the curvature, normal, and Laplacian images.

In the bottom right corner contains a panel with information about edges, loops, segments, etc based on the current display mode.
The RangeCalibrate Program

The RangeCalibrate program is used to derive a compensation vector for the inaccurate horizontal deflection motor of the range-finder. This compensation is done for every scene.

This is done by finding a planar region in the scene that extends from side to side. The range data in this region is used to fit a plane and the average difference between the plane and the range data in each column in the region is attributed to the inaccuracy of the horizontal deflection motor. Using this assumption the inaccuracy can be mostly compensated for. The range images from the oct.1 scene generated before and after the calibration are shown in figure 50. The average distance to the surface is shown in graph 51 and the resulting horizontal compensation profile is shown in graph 52.

The input and output to the program was:
<pre>#> ./RangeCalibrate oct_l 10 30
The surface normal v =
  0.999742
  -0.0142137
  -0.0177111
Distance to surface d = 0.84125
Average error is 0.00149263
Max error is 0.00285123
Save to file: oct_l.cal
The surface normal v =
  0.999844
  0.00204418
  -0.017543
Distance to surface d = 0.841817
Average error is 0.000202443
Max error is 0.00180355
</pre>
Figure 51: Average distance to Surface for Calibration Region for oct.1 scene
Figure 52: Horizontal Compensation Profile for oct.1 scene
APPENDIX III: SOFTWARE

This appendix describes the library of classes and functions used to implement the algorithms and programs described in this thesis.

Early in this thesis work (1989) the author decided to use C++ as the language of preference. This language was chosen as it was similar to C, highly efficient, and object-oriented. A complete object recognition system requires a very large base of support software before it is usable and in managing this complexity a object oriented language helps greatly. The choice had its complications as the language has not yet (1994) been standardized and some things have changed in the C++ version used (g++ and libg++). The language is now pretty much settled though after several revisions of an ANSI draft standard. The language has also gained substantial popularity since 1989.

A lot of the software development work was done to figure out how to organize the software so that it was easy to understand and maintain. This was done by building up a trees of derived classes. The software was also organized so that all main algorithms were implemented as generally and with as few dependencies as possible. This was done by having most of the software in a directory tree as shown in figure 53. The main library is in lib/libObjRec. It contains the source code for the library libObjRec.a. The different subdirectories represent fairly independent parts of this library and are described below.

Besides the libObjRec library the package also uses the libg++, BLAS (-lcor), LINPACK (-lilinpack), and EISPACK (-leispack) libraries. The first can be obtained from Free Software Foundation at ftp://prep.ai.mit.edu/pub/gnu/libg++-2.6.tar.gz and the others can be obtained from a netlib site, e.g. netlib.att.com in directory/netlib. The libg++ library contains standard C++ parts such as iostreams, but also contains some other things such as Gaussian random number generators.
Figure 53: Directory tree for the Object Recognition software
and other mathematics functions. The BLAS library is for Basic Linear Algebra System. The LINPACK library contains Linear Algebra functions, and depends on the BLAS library. The EISPACK library contains Eigenvalue System functions and depends on the LINPACK library. The BLAS, LINPACK, and EISPACK libraries are all in FORTRAN.

Unless otherwise noted a class is normally implemented in two files, a header file and a implementation file. Both files have the name of the class as base and the header file and implementation files have .h and .cc suffix respectively.

The software was mainly designed and implemented by the author. The initial range-finder simulator was however done by Scott Stiefel and the client/server additions to the range-finder and range-finder simulator as well as initial verification software were done by Thomas Mueller.

**Supp** Support functions. Some mathematics functions, e.g fast median calculation and fast conversion between polar and rectangular descriptions of vector. Also several lists (single and double linked), sets (lists, and hashed), and queues used in other algorithms. The lists, sets, and queues are derived from libg++. Also contain support for reading and writing various image formats.

**Image** Contains the main image class Image, this is a virtual base class for the whole image hierarchy as seen in figure 54. Also contains the derived Image_1 class that is the base for the scalar image classes. The scalar image classes contain ImageBin a binary image class, Image8 a grey scale class where pixels are stored in 8-bit unsigned values, Image16 a class where pixels are stored in unsigned 16-bit values, Image32 where pixel values are stored in 32 bit 2's complements values, ImageFloat a class for single precision floating point images. From the ImageFloat class the ImageStripe class was derived, this class is used for the stripe images that the range finder
Figure 54: Class hierarchy for the Image classes
generates.

Also contains classes that operate on the scalar image classes. DataSegm is a class for blob coloring of binary images. Distance is used to calculate Euclidean distance transforms. ImageDer32 and ImageDerFloat are two derivative classes. They calculate the u, v, uu, uv, and vv derivatives for every point in the input image. The results are stored in fixed-point 32 bit 2's complements and single precision floating point values respectively.

Finally the Simulate class is used to store the results that the range-finder simulator generates.

**Range** The Range..1 class is the base class for the range image classes. There are the fixed point Range32 and the floating point RangeFloat derived classes.

The RangeFinder class is used to describe the current range-finder. It can convert between the stripe images and range images.

There are range derivative classes in fixed and floating point versions, RangeDer32 and RangeDerFloat, these calculate the u, v, uu, uv, and vv derivatives of the x, y, and z coordinates in every pixel. This is done by derivating the x, y, and z coordinates.

The StripeDerFloat class also calculates the u, v, uu, uv, and vv derivatives of the x, y, z coordinates for every point, but does so by calculating the u, v, uu, uv, and vv derivatives of the stripe image and from this calculating the derivatives of the x, y, z coordinates. It can also calculate the normal and curvatures for each point.

Curvatures can be calculated using the RangeCurv and RangeCurvMat classes. The RangeCurvMat is a modified version where it is possible to average some intermediate results to reduce noise.
Finally RangeNorm can be used to calculate the normals of every point of a range image.

**Edge** Contains the edge image classes. These are basically different edge detection methods. There is the base class ImageEdge that is derived from the Image class and the derived classes ImageZero, ImageMax, and ImageMin. The ImageZero class finds zero-crossing of the Laplacian and is used to find intensity and jump edges. The ImageMax and ImageMin are used to find peaks and ridges and valleys respectively. This is done by checking if each pixel is the maxima (or minima) for pixels along any line through this point.

There is also the classes for edge and vertex descriptions, these are called Edge and Vert. They are derived from the Ident class so that each edge/vertex will have a unique identity number.

**Fusion** This directory contains most of the sensor fusion and segmentation classes.

The EdgeEvidence class is used to calculate normalized evidence values for the different edge types. The class also contains thresholding with hysteresis member function that is necessary to extract good edges particularly from the fold edges. The EdgeEvidence class is derived from the ImageEdge class.

The SceneEvidence class packs the edge evidences for intensity, jump, and concave and convex fold edges into one object. This class is derived from the Image class.

The independently detected edges are fused into a composite edge image with the ImageFusion class. This is done with Euclidean distance transforms
and rules for the edge interactions. The ImageFusion class is derived from
the ImageEdge class.

The EdgeMap class converts edges stored in image format to linked list
format. The class is derived from the Image class.

The LoopMap class converts edge maps into loop maps. It also implements
the loop splitting. The class is derived from the Image class.

The SegmMap class converts loop maps into segment maps. It is derived
from the Image class.

The Loop and Segm classes contain the loop and segment descriptions, they
are derived from the Ident class.

The SegmFeatures class is used to calculate the different features for indi-
vidual segments. It is derived from the Ident class.

The LoopIter class is a virtual iterator base class that can be used to success-
vively get points in the loop. There are two derived classes LoopForwardIter
and LoopBackwardIter.

The SegmIter class is used to successively get the pixels that are contained
in one segment.

Object This directory contains the object descriptions used for recognition and
simulation. Some recognition specific parts are in the Recognize directory.
The class hierarchies for the object description classes is show in figure 55.

The model objects are described by the OR_Object class. It contains infor-
mation about surface material (ReflProps class), name of the object, number
of points and an array of the points (DPoint class), number of edges and an
array of pointer to the edges (DEdge class), number of loops and an array of
Figure 55: Class hierarchies for the Object Description classes
pointers to the loops (DLoop class), and number of surfaces and an array of
pointers to the surfaces (Surface class). It also contains a set of tuples that
describes what surfaces are neighbors. The class is derived from the Ident
class.

The DPoint class describes a 3D point.

The DEdge class is the virtual base class for the different edge classes. It is
derived from the Ident class.

The DstraightEdge class describes an edge of straight segments between a
list of points.

The DCircEdge class describes a closed or non-closed circular arc edge.

The DLoop class is the virtual base class for the different loop classes. It is
derived from the Ident class.

The DGenLoop class describes a general loop built up of any combination
of edges.

The DCircle class describes a circular loop built up of a single closed
DCircEdge.

The DPolygon class describes a polygonal loop built up of any number of
DstraightEdges.

The Surface class is the virtual base class for the different surface classes.
It is derived from the Ident class.

The PlanarSurf class describes planar surfaces bounded by any external and
internal loops.

The Conv4Surf class describes a planar surfaces bounded by DPolygon loop
with four corners.
The Cylinder class describes a complete or portion of a cylinder.

The Sphere class describes a complete sphere.

**ImgUtil**  This directory contains assorted image utilities. The util.h and util.cc files contain code to read in scenes from files.

The ColorMap.h, ColorMap.cc, displayX.h, and displayX.cc are used to describe the color-map used and to display different images and other classes in an X window.

The RangeFit.h and RangeFit.cc are used by the RangeCalibrate program. The fit a planar surface to a portion of a range image.

**minpack**  This contains FORTRAN code for the MINPACK package by Burton s. Garbow, Kenneth E. Hillström, and Jorge J. More from Argonne National Laboratory. This package is available from netlib, e.g. from netlib.att.com. It contains various optimization algorithms, including a modified Levenberg-Marquerdt algorithm used to fit spheres and cylinder to range data.

**Verify**  Contains some preliminary implementations of verification.

**Socket**  Contains some parts used in the implementation of the range-finder and simulation servers and clients.

**Finder**  Contains the range-finder server and client functions. The original range-finder code was modified to support the client/server functions.

**Recognize**  Contains the object recognition code.

The SegmGrouping class contains the description for an object region. An object regions consists of connected segments believed to come from the same object.
The SurfHyp class describes a hypothesis of what surfaces in a particular object a segment matches. It is derived from the Ident class.

The ObjHyp class describes a hypothesis of what segments a particular object matches. It contains SurfHyp objects.

The EdgeMatch.cc file implements match functions for the different edge classes.

The SurfaceMatch.cc file contains the match functions for the different surface classes.

The ObjectMatch.cc file contains the match and verify function for the OR_Object class.

Scene This directory contains the C_Scene virtual base class and the derived C_FileScene, C_ServerScene, and C_SimuScene.

The C_FileScene class is used to read in a scene from files generated by the range-finder. This is done via some functions in the ImgUtil directory.

The C_ServerScene is used to read in a scene from the range-finder server. See the Finder directory.

The C_SimuScene is used to read in a scene from the range-finder simulator. See the Simulate directory.

SceneUtil.h and SceneUtil.cc contain some utility functions to open and read from the various scene classes.

Simulate This directory contains the range-finder simulator and the client/server functions for it.

RFSimulator.h and RFSimulator.cc contains the main simulator.
The class SimSegm in the SIM_segmentation.h and SIM_Segmentation.cc files contain the mechanisms for segmenting the simulated scene.

The SIM_Client.cc, SIM_Client.h, SIM_Command.h, SIM_Server.cc, and SIM_Server.h contain the client/server functions for the range finder simulator.