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.

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.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

DISSERTATION

REASONING ABOUT OBJECT APPEARANCE IN THE CONTEXT OF A SCENE

Submitted by

Mark Richard Stevens

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 1999

COLORADO STATE UNIVERSITY

July 1, 1999

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY MARK RICHARD STEVENS ENTITLED REASONING ABOUT OBJECT APPEARANCE IN THE CONTEXT OF A SCENE BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

REASONING ABOUT OBJECT APPEARANCE IN THE CONTEXT OF A SCENE

Automated recognition of objects is complicated by occlusion. Algorithms that operate reliably when objects are in plain view typically fail when more than 50 percent of one object is obscured by another. The reason, in short, is that these algorithms fail to find the majority of the object and therefore conclude it is not present. Moreover, since previous techniques look for single objects in isolation, they cannot reason about the simple fact that one object is in front of another. Instead, the occluded portions of objects must be labeled as not found.

To explain the absence of occluded features, a recognition system must use knowledge about an object's relationship to other objects. This dissertation presents a method for recognizing occluded objects in the context of a scene. Partial scene models are constructed which explain absent features in terms of occlusion by other objects. The scene modeling exploits standard computer graphics techniques: a parameterized partial 3D scene model is used to generate a prediction of how an image will appear. Recognition is then cast as a search for the scene parameters which best match predicted to observed imagery.

Mark Richard Stevens
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 1999

TABLE OF CONTENTS

Chapter 1

Introduction

"And so it begins ...."

- Kosh

Occlusion greatly hinders the performance of object recognition algorithms since it is not a phenomena which can be predicted in isolation: occlusion is a function of an object's relationship to the scene in which it is embedded. Even though occlusion is determined by an object's relationship to other objects in the scene, automatic recognition algorithms seldom approach the problem in terms of multi-object interaction. Instead, algorithms focus on locating a single object in a single image. These techniques often explore possible matches between model features and homogeneous data features (e.g., matching model lines to data lines (Lowe 1985)). The search for correspondences takes place in *feature* space and is plagued by combinatorics: the number of possible model-to-data feature pairings grows exponentially with the total number of features (Grimson 1990c).

Searching for individual objects in isolation precludes explicit reasoning about occlusion. Although the absence of a model feature can be detected (i.e., no corresponding data feature), the absence cannot be explained (why is there no corresponding data feature?). As the number of missing features increase, recognition performance degrades (Stevens and Beveridge 1998). One way to prevent such degradation is to explain the absence of predicted model features in terms of occlusions. To construct such an explanation, there must be evidence indicating the existence of another occluding surface in the scene.

By switching the approach from searching for single objects to searching for multiple objects, knowledge about the interactions between objects in the scene is readily available. Knowing that one object in the scene physically lies between the sensor and another object

allows correct explanations about why certain features are obscured. Model-based object recognition algorithms must begin to use scene representations to explain object interactions if they are to improve in domains where occlusion is common.

1.1 What Does it Mean to Interpret a Scene?

A scene is defined as a subset of the world lying before a camera. The camera produces an image of the objects residing within the sensor field of view: a window into the world. At the extreme, scene interpretation is the process of labeling each pixel in the image with the names of the object that most likely produced the recorded measurements. Until each pixel has been explained, we cannot say with complete confidence that the entire scene has been explained.

In order to completely interpret the scene, the location and orientation of each object in the world must be recovered. In model-based vision, objects are typically stored as $3D$ models in a large database of known objects. A *model* stores any type of explicit knowledge about an object in the world. Once the scene has been interpreted, a virtual world of $3D$ models in their proper positions is created.

Take the case of three blocks lying on a table. Figure 1.1 shows two separate images: Figure 1.1a shows a scene where the objects are not occluded and Figure 1.1b shows objects being occluded. Assume for the moment that we are interested in only finding the block marked with the letter I (*I-block*). When presented with an image of the object without any occlusion, the problem is quite easy: all the pixels associated with the *I-block* can be located and explained (the white outlines show the object geometry).

Once occlusion is added to the scene, the problem becomes more difficult. When the *I-block* is located in the image with occlusion, a large portion of the object's projection does not match the image data. Even if the *I-block* was located precisely, reasoning about only that block and the sensor geometry would lead the system to incorrectly label many pixels. For example, most of the *V* face on the *V-block* obscures part of the image which would otherwise be labeled as *I-block*. In the single-object framework, no method exists for explaining the bad observed match. When moving to a multi-object scene-based inter-

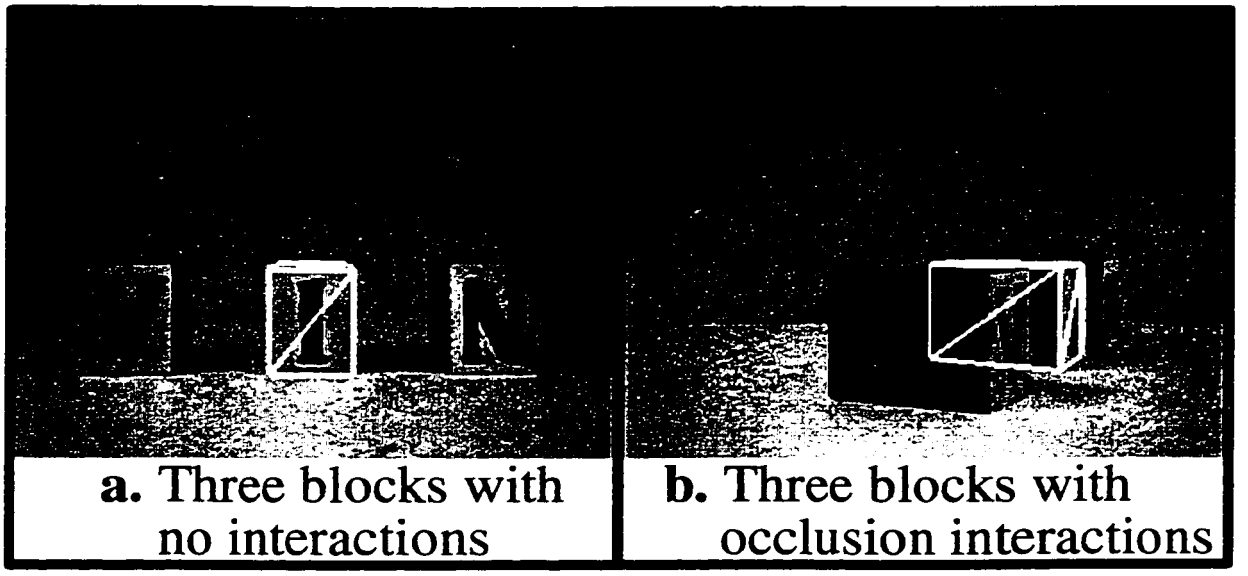

Figure 1.1: Two example images of three blocks lying on a table. One of the images contains no object occlusion, where as the other contains occlusions. The geometry of the *I-block* is overlayed on each image.

pretation, recognition is concerned with locating the *I-block* as well as the *V-block* and the *N-block*.

The absence of expected features for the *I-block* can now be explained with knowledge about the other blocks: portions of the object do not match well because another object (the *V-block*) is occluding a significant portion of the object. By shifting features from the missing category into the occlusion category, we have increased our confidence that the scene has been correctly explained. Thus, the entire scene context constrains the appearance of the individual object. In order to reason about scenes, a knowledge representation must exist. This knowledge about the world is captured by a scene configuration.

1.1.1 What is a Scene Configuration?

The appearance of an object is influenced by many factors: the time of day, weather, positions of the sun or other illumination sources, whether the object is indoors or outdoors, atmospheric conditions, time of year, color of the object, object surface and reflectance properties, camera type and characteristics, the non-object background environment, and the positions of other objects. We introduce the generic term *scene configuration* to encapsulate any information supplied about the world within which a sensor has been placed.

Given the large array of factors one might wish to model, one must decide where to draw the line in terms of configuration complexity. For this thesis, scene configurations consist of the objects believed to be present in the image, the location where they are believed to be present, and basic geometry and texture information describing object shape and appearance. Additional information certainly can and should be added to the configuration when the following conditions are met:

- The information can be explicitly inferred from the imagery.
- Including the information improves recognition performance.
- Interactions with existing information are minimal.

The most obvious information we have neglected to incorporate is scene illumination. While lighting information would increase the predictive power of the configuration description, it will also compound the complexity of the reasoning process.

1.2 Using Partial Scene Models for Object Recognition

Scene interpretation may seem quite difficult when viewed using previous model-based computer vision methods. For this dissertation, we employ a wide range of techniques more often associated with computer graphics and artificial intelligence than with object recognition. The duality between computer graphics and computer vision is shown in Figure 1.2. For computer graphics, we are given knowledge about the world in the form of models, cameras, lighting, etc., and are asked to infer an image of the scene. In computer vision, the exact opposite is required: an image is present, and we are asked to infer the configuration of the world. It is for this reason that vision is sometimes called the inverse graphics problem (Lengyel 1998).

The separation between the two fields does not need to be so forced. Instead, we use computer graphics techniques as a component in the computer vision process. By coupling the two processes, we form an iterative cycle where we first hypothesize knowledge about the world and predict the appearance of the scene. The prediction is then compared to the actual sensor image.

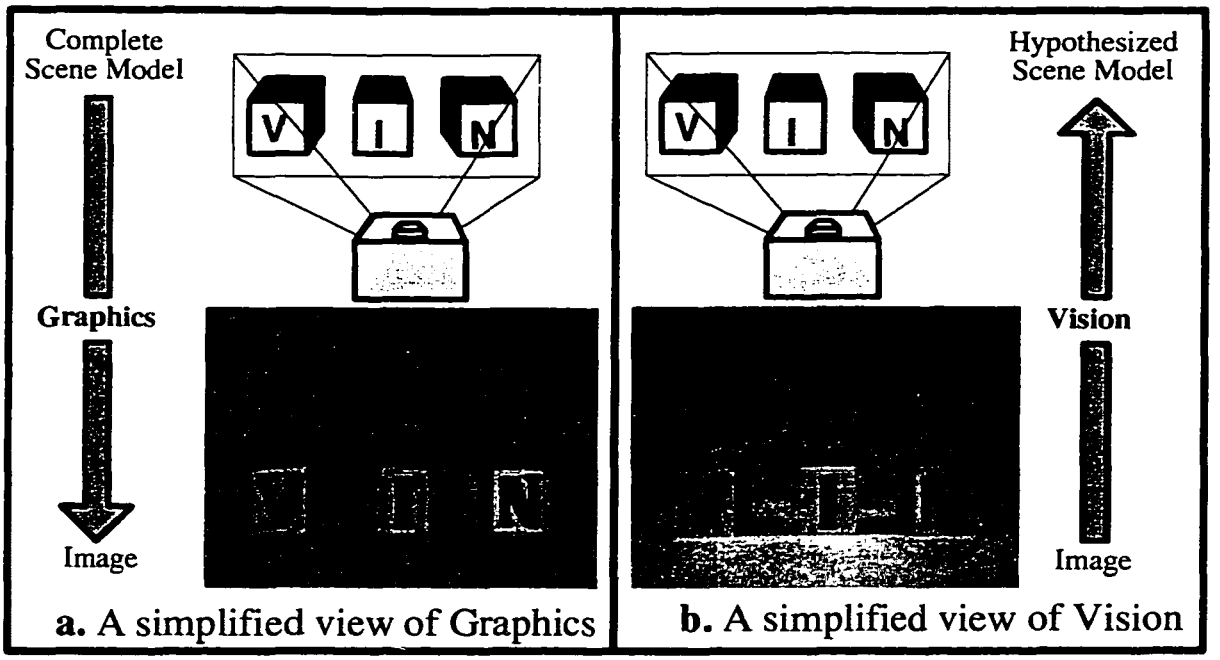

Figure 1.2: Relating the fields of computer graphics and computer vision.

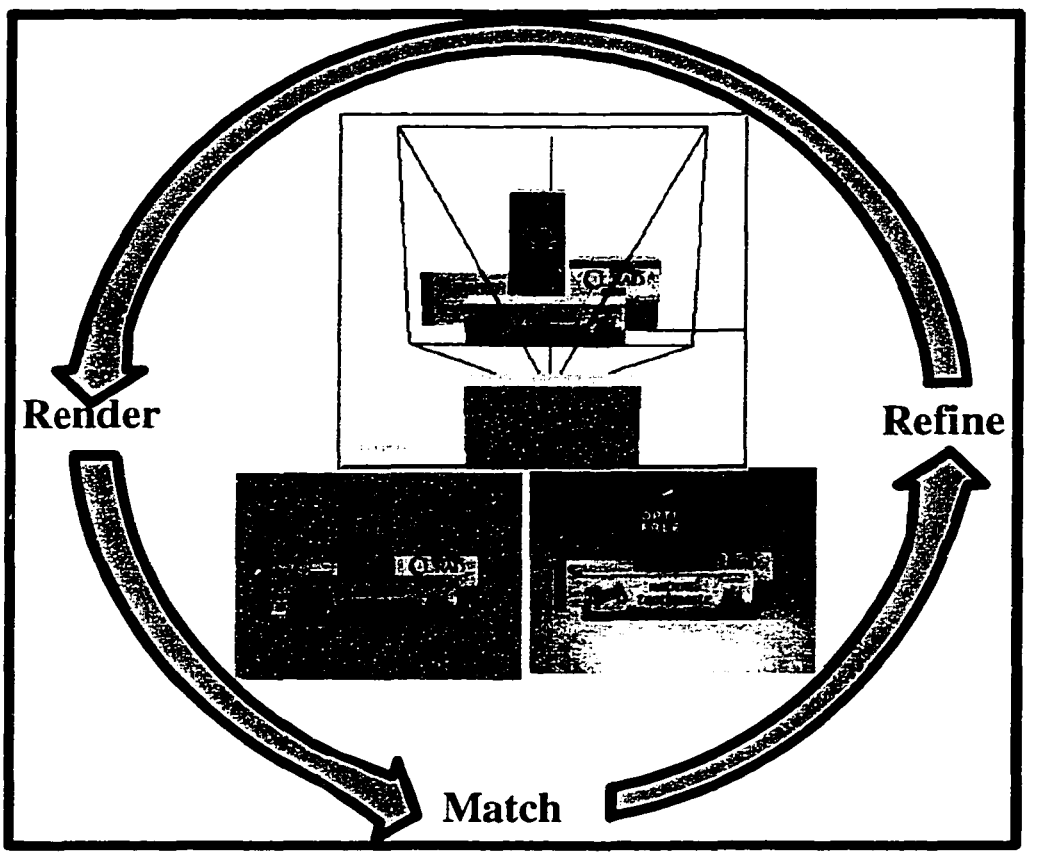

Figure 1.3: Using computer graphics as a component of computer vision.

Comparing the predicted and observed images provides insight into the quality of the scene configuration. As we make changes to the hypothesis, the prediction is re-generated and the effect on the match observed. The goal is to make intelligent decisions about how to change the hypothesis so as to bring the current scene configuration into alignment with reality. This iterative cycle, referred to as Render-Match-Refine or RMR (shown in Figure 1.3), is at the heart of our object recognition methodology. Three key questions are explored by this dissertation:

1. **Which computer graphics techniques should be used to produce the rendered image?** Given a hypothesized scene configuration, a prediction of object appearance is made. The prediction stage renders a single image based upon the hypothesized locations of each object. When an accurate model of the camera is employed, the rendered image will be at the resolution of the observed sensor image. Thus, a pixel-to-pixel correspondence is established between the two images. Chapter 3 describes in detail the actual rendering techniques used.

2. **What is the best method for measuring the quality of match between the rendered and observed images?** Given the pixel level correspondences, an error function is defined to measure the disparity between the predicted and the observed data. The goal of this measure is to rank the quality of a wide variety of hypothesized configurations: configurations closer to the correct solution should rank above those that are farther from the correct solution. Chapter 4 describes an extensive collection of possible error functions and compares which metrics display desirable properties for refining a hypothesis.

3. **How do we correct measured errors in knowledge of the scene?** Given a method for ranking predictions, a search algorithm is used to explore sets of possible scene configurations. During exploration, only examined configurations that are ranked highest are retained. Instead of searching for objects in feature space, a top-down generate and test strategy can search for objects in *scene* space using our partial scene model. A benefit of switching search spaces is the reduction in problem

dimensionality. The dimensionality of scene space is the cross product of the number of objects and their pose parameters (i.e., the six degrees of freedom to relate each object to a scene times the number of objects). This compares with feature space where dimensionality is a function of the number of model to data feature pairings.

As an example, imagine the combinatorics of matching 10 model features to a set of 100 possible data features. When examining possible model to data pairings, the size of the correspondence space is $O(2^{1,000})$. The problem becomes much more difficult when multiple objects are simultaneously considered: $O(2^{4,000})$ for 4 objects with 10 features each. In scene space, the dimensionality is fixed at 6 for one object and 24 for the four object case. Therefore, the exponential space associated with feature matching is replaced by a continuous space which grows linearly with the number of objects. Most importantly, with scene space search, multi-object recognition is practical.

A key assumption underlying RMR is the bootstrapping process: the iterative cycle must have a starting point. For the moment, we assume some pre-processing stage will be used for initialization. This assumption is clarified in Section 1.4. It should be noted that the idea for RMR is not new. In fact, the idea was proposed in (Besl and Jain 1986):

> ".. given a plausible model-based interpretation of a scene, the system could generate a synthetic image and process it as if it were real sensor data to obtain a new description. If the new description closely matches the real data description, the hypothesized model is verified; if not, the system must continue to search for a better interpretation ... [m]ore research in this area is needed."

There have been many examples of graphics techniques used for computer vision (Wells, Halle, Kikinis, and Viola 1996; Fua and Leclerc 1994; Camps 1993; Lengyel 1998; Hoff, Nguyen, and Lyon 1996). However, to the best of our knowledge, no one has used the tight coupling of graphics and vision to the extreme proposed by Besl and fully developed in this dissertation.

1.3 Example Problem Domains

A test domain is essential for demonstrating the results of an object recognition algorithm. According to Hanks *et al.* (Hanks, Pollack, and Cohen 1994), a good test-bed is challenging and able to highlight interesting aspects of a system's performance. In addition, the test-bed should aid in the explanation of why a system is performing the way it does for specific examples. To demonstrate the ideas presented in this dissertation, two different problem domains are examined. The first domain consists of colored, but not highly textured, blocks. The second domain contains highly textured products.

Figure 1.4 shows three color images from the blocks-world domain. The images in this dataset contain a wide range of object occlusions. At first glance, the blocks-world test domain may seem quite trivial. We have chosen to use this domain for two simple reasons. First, blocks-world has a rich history. Techniques such as Roberts' seminal work on line matching (Roberts 1965) and Waltz filtering (Waltz 1972) have both been demonstrated on blocks-world domains. Other traditional reference sources have also used blocks-world (Cohen and Feigenbaum 1982; Duda and Hart 1973). The second reason blocks-world was chosen is that it is often considered the base case for performance: our algorithms should perform well in this domain.

Figure 1.4: Three color blocks-world images.

Products-world is a more complicated version of blocks-world. The objects are all highly textured, with each image exhibiting varying levels of object occlusion. Figure 1.5 shows color images from products-world. We chose products-world for two reasons. The first reason was to emphasize texture. Many of the recognition techniques demonstrated on

blocks-world have considerable difficulty when complex surface markings are present. This is in part because many of these techniques require the extraction of line segments to be matched against lines from a model (Roberts 1965; Waltz 1972). For highly textured objects, these lines are extremely difficult to reliably extract, and even more cumbersome to explicitly model. The second reason products-world was chosen is that the domain has recently become a popular test set for a field of research known as appearance-based matching (Nayar, Nene, and Murase 1996). While appearance-based techniques are adept at solving problems involving objects in isolation, they have considerable difficulty with the realistic, structured occlusions present in our data set.

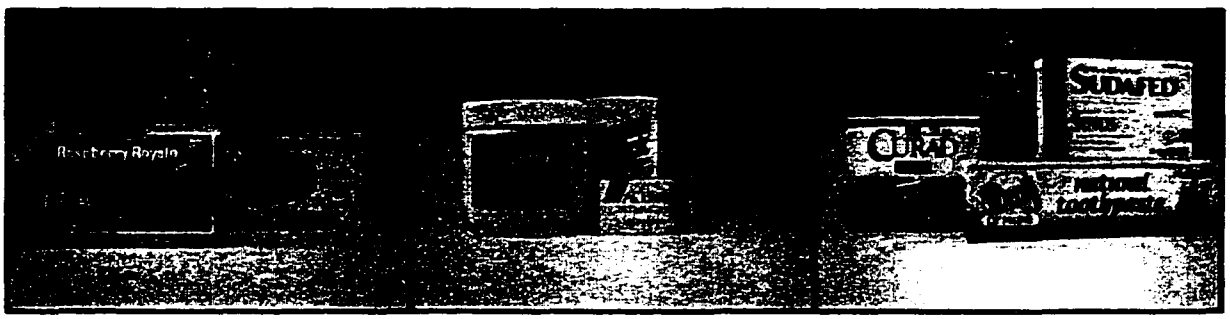

Figure 1.5: Three color products-world images.

1.4 Assumptions Made During the Thesis

During the course of this dissertation, several underlying assumptions are made about the scope of the problems being solved. For clarity, we will highlight those assumptions now to avoid any later misconceptions:

1. Illumination is fairly constant across images and is of a diffuse nature. The current RMR implementation does not explicitly model light sources. Chapter 4 discusses methods which could be incorporated to account for the effects of changes in scene illumination.

2. A pre-processing step known as indexing has already been done. Recognition has often been viewed as a three stage process: indexing, combinatorial matching, and

verification (this pipeline is further detailed in Chapter 2). Indexing determines what objects are present and their most probable locations in the image (Grimson 1990a). The next step, matching, finds possible locations of these objects in the scene. The final step, verification, takes all the information provided from the previous two stages and returns an interpretation of the scene. RMR focuses on matching and verification. For completeness, an algorithm is discussed in Appendix A for generating initial hypothesizes. The indexing phase used is a variant on a technique known as probing (Bevington 1992) and has been adapted to our specific problem domain.

3. Knowledge about object geometry and texture is available. As with most model-based algorithms, object models must exist before recognition can commence. In addition, the model representation must lend itself well to rendering. Representations which require extensive computation to produce a rendered image are not appropriate.

4. The models are distinct. The model representation must be unique so that measurements about the quality of the recovered scene configurations can be made. Ambiguities in model appearance will lead to incorrect scene interpretations.

1.5 Thesis Overview

The dissertation has been divided into chapters closely following each phase of the overall recognition process:

Chapter 2 focuses on relevant previous work in the field of object recognition. Special attention is paid to how traditional techniques cope with varying amounts of occlusion and texture. The chapter concludes with an overview of our own previous work with a focus on how the techniques presented here were developed.

Chapter 3 details the graphical rendering mechanisms used to predict the appearance of a scene. Complete information about specific representations is provided. These representations include the camera model, object representation, and the graphical rendering application programmers' interface (API).

Chapter 4 provides the mechanisms for comparing a predicted image to an actual sensor image. Twenty-seven different mechanisms are considered for measuring the discrepancy between two images, and these mechanisms are empirically compared. The evaluation provides insight into which mechanisms are best under the conditions considered in this thesis.

Chapter 5 defines five search strategies for generating hypothesized scene configurations based on the current hypothesis and information provided by the prediction and comparison mechanisms. This chapter also empirically compares the various algorithms on a set of example problems.

Chapter 6 discusses the experiment design and the statistical analysis of the entire RMR algorithm on 60 different scenes. A special focus is placed on analyzing how well the entire algorithm copes with varying amounts of occlusion.

Chapter 7 contains the future work and concluding remarks.

Appendix A details one method for generating the initial hypothesis required by the algorithms presented in the dissertation.

Chapter 2

Previous Work

"Standing on the shoulders of giants leaves me cold."

- *Michael Stipe*

In the purest sense, model-based recognition uses stored information about an object's shape and/or appearance to determine the orientation and position of that object in an image. Once the pose of the object has been recovered, qualitative measurements are made to support or refute the belief that the object is actually present in the scene at the hypothesized location. Typically, a set of sequential steps, referred to as a *pipeline*, are used in the model-based recognition paradigm.

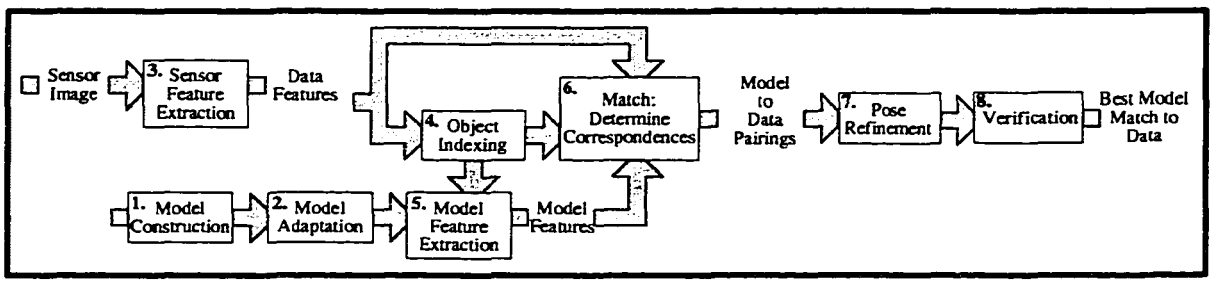

Figure 2.1: Detailing the various stages of the recognition pipeline.

The recognition pipeline was first hinted at in (Roberts 1965), discussed in (Marr and Hildreth 1980), and formalized in (Grimson 1990c). Since then, many technical enhancements have been added to each stage and the boundaries between specific tasks have been adjusted. However, the overall structure has remained intact. Figure 2.1 shows an overview of the entire pipeline. The pipeline begins with the construction of models and concludes with the best alignment of those models to imagery:

Model Construction: Prior to model-based recognition, a representation of the object must be constructed. Models can be obtained automatically from the data (Pope 1995), or constructed by hand using a CAD/CAM program. The output of the model construction phase is a collection of features about which evidence can be measured in an image.

Model Adaptation: Once the model is constructed, it can be adapted from a set of generic model features to those appropriate for a specific recognition domain. Usually these adjustments incorporate heuristic knowledge about the appearance of each feature for a given image obtained from a specific sensor (Camps 1993). Adaptation may also involve reducing the number of features present in the model (Stevens 1995).

Sensor Feature Extraction: Once the model has been constructed and adapted, the complex task of finding the object in the sensor data begins. The first step is to extract symbolic features from an image. The choice of extracted features is not arbitrary since the goal is to produce features similar to those already existing in the stored model. For instance, if the model consists of $3D$ line segments, the feature extraction routine will typically produce $2D$ straight lines.

Object Indexing : Given a set of data features, the indexing phase generates a list of objects which could explain regions of pixels in an image. Techniques such as geometric hashing (Lamdan and Wolfson 1988) or color histogram intersection (Swain 1990) produce an index into the model database from sets of feature groupings (Lowe 1985). The indexing phase is very difficult (Grimson 1990b) and may be considered ill-defined.

From an abstract point of view, the role of indexing is to provide the high level context. In the worst case scenario, the database contains heterogeneous models such as cars, beds, tables and refrigerators and the image may contain any number of these objects. For practicality, indexing might be given answers to questions of the following nature: Is the scene indoors or outdoors? How many objects might be present? Which room of a house did the objects come from? The more contextual knowledge the easier is the indexing task: knowing the camera is in the kitchen would increase the chance

of seeing a refrigerator and decrease the chance of a bed. Without such knowledge, indexing may be intractable.

Model Feature Extraction: Once the indexing phase is complete, features similar to those extracted from the data are extracted from the model. The model feature extraction stage of the pipeline is often overlooked (Plantinga 1988; Koenderink and vanDoorn 1979). However, choosing the wrong subset of model features can greatly complicate the later matching stages and may even cause it to be intractable (Stevens 1995). For example, if a model contains 1,000 features, we may wish to select only 100 that are most appropriate to the current recognition problem. By selecting the wrong 100, the object may never be located. If we select more than 100, computational demands may become too extreme. The simplest form of model feature extraction is the removal of self-occluded features using a process known as back-face culling (Foley and vanDam 1982).

Matching: A matching algorithm generates the most likely model-to-data feature pairings. A hard constraint exists on these pairings: the pairs must remain consistent under a single projective transformation (Lowe 1987b). For instance, if we find the windows on a house, they should not appear above the location of a detected roof. Instead, the house topology must be preserved.

Pose Refinement: Once a set of pairings is found, they are used to recover the position and location of the object in the scene. Typically, the pose parameters are solved using an over-constrained set of equations (Kumar 1992). Slight adjustments to these parameters allow for correction in object placement due to errors in sensor feature extraction. Often times, pose refinement is seen as a part of the previous matching phase (Beveridge 1993) with no distinction drawn between the two. We break the process into two stages: while a stronger model of the pipeline would have matching and refinement fused, many techniques do not address the problem in this manner.

Verification: The final stage of the pipeline is referred to as verification. Verification reduces the hypothesized list of possible objects and their corresponding locations

down to a single explanation of the scene. Pruning is accomplished by removing both false hypotheses (the hypothesized object is not actually present) and redundancy (similar objects occupy the same positions). After pruning, a definitive decision must be made as to which hypothesized configuration is correct. Probabilistic reasoning is sometimes used to find the most consistent configuration.

2.0.1 Contributing Individuals

Many researchers have contributed to different aspects of this pipeline (see Table 2.1 for a specific list). These contributors can be divided into three clusters by grouping authors who shared some commonalities in how they use two distinct sources of declarative knowledge: 1) geometric features of the object being sought, and 2) stored views of object appearance.

Many techniques have been proposed for utilizing both types of knowledge, but we will center on three distinct categories: geometric feature matching, augmented geometric feature matching, and a pattern matching technique based on Eigenvector analysis. The intention is to briefly summarize each of these three categories, paying particular attention to both how occluded objects are handled and the effects of texture.

2.0.2 Our Place in the Pipeline

As mentioned in the previous chapter, verification is the focus of this dissertation. However, additional contributions are also made to the pose refinement, model adaptation, sensor and model feature extraction stages of the pipeline. Our view on the final stages of the pipeline may be somewhat different from the traditional interpretation. If the last two phases (verification and pose refinement) can be strengthened, then inaccuracies can be tolerated in the earlier stages.

For instance, it has been argued elsewhere (Grimson 1990b) that indexing is one of the most difficult stages of recognition. If we relax the accuracy requirements on the indexing phase, errors in object hypothesizes will result. Relaxing the required accuracy of matching phase will introduce slight errors in pose. Both of these errors can then be corrected in the subsequent pose refinement and verification stages. Thus we shift the computational burden from the earlier stages of the pipeline to the later stages.

	Individual Contributor	Model Construction	Model Adaptation	Sensor Features	Model Features	Indexing	Matching	Pose Refinement	Verification
Geometric Feature Matching	Roberts			√		√			√
	Grimson					√	√		√
	Beveridge						√	√	√
	Lowe			√	√	√	√	√	√
	Huttenlocher				√	√	√	√	√
	Wells							√	√
Augmented Geometric Matching	Pope	√	√	√	√		√		√
	Camps		√	√	√	√	√		√
	Wheeler		√	√	√	√	√		√
	Hoogs		√				√	√	√
Appearance Matching	Nayar	√							√
	Camps	√							√
	Ohba	√							√
Render Match Refine	Fua	√							
	Wells						√	√	
	Hoff						√	√	
	Stevens		√	√	√			√	√

Table 2.1: Detailing the individual contributions to various stages of the recognition pipeline.

2.1 Geometric Feature Matching

Geometric object recognition deals with the search for correspondences between geometric model features, such as points, lines, planes, etc., and comparable features extracted from sensor data (Lowe 1987b; Huttenlocher 1998; Beveridge 1993; Wells 1993; Grimson 1990c). While a variety of different methods have been developed, most require the construction of a *correspondence set* in order to form a match. A correspondence set contains tuples of model features matched to one or more data features. To be considered valid, these matches must remain topologically consistent under a single geometric transformation aligning the model to the data.

As valid sets are constructed, likelihood measures are defined to ascertain match quality. This part of the process has been referred to as verification, with the end goal being to accept or reject the hypothesis that the object is present in the image (Grimson 1990c). These likelihoods are also used to rank one hypothesized object instance over another.

The role of geometric feature matching for object recognition is therefore twofold: 1) the construction of these combinatorial correspondence sets and 2) how to rank all constructed sets by how well they explain the scene. Most geometric methods assume that some form of indexing has already produced a set of objects known to be in the current image.

2.1.1 Effects of Texture

Geometric matching is dependent on the quality of the symbolic features extracted from the imagery. Moreover, it is commonly assumed that the extraction algorithm is separate from the matching process. Unfortunately, this assumption devalues the difficulty of sensor feature extraction. For instance, in complex scenes with many objects and non-uniform backgrounds, the extraction of edges from the 2D imagery is highly error-prone. Often, features are fragmented, displaced, or not even detected.

The feature extraction task is further complicated when looking for textured objects. In the products-world test domain, each object contains surfaces with complex markings. These markings often frustrate traditional low-level straight line extractors which either rely on regions of constant pixel intensity in order to locate step edges (Burns, Hanson, and

Riseman 1986) or group pixels along continuous edge chains (Boldt, Weiss, and Riseman 1989). Even if straight line segments could be reliably extracted from images, problems still arise with the model representation. Even simple products-world objects contain upwards of $1,000$ line segments. Such a large number is detrimental to algorithms which search the combinatorial space of model-to-data pairings. To make the problem painfully clear, assume a straight line detector is able to extract valid data lines from an image of a highly textured object. Realistically, we would expect upwards of $1,500$ lines to be extracted from a single image. If we have a model with $1,000$ lines, the worst case number of pairings is $2^{(1,000)(1,500)}$. The combinatorics further deteriorate when multiple objects are present (as was discussed in Chapter 1).

2.1.2 Effects of Occlusion

A pathology arises in the presence of occlusion. When objects become partially occluded, model to data feature pairings disappear from the correspondence set. A question arises: is it better to obtain a correspondence set containing a subset of the model matched with high confidence, or better to have a correspondence set which contains more of the model matched but with less confidence? In other words, how should the occlusion of features alter the ranking of matches?

Several approaches to the problem have been taken. Beveridge has introduced a measure which trades off the importance of finding model features and their quality of match (Beveridge 1993). Both Lowe (Lowe 1987b) and Huttenlocher (Huttenlocher 1998) have independently used an *explain-away* approach: as data features are considered matched they are removed from further consideration. Wells has dealt with occlusion by forcing the measure of match quality to be penalized for missing data features (Wells 1993). In most of these cases, confidence in the match measure will degrade as more model and data feature pairings are removed.

2.1.3 Individual Contributions

A variety of specific geometric object recognition techniques (listed in Table 2.1) are now discussed: Roberts examined the problem of comparing two wire-frame models (Roberts 1965),

Grimson searched for model-to-data feature pairings using an interpretation tree (Grimson 1990c), Beveridge iteratively constructed feature pairings using Local Search (Beveridge 1993), Lowe fused multiple stages of the recognition pipeline to streamline image interpretation (Lowe 1985), Huttenlocher developed a technique known as *alignment* to find the best transformation which aligns a model to data (Huttenlocher 1998) and Wells used statistical techniques for verification (Wells 1993).

2.1.3.1 Roberts: Putting Forth the Idea of Matching Models to Images

Roberts first defined the problem of machine perception as that of the search for correspondences between model and data features (Roberts 1965). He argued that matching $2D$ models to $2D$ data was a step in the wrong direction. His algorithm began with rudimentary feature extraction, followed by a simple grouping technique to generate a closed $2D$ line drawing from a gray scale image.

A mean-square error metric was used to determine if the polygons extracted from the imagery could be consistently explained by a model in the database. Brute force search returned all models topologically consistent with the data. The algorithm implementation relied heavily on the symbolic extraction routines to generate near-perfect models from the imagery. The algorithm was designed to interpret line drawings produced by a CAD/CAM sketching tool and determine the best set of $3D$ model primitives which could explain the observed $2D$ sketches.

2.1.3.2 Grimson: Using An Interpretation Tree for Search

Grimson *et al.* proposed using a data structure called the interpretation tree to find consistent matches between model and data features (Grimson 1990c). Grimson utilizes a Hough transform first for indexing and then to suggest a subset of possible features for matching (Grimson and Huttenlocher 1988). From this subset, an interpretation tree is constructed. The tree is used to search across possible model-to-data feature pairings. At the root of the tree is the null match (no correspondences). Each subsequent level in the tree represents a 2D data line matched to all possible model lines. Traversing the tree results in the construction of a set of one-to-many mappings between model and data features. Spe-

cial pruning algorithms based on local geometric consistency limit tree expansion. For each valid path, a verification process determines if the object is actually present: verification is important in order to guarantee global consistency.

For verification, a switch is made from correspondence space to image space (Grimson and Huttenlocher 1991). Thresholds are set to discard pairs in the interpretation which do not contain enough overlap in image space. Two different metrics can be computed: counting matched pairs, or overlapping feature length. Either metric is converted to a probability that the object is present. This probability is then compared to the probability that the feature pairings could have resulted at random. If the ratio of these two probabilities is high enough, the match is considered verified.

Different search strategies for traversing the tree have been examined, with Grimson favoring depth-first (Grimson 1990c) and others preferring breadth-first (Gaston and Lozano-Pérez 1984). One of the main contributions of Grimson's work is the analysis of stopping conditions for determining when an adequate portion of the tree has been traversed. At each level of the tree, both the probability the object is present and the probability of the random hypothesis are computed. Once the ratio falls below a threshold, the branch is discarded. When the branch is fully expanded, search is terminated. The best path through the tree is returned as the best model-to-data alignment. Grimson has analyzed various thresholds empirically using a variety of images with varying amounts of occlusion (Grimson and Huttenlocher 1991).

2.1.3.3 Beveridge: Matching as Combinatorial Optimization

Beveridge has cast feature matching as a combinatorial optimization problem (Beveridge 1993). Instead of defining the exhaustive interpretation tree for all possible model and data features in an image, a local search strategy is used. Matching begins with an initial set of model-to-data pairings, which can be generated randomly (Beveridge 1993) or with a key-feature algorithm (Beveridge 1998). An error term is then defined over the current match. Inherent in the heuristically derived error function are two terms designed to find the best set of correspondences. One measures fitness and the other omission.

Adjusting a match by adding or removing model-to-data correspondence pairs changes the value of the error function. The error term can be used in conjunction with a search strategy to explore the space of possible matches: each search neighborhood is based on varying a single model-to-data pairing. If enough matches can be explored, the best global model-to-data match can be found. Local search (Beveridge 1993) as well as genetic algorithms (Whitley, Beveridge, Graves, and Mathias 1996) have been applied to this search problem. Beveridge has shown good performance on several $2D$ problems as well as more difficult $3D$ problems in both outdoor and indoor scenes (Beveridge 1993).

2.1.3.4 Lowe: The Viewpoint Consistency Constraint

Lowe has developed a complete recognition system called SCERPO: Spatial Correspondence, Evidential Reasoning, and Perceptual Organization (Lowe 1987a). Lowe's approach provides a tight coupling of the indexing, matching and verification stages of the recognition pipeline. First, perceptual organization techniques are used to group low level data features into more reliable global features for matching (Lowe 1985). These global features are less likely to have arisen by accident and are more likely to belong to a structured object. Three different grouping strategies were exploited: proximity, parallelism, and co-linearity.

After grouping, a set of model *trigger* features ($3D$ features from the model most likely to be present in the $2D$ image) are identified. A computationally efficient algorithm is used to generate possible hypotheses as to where in the image the model most likely resides. Thus, indexing is done in a top-down manner as compared to the bottom-up method of the Hough transform. Next, for each hypothesis, model features are projected one by one into the image. Based on the image location of the projected features, corresponding data features are identified. Thus, the combinatoric nature of search for model-to-data correspondences during the matching phase has been greatly reduced.

The effect on the object pose of the new correspondences is computed using the iterative Newton method for solving a simultaneous set of equations. As more features are added, the Newton method will either continue to converge, or it will fail to converge in a fixed number of iterations. Lowe likens the convergence failure to a drastic degradation in the hypothesis

that the object is present. If all of the model features are successfully incorporated, the match is retained. Lowe refers to the final match in terms of viewpoint consistency (Lowe 1987b): all model features must align with corresponding data features under a consistent viewpoint transformation.

Lowe's approach has been demonstrated on several scenes with multiple objects and varying amounts of occlusion. The algorithm deals with occlusion by simply removing data features from consideration once they have been matched to a model. Removing matched features reduces the ability to account for missing features on subsequent objects.

2.1.3.5 Huttenlocher: Recognition by Alignment

Huttenlocher has presented an algorithm for matching $3D$ models to $2D$ data which shares many commonalities with Lowe's viewpoint consistency constraint. The algorithm, called *alignment*, is based on the notion of finding matches which best align the model and data features (Huttenlocher 1998). Similar to Lowe, the process begins by determining an initial match between a subset of model features and the given data. However, Huttenlocher uses smaller, local *edgel* features for matching as opposed to the larger line segments found by Lowe's perceptual grouping. Huttenlocher claims that the likelihood of finding a local feature is higher than that of the global features, since perceptual grouping is highly error prone in the face of large amounts of sensor noise (Huttenlocher 1998).

Next, Huttenlocher projects the entire model into the scene to determine if the object is actually present. Huttenlocher has developed a closed form solution, based on subsets of model features, for determining the optimal pose estimate. Lowe relied on an algorithm which required several iterations each time a new model feature was added to the scene. To obtain the closed form solution, Huttenlocher assumes orthographic projection as opposed to Lowe who used full perspective.

The merits of orthographic versus perspective projection are at the heart of differences between alignment (orthographic) and the viewpoint consistency constraint (full perspective). Huttenlocher argues that the underlying assumptions that went into developing the SCERPO system are not valid for perspective sensitive problems. Essentially, perspective

neither preserves the angles between straight lines nor their distances. Therefore, grouping based on parallel line features will be incorrect (Huttenlocher 1998).

The alignment algorithm has been demonstrated on several indoor and outdoor scenes (Huttenlocher 1998). The objects located were more complicated than those used by Lowe mainly because the edgel-feature approach has the ability to recognize a more diverse set of curved objects. Huttenlocher deals with occlusion in the same manner as Lowe: as data features are matched to the data, they are removed from consideration by other objects.

2.1.3.6 Wells: MAP Estimation

Wells has developed an algorithm for locating objects based on Maximum A-posteriori Probability (MAP) estimation (Wells 1993). MAP requires that a set of correspondences between model and data features already exists: Wells used Huttenlocher's alignment algorithm to generate these sets. Using a Bayesian framework, the problem is cast as optimizing the likelihood that the object is present given the data conditioned by the prior probabilities placed on the scene and the model. Verification is accomplished by choosing the configuration with the highest associated probability.

The MAP estimation technique has been demonstrated on images involving multiple viewpoints of a car model on a desktop. In most cases, the algorithm was able to correctly locate the vehicle in the scene. Wells points to one underlying assumption made which may not be valid in all domains. To use the Bayesian framework, the model features are treated as independent. He notes that this assumption is not valid for occlusion since occlusions are locally correlated. However, he says the independence assumption did not inhibit the results obtained, and suggests an alternative method based on Markov modeling to alleviate the assumption.

Uncertainty in the set of model and data features is handled with a prior probability. This prior probability must deal with numerous forms of uncertainty: feature occlusion, feature omission, sensor noise, and data feature extraction errors. Attempting to implicitly account for the numerous forms of uncertainty in a single term can be problematic. Estimat-

ing the priors for a single-object domain has been shown to work well, but the multi-object case was not examined.

2.2 Augmented Geometric Matching

Geometric matching assumes a fixed importance on all features in a stored model. Intuitively, this notion seems like a poor model for predicting feature visibility. For instance, consider an edge formed by the junction of two faces on a 3D model. Under certain lighting conditions and viewing angles, that feature may appear prominent when imaged. Under other conditions, it may not even be detectable.

A more expressive representation would characterize the conditions under which a specific model feature should be observed in an image. A matching algorithm could then weigh the evidence supporting a given feature by the belief it should be observed. The new representation not only contains the geometry of feature F_i, but also a probability density function (pdf) governing its predicted visibility:

$$P(F_i|S, E, L, V) \tag{2.1}$$

where the probability of matching the feature (F_i) is dependent upon the sensor being used (S), the feature extraction algorithm (E), the illumination in the scene (L), and the viewing angle (V). This probability can be used to weigh each observation by the expectation. Therefore, in configurations where the lighting and viewing angle imply that there is little likelihood of detecting a feature, the weighting of that feature in the match will be low. Recognition using additional information is referred to as *augmented geometric matching.*

2.2.1 Effects of Texture

Augmented geometric matching should perform better when recognizing highly textured objects then when using geometry alone. The reason is augmented representations are intuitively designed to compensate for the shortcomings of a specific data feature extractor (Camps 1993; Hoogs and Bajcsy 1996). If an object feature is not reliably detected due to failures in the feature extractor, it should have a low associated probability and should

therefore be removed from the match. Thus, model features associated with volatile texture will typically be discarded during recognition.

Augmented techniques still suffer from the problem of representing highly textured surfaces. Models will contain the thousands of geometric features necessary to characterize textured surfaces. The combinatorial problems have not disappeared, but they can be reduced since there is now a method for ranking the best features to use in a match. If there are 1,000 model features, an initial match could be formed based on five percent of the most likely features. The match can then be iteratively filled out using successive sets of significant features.

2.2.2 Effects of Occlusion

Unfortunately, most of the augmented techniques can be highly sensitive to occlusions. Problems arise when the algorithms rely upon off-line analysis to predict future object appearance. Over all views in the training set, certain features may or may not be visible. Generalizing a likelihood for a feature based on a set of such observations can degenerate into an extremely weak model of occlusion. For example, while some point feature F_1 has an apriori probability of being occluded equal to 0.6, and another point feature F_2 has an apriori probability of 0.1, in any given image the feature either is or is not occluded[1].

Use of priors implies an implicit bias toward some scene configurations over others. For example, a system choosing between the situation where F_1 is missing versus the situation where F_2 is missing would favor the case where F_1 is missing. Some percent of the time this will be the wrong choice since there are presumably configurations where F_2 is occluded but F_1 is not.

If occlusion is truly a function of a dynamic scene, as we believe, prior knowledge of static appearance may not improve performance when portions of an object are occluded. Recognition quality may degrade when features with a high apriori probability of being

[1]We neglect issues arising out of image sampling. Obviously, at the pixel level, image projection and quantization can lead to local ambiguities. For features with spatial extent, only a portion may be occluded. For simplicity of argument, we assume a feature is either completely occluded or completely visible.

found are occluded. One problem is integrating occlusion into a pdf model for a set of features: F_i, $1 \leq i \leq k$. In domains with multiple objects, the pdf will be conditioned upon the relative placement of objects:

$$P(F_i|S, E, L, V, O_A, O_B, ..., O_N) \tag{2.2}$$

where $O_A, O_B, ..., O_N$ represent N objects at specific locations in a scene. For some domains, such as building identification in aerial imagery, computing such probabilities may be simplified because the scene is static and the only free variable influencing occlusion is the camera viewpoint. In this case, the occlusion pdf follows directly from the pdf defining the possible views.

However, in non-static environments, characterizing the pdf is difficult, and the meaning is problematic. Consider two blocks on a table. To estimate $P(F_i|S, E, L, V, O_A, O_B)$ for the features in one model, we would need either a pdf defined over the space of object configurations or a large training set showing the possible ways that the two objects could interact. Adding another object would require either a more complex pdf over the three object configurations, or alternatively a larger training set. Finally, given many objects, it is not clear that a pdf defined over all views and configurations would be helpful in establishing the precise view and configuration for an observed object instance.

2.2.3 Individual Contributions

Several different methods for incorporating augmented information into a model have been proposed (Camps 1993; Wheeler and Ikeuchi 1995; Pope 1995; Hoogs and Bajcsy 1996). In some cases, the training information comes from images which are artificially generated (Camps 1993; Wheeler and Ikeuchi 1995), and in others the information is from sensor data (Pope 1995; Hoogs and Bajcsy 1996). The augmented information can either contain likelihoods that each feature will be observed given its presence in the training data (Pope 1995; Hoogs and Bajcsy 1996) or the best detection algorithm to use for locating that feature (Camps 1993; Wheeler and Ikeuchi 1995). In all of these cases, the goal is to extend the model representation to make it more appropriate for matching an object in new imagery based on past experience.

2.2.3.1 Camps: Rendering for Augmented Models

Camps has developed a system called PREMIO which utilizes complex sensor models, detailed lighting models and CAD models to generate what they call a *prediction model* (Camps 1993). The prediction model captures two different attributes of model features: the likelihood of finding a feature in a given image and the best detection method for locating that feature in an environment. Currently, PREMIO supports five different features: closed curves between surface boundaries, a curve on a surface, an occluding contour curve, the junction of two curves, and the cusp point of a curve.

Using detailed lighting, sensor, and object models, it is possible to realistically render objects from a wide variety of camera and lighting positions. For each rendered image, a multitude of different feature extraction algorithms can be run. By assessing the performance of the different algorithms, a characterization of how each feature will appear in the image is obtained. This information is used to construct the likelihood of finding a feature given the specific extraction algorithm for the current sensor.

Camps uses an augmented model to reduce the computation associated with matching models to features. First, the prediction models provide information about which feature detectors are most likely to produce useful information. At run-time, only the appropriate detection algorithms need to be run. Second, the prediction models make use of the probabilities associated with each feature to reduce the matching combinatorics. Data features which do not conform to the expected distributions can be removed from consideration. Thus, the possible features to use for matching are pruned by the predicted expectations.

2.2.3.2 Wheeler: Rendering for Augmented Models

Wheeler *et al.* have also developed a system which uses rendering to generate an augmented model (Wheeler and Ikeuchi 1995). In their vision algorithm compiler (VAC), the physics of a range sensor are accurately modeled. In range data, the only variable affecting appearance is viewpoint (illumination does not influence the imaged result).

Similar to Camps, the VAC architecture begins by rendering an object from many different viewpoints using known sensor characteristics. The resulting images are fed into

a general purpose segmentation algorithm, and several area and moment measures for each planar object face are taken.

Using Markov Random Fields (MRF), the likelihood that each feature is present over the range of possible viewpoints is computed. The augmented model contains empirical generalizations about the area of each planar face expected to be found versus the area obtained with formal analysis. For instance, the actual physical area of an object face in the scene will be different from that visible in the imagery. Wheeler's approach has been demonstrated on several different objects such as staplers and tape dispensers. Using the MRF method, the system can handle partially occluded objects by reasoning about spatially adjacent portions of the model having a similar quality of match with the data. A relaxation method is used to propagate the effects of occlusion to neighboring model features.

2.2.3.3 Hoogs: Augmented models from Real Data

Hoogs *et al.* have taken a different approach to the creation of augmented models. Rather than assuming complex renderings can fully capture the nuances of outdoor scenes, feature attributes are learned directly from a set of training images (Hoogs and Bajcsy 1995). The system requires a set of hand-registered model-to-image sets taken from various viewpoints and with a range of lighting conditions.

From the training data, edges are extracted. These edges are paired with the model features and used to construct a pdf over each model edge. This pdf represents the likelihood that an area of the model edge will be detected. The pdf has two free parameters for each feature: viewing angle and lighting angle. Hoogs has found that a very coarse sampling of these two parameters is sufficient to accurately model the true pdf. Coupling the resulting probabilities with a pose refinement algorithm (Hoogs 1996), recognition was greatly improved (Hoogs and Bajcsy 1996).

Hoog's method has been specifically designed to locate buildings. Buildings have the nice property of being stationary, rigid, simple to model and enclosed within a static environment. As such, the learned information will tend not to change significantly over time.

Since object composition and placement is fixed, self occlusion and multi-object occlusion are essentially equivalent.

2.2.3.4 Pope: Learning Complete Appearance Models

Realizing the limitations of requiring a pre-existing model for matching, Pope developed a system to learn model feature attributes directly from a set of training imagery (Pope 1995). The process begins by grouping training images into sets of characteristic views. For each of these views, seven types of features are extracted from the images: curves, junction of two curves, two junctions joined by a common curve, three junctions joined by two curve segments, parallel curves, a convex region of curve segments and a nearly enclosed circle or ellipse.

A relational graph is constructed over these features, providing information about which feature appears in each view. The number of times the feature is present for the characteristic view is used to estimate the probability that the feature will be detected in future imagery. The same feature may appear in several different characteristic views and have a different probability of detection. In essence, the likelihood of a feature being detected is a function of pose. The detected features, the view graph and the likelihood estimates form the augmented model.

During recognition, match scores evaluate the quality of the alignment between model and data features. This measure is based on what features are paired, the significance of those features, and how well their positions align. A Bayesian framework is used to combine the evidence across features. An iterative alignment technique based on (Lowe 1985) is used to adjust features in the current match.

The method has been shown to work well for objects in office settings, as well as scenes with high amounts of clutter and occlusion (Pope 1995). Pope's system appears to be highly versatile and easily adaptable. By simply adding enough training imagery, any object can be recognized regardless of the setting and environment. While the system has been demonstrated in the presence of occlusion, the type of occlusion present dictates

performance. For instance, if the object is ninety percent occluded, but most of the learned features are visible, the system will perform perfectly.

2.3 Appearance Matching

Other methods have completely abandoned the use of geometric models in favor of view-based information (Nayar, Nene, and Murase 1996; Ohba and Ikeuchi 1996; Edwards and Murase 1997). Off-line Eigenspace analysis of a set of training images produces a set of orthonormal basis vectors. The lower order vectors, as determined by their Eigenvalues, are discarded. These low order vectors represent dimensions of the data along which there is very little variance: almost all of the data for the dimension share the same value. The idea is that training images contain significant amounts of redundancy; by reducing the dimensionality, the redundancy is also removed. For instance, if a vector with 100 elements is embedded in a 1000 dimensional space, then only 100 values are needed to fully represent the data. The Eigenvector approach allows the determination of which 100 dimensions are important out of the 1000 total.

When a new image is provided, it is projected into the lower dimensional subspace using these basis vectors. Any number of distance metrics can be used to compute a measure of similarity between the new sample and the closest training sample. The computation involved is significantly less than if the distance was measured in the original image space. Due to the extensive reduction in computation, recognition can be performed in near real-time (Nayar, Nene, and Murase 1996).

One of the obvious drawbacks of this technique is the large number of training examples required to reliably recognize a given object from an arbitrary viewpoint. Assuming for a minute that the object is constrained to lie flat on a table, then there are really only three degrees of freedom: one rotation, a translation away from the sensor, and a translation parallel to the sensor image plane (see Section 3.2.1 of Chapter 3 for more details on this constraint). Neglecting the effects of perspective distortions as the object is moved parallel to the image plane, there are two degrees of freedom.

The first degree of freedom is object rotation. For appearance matching, typically 72 views of the object rotated every five degrees are used. The second degree of freedom is object scale. The problems of scale have not been substantially addressed in the literature. Nayar copes with scale by re-sampling each object to a fixed dimension (Nayar, Nene, and Murase 1996). Bischoff has looked at multi-scale representations for the Eigenvector basis set (Bischoff and Leonardis 1998). Neither completely solves the problems of scale, and therefore significantly large training sets are needed just to capture these two degrees of freedom.

2.3.1 Effects of Texture

The appearance matching techniques are extremely well suited for recognizing highly textured objects (Nene, Nayar, and Murase 1996). Since no explicit geometric features are stored, the issues associated with combinatorics are not relevant. All that is required is that the complex surface markings must be captured within the set of training views. However, texture will play an insidious role in the early stages of the appearance matching process.

While the view-based approach does not require the extraction of straight lines segments for matching, it does rely on image segmentation. In this context, segmentation is the process of isolating those pixels belonging to the object from the rest of the scene. Segmentation is known to be a difficult task (Beveridge, Griffith, Kohler, Hanson, and Riseman 1989; Haralick and Shapiro 1985) with algorithms typically requiring numerous parameters that dictate the quality of the segmentation.

Over or under segmentation of the object can be devastating to appearance based matching. While the Eigenvector approach has been applied to numerous scenes with varying backgrounds, most of these scenes were either generated by artificially inserting the object into a complex scene or using a constant background which is easy to segment and remove. Few scenes containing multiple objects have been used where an actual sensor produced the image data. Thus, when both objects and the background are highly textured, it may be impossible to obtain a correct segmentation.

2.3.2 Effects of Occlusion

During the computation of the Eigenvector basis set, each pixel in an image chip is considered. When a new image is presented for on-line matching, each pixel in the new chip is again used. As portions of an object become occluded, incorrect object pixels will be present in the subspace projection. These incorrect pixels shift the location of the projected point in Eigenspace away from where it should have projected without the occlusion. Hence, even modest occlusion can have detrimental effects on the algorithm.

It is for this reason that Ohba *et al.* have used smaller regions covering only portions of objects (Ohba and Ikeuchi 1996). Others have performed off-line analysis in attempts to limit the effects of occlusions (Edwards and Murase 1997). However, it is extremely difficult to predict all possible forms of occlusion. Even under quite modest assumptions, the possible ways a single solid object might occlude an image chip grows exponentially as a function of the image size (Stevens and Beveridge 1998). Clearly, efforts to develop training data for dealing with occlusions will be profoundly hampered by this relationship. One last approach that might be thought of as a solution to the combinatorics of occlusion is to break the model into small parts. Then one might expect those parts that are fully visible to be reliably detected (Huang, Camps, and Kanungo 1998).

2.3.3 Individual Contributions

A variety of appearance-based recognition techniques (listed in Table 2.1) are now discussed: Nayar applied Eigenvector analysis to the object recognition problem (Nayar, Nene, and Murase 1996), Ohba used a similar technique on a bin of parts problem (Ohba and Ikeuchi 1996) and Camps examined method for recognition objects based on parts (Huang, Camps, and Kanungo 1998).

2.3.3.1 Nayar: Eigenvector for Object Recognition

Nayar and Murase were the first to apply the Eigenvector techniques to object recognition (Nayar, Nene, and Murase 1996). Prior to that, Eigenvectors had been used successfully to recognize human faces (Kirby and Sirovich 1990; A.P., Moghaddam, and Starner 1994). Their Software Library for Appearance Matching (SLAM), should be considered a

strong first demonstration of the underlying methodology. Using the SLAM system, Nayar *et al.* were able to obtain real-time recognition of over 100 complex objects (the objects were both complex in terms of shape and surface textures). To accomplish such a task, an efficient matching algorithm was developed to quickly determine the closest training view in Eigenspace for a newly projected image (Nayar, Nene, and Murase 1996). However, many assumptions were made about scene lighting, image segmentation and occlusion, which greatly reduced the generality of the approach in arbitrary domains.

Realizing the limitations of these assumptions, Nayar has recently focused on characterizing the effects of Eigenvector matching with partial data (Hadjidemetriou and Nayar 1998). Here, they compare three different techniques for selecting subsets of pixels to be used in the Eigenvector analysis (instead of the entire image): random selection, window selection and heuristic cluster selection. The random selection algorithm does just what the name implies (selects subsets of pixels randomly). The other two selection mechanisms pick pixels in the image which maximize reconstruction accuracy: the subset is projected into the lower dimensional Eigenspace, and then projected back into image space. The difference between the reconstruction and the original image provides a residual error. Lower error leads to better pixel selection. The windowing algorithm picks the subset from fixed size windows in the image, and the heuristic clustering picks them based on individual pixels and how they effect the residual error.

Both selection algorithms demonstrate improved results over the random algorithm on artificially degraded images. In terms of identification and pose recovery, the heuristic cluster algorithm performed the best. Since the pixels are selected so as to not degrade recognition performance, the hope is that the subset selection methods will be less sensitive to the effects of occlusion.

2.3.3.2 Ohba: Eigen-Windows for Dealing with Occlusion

Realizing the problems of the Eigenvector technique with respect to occlusion, Ohba and Ikeuchi developed a modification of the technique (Ohba and Ikeuchi 1996). Similar to the recent work of Nayar (Hadjidemetriou and Nayar 1998), they use subsets of pixels

instead of the entire image. For all possible pairs of sub-windows in each training image, the Euclidean distance between the two is computed. If that distance is small, one of the windows is discarded, and the process continues. The result of this first stage of training is a group of sub-windows per object. The Eigenvectors for all the sub-windows are then computed and used for matching in the same way the vectors for the entire image are used: each sub-window from a new image is projected into Eigenspace and the closest training sub-window is found. A voting scheme clusters regions of the image that have high similarity with the same sub-window. Peak detection over the resulting votes determines the locations of objects in the scene.

The approach has several advantages. First, perfect segmentation is not required, and therefore, problems associated with occlusion can now be solved. Second, the sliding window allows multiple objects to be present in the scene. Ohba and Ikeuchi successfully applied the technique to a traditional bin picking problem. In this application, they assumed all of the objects were of the same model (metal screws). It may be significant that the screws have a high degree of self-similarity which reduced the size of the required training set.

2.3.3.3 Camps: Dealing with Segmentation Problems

Camps *et al.* have taken a different approach to representing the information projected into Eigenspace (Huang, Camps, and Kanungo 1998) which they refer to as recognition by parts. The focus of this work is to reduce the sensitivity of the Eigenspace technique to the problems associated with segmentation. Training begins by segmenting the objects in an image into regions of pixels with similar intensity. Segmentation is accomplished using the MDL principle.

A set of relations over the segmented *parts* can be defined. These relations define how the union of the parts creates the complete object. The Eigenvector basis set for the parts regions are then determined. Recognition in novel images begins with a similar segmentation and projection of regions into Eigenspace. The object identified must be both close in Eigenspace to a set of similar parts regions in the training set and satisfy the correct relationship between parts.

The technique has been demonstrated on scenes with a variety of objects and varying amounts of occlusion. Because the technique is built upon recognizing pieces of an object, finding more pieces leads to a stronger belief that the entire object is visible. The technique will not lend itself well to recognizing objects with highly textured surfaces as they will be highly problematic for the segmentation algorithm.

2.4 RMR: Fusing Geometry and Appearance

So far, we have discussed the general concepts of three different areas of research in model-based object recognition: geometric, augmented geometric and appearance matching. We believe the algorithms presented in this dissertation serve as a bridge between each of the three sub-areas of model-based vision. The Render-Match-Refine (RMR) approach, as briefly outlined in Chapter 1, is capable of dynamically generating a rendering of an object from an arbitrary view using the underlying object geometry. Thus for any given configuration, a texture mapped prediction captures the power of the appearance matching techniques without having to store a large number of templates. Since the $3D$ geometry of the scene is used in the rendering, explicit reasoning about occlusion is possible.

2.4.1 Effects of Texture

As was the case with appearance matching, the RMR paradigm is capable of representing objects with arbitrarily complex surface markings. Here, we will use a limited number of training views (currently four as discussed in Chapter 3) per object to construct a texture map. This sharply contrasts with the 72 training images required for the appearance-based techniques. The texture will then be mapped onto simple $3D$ geometric structures. Since the prediction is made in a top-down manner, no symbolic feature algorithms are run over the sensor data. Hence, no errors are introduced by either straight line extractors, or segmentation routines.

2.4.2 Effects of Occlusion

A benefit of rendering is the explicit handling of occlusion. When a rendering is generated for a hypothesized configuration, the object parameters are specified in scene coordinates.

Therefore, if one object is closer to the sensor than another, the farther object will appear occluded in the rendering: the geometry is used along with the scene parameters to predict specific occlusions.

2.4.3 Individual Contributions

A variety of RMR recognition techniques (listed in Table 2.1) are now discussed: Fua used rendering and iterative search to build a polygonal mesh from a series of object images (Fua and Leclerc 1994), Hoff used RMR for locating implants in X-ray images (Hoff and Sarojak 1998), Wells used RMR to track a patient's head in a video sequence (Wells, Halle, Kikinis, and Viola 1996) and Stevens used RMR for Automated Target Recognition (Stevens and Beveridge 1997).

2.4.3.1 Fua: Registration without Correspondences

Fua *et al.* have developed the best known implementation of RMR (Fua and Leclerc 1994). Their technique used a set of object images from a variety of camera viewing angles. One of the camera images is selected as a texture map and rendered using a hypothesized $3D$ model to produce a series of images corresponding to the remaining camera viewing angles. An objective function compares each predicted rendering to the observed data. This function contains parameters for the positions and orientations of the various cameras as well as parameters for controlling the shape of the underlying $3D$ object geometry.

Both conjugate gradient methods as well as the Simplex search algorithm (see Section 5.1.1 of Chapter 5 for a complete description of the Simplex search algorithm) have been used to minimize their objective function. Through empirical analysis, they determined the algorithms work best when the object surface was optimized independently from the camera positions. Throughout the search process, the algorithms sequentially switch back and forth between the sets of parameters being adjusted.

The end result is a new $3D$ surface which is the best model re-construction of the object in the scene. While their technique was concerned with building models and not object recognition, the algorithms developed share many commonalities with those presented in this thesis. In fact, their work provided justification for many of our earlier techniques.

2.4.3.2 Hoff: Rendering for Implant Location

Hoff *et al.* have used a variation of the RMR technique for recovering the pose of lower extremity implants (such as artificial knee joints) in X-ray images (Hoff and Sarojak 1998; Sarojak 1998). The process begins with a binary rendering of the pixels which are predicted as object, and those predicted as non-object. An objective function matches the predicted occluding contour of the object to similar features in the X-ray image.

Hoff *et al.* make use of both the Simplex search algorithm as well as simulated annealing to search over the error surface defined by their objective function. They have compared the performance of using the RMR method to that of matching stored templates to imagery. With high performance rendering hardware, they were able to achieve low computation times as well as a huge improvement in recognition quality over just using templates.

2.4.3.3 Wells: Rendering for MAP Estimation

Wells *et al.* have developed a prototype system for using rendering techniques to predict visible object features (Wells, Halle, Kikinis, and Viola 1996). The process begins with a single object aligned to an existing image. The goal is to track the object through a series of video frames. The intended application is computer assisted surgery; the model is a human face and is overlayed onto a video sequence of a person during surgery. The models were obtained prior to the surgery for each individual.

A rendered image of the human face model is used by a point extraction algorithm. These points are then tracked through the video sequence. Wells used graphical rendering for two reasons: to obtain real-time predictions of object geometry and to deal with self-occlusions. The predicted points were fed into a MAP estimation routine (Wells 1993) to gauge the quality of the current alignment of the object to the video imagery. A simple refinement algorithm based on the feedback from the MAP function was quickly able to correct for mis-alignments in object pose due to slight shifts in the patient's head.

2.4.3.4 Stevens: RMR for Automatic Target Recognition

Our earlier work focused on reasoning about occlusion with the aid of information from heterogeneous sensors: information about occlusion from a $3D$ range sensor was propagated

through the model to reason about occlusion in a $2D$ optical image (Stevens and Beveridge 1997; Stevens and Beveridge 1998; Beveridge, Draper, Stevens, Siejko, and Hanson 1997). For that work, we employed a limited subset of the ideas detailed in the following chapters. Graphical rendering was used to make a prediction of a military vehicle's appearance in both range and optical imagery. These predictions were based on rendering the object geometry (see Section 3.4.1 of Chapter 3). The range sensor was then used to adapt these predictions based on information about occlusion.

The Simplex search algorithm was used in conjunction with a fairly complex error function to refine the hypothesized location of a given object. The error function and search algorithm evolved over the years; early implementations focused on recognizing simple unoccluded objects (Stevens and Beveridge 1997). Recently, the goal has been to solve more realistic target recognition problems involving substantial amounts of occlusion (Stevens and Beveridge 1998). While different error functions, rendering methods and search algorithms were used, the underlying iterative method is quite similar.

Chapter 3

Render: Predicting Scenes

"From a drop of water a logician could predict an Atlantic."

- Sir Arthur Conan Doyle

The main goal of scene prediction is to generate an image of a virtual world which can be directly compared to an image from a real sensor. How accurate the prediction must be in order for the two images to be comparable is dependent upon the specific application domain and the comparison metric used. Intuitively, the closer the prediction mirrors reality, the higher the chance the RMR (Render-Match-Refine) algorithm will be successful. Unfortunately, generating a perfect prediction is an impossible task. Information such as the background scene context, lighting, and complex camera distortions are difficult to model and often cannot be inferred from a static sensor image. Therefore, the role of prediction is to produce an image, at the resolution of the real sensor, that provides information exploitable by the subsequent stages of the RMR method (matching and refinement).

To meet this requirement a well understood practice in computer graphics known as *texture mapping* is used. Texture mapping overlays previously stored images, called *textures*, onto the $3D$ face of an object as that object is projected onto the image plane (Heckbert 1989). Texture mapping was chosen because it provides a strong link between the appearance-based and geometry-based techniques (see Chapter 2 for details). In appearance-based matching, a large number of stored object views (a minimum of 72 training images) are compared with novel sensor images (Nayar, Nene, and Murase 1996). Since we are using the underlying object geometry during texture mapping, far fewer training images are required to construct the model representation (in the case of this dissertation only four were used). The $3D$ object geometry allows us to generate a dynamic prediction

of appearance for each object at any position and orientation in the image; not just those stored in a $2D$ collection of training images.

Furthermore, the textures augment the geometric representation by easily representing complex surface markings. Previously, most model-based object recognition representations were based on the object boundary and not on internal surface variation. Texture mapping allows the underlying surface geometry to be added to the appearance representation while at the same time allowing the addition of surface appearance to the underlying geometric representation.

The field of computer graphics has numerous other techniques which could be employed during the rendering task. A few of the more relevant are scene illumination, object reflection, shadows, and models for complex sensor distortions. Under certain conditions, augmenting the prediction mechanism with any one of these techniques could greatly improve the quality of the rendered image (making it more similar to the sensor image); thereby improving the accuracy of the subsequent matching and refinement tasks. If the assumptions underlying those techniques are erroneous, overly specific, or do not actually fit reality, quality of the rendered image will degrade; thereby reducing the reliability of matching and refinement.

Deciding what or what not to incorporate into a rendered prediction should be based on the intended application and decided upon by trading off generality and specificity: incorporating overly specific information into a rendering will reduce the generality of the approach across a range of problem domains. A general rule-of-thumb has been discussed in (Bowyer and Phillips 1998):

> "Unfortunately, conceptual elegance and sophistication of the mathematics [underlying the algorithm] are not necessarily correlated in a positive way with performance of an algorithm in [an] application. If the use of more sophisticated mathematics requires specific assumptions about the application,and these assumptions are not satisfied by the application, performance could even degrade."

The goal, as it relates to rendering in RMR, is to continue to model and incorporate into the prediction characteristics of a scene until recognition performance begins to degrade.

(see Figure 3.1, reprinted with permission from (Bowyer and Phillips 1998)). To ensure generality across a variety of domains, the degradation in performance should be measured on a variety of different scenes across several domains.

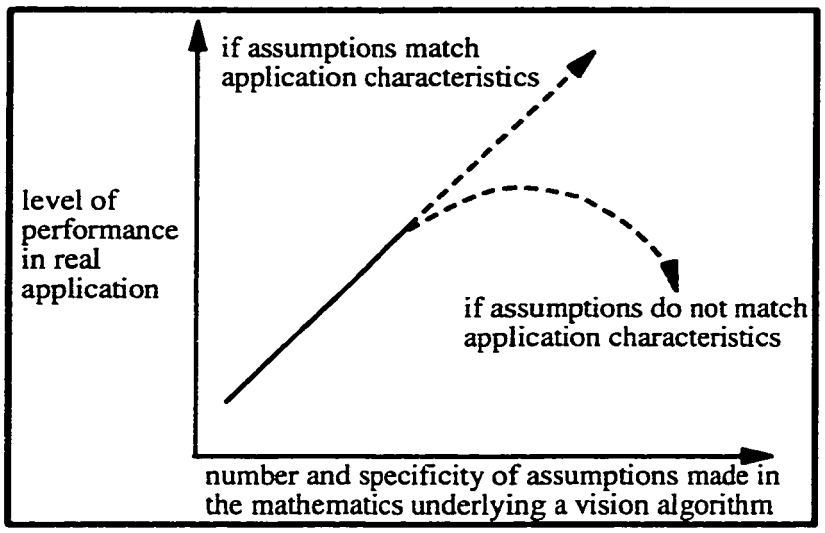

Figure 3.1: Divergence of assumptions from reality.

Every research contribution has limitations. Further work in the area of RMR could examine the effect of adding any number of standard graphics techniques to the prediction phase. Such work would need to determine whether those augmentations increased the robustness of the entire object recognition algorithm for a given application domain. The work presented here may be viewed as establishing a baseline of capabilities which will be elaborated upon as appropriate in the future. While the true contributions of this thesis lie within the field of object recognition, a strong component of these contributions is the use of computer graphics as a tool for object recognition.

This chapter should be viewed as a juxtaposition of standard computer graphics techniques and the thesis specific variants. The remainder of this chapter is divided into four sections:

1. Section 3.1 presents the entire dataset, including images and models, used for evaluation throughout the dissertation (thesis specific).

2. Section 3.2 formally defines the parameters used by the prediction mechanism to project an object from model coordinates onto the image plane (mostly standard

graphics techniques except for the thesis specific pseudo-polar pose representation). The parameter representation is based on the standard computer graphics rendering pipeline.

3. Section 3.3 describes how the model is represented in terms of texture maps and object geometry.

4. Section 3.4 describes texture mapping in more detail as well as additional rendering techniques used to measure attributes about a given sensor image (thesis specific).

3.1 Overview of the RMR Dataset

Before describing how to predict a scene, it helps to better define the types of scenes being interpreted. The datasets collected for evaluation of our RMR implementation are broken down into two distinct groups: blocks-world and products-world. Within each group 40 images were collected which yields 80 images for the entire collection. To the best of my knowledge, a standard model-based recognition test domain does not exist. Several test sets are available for only imagery (Nene, Nayar, and Murase 1996) and only models (Flynn and Campbell 1998), but not for both.

The datasets are shown in Figures 3.2 and 3.3. Each image was taken with a Kodak DC25 digital camera (Kodak 1999) and is stored in the *24bit* color TIFF file format. A VRML file is also present for each scene containing the ground truth location of the objects as well as the camera parameters (fully described later in this chapter). The stored models for each object are shown in Figure 3.4. Table 3.1 lists the specific images in which the models appear.

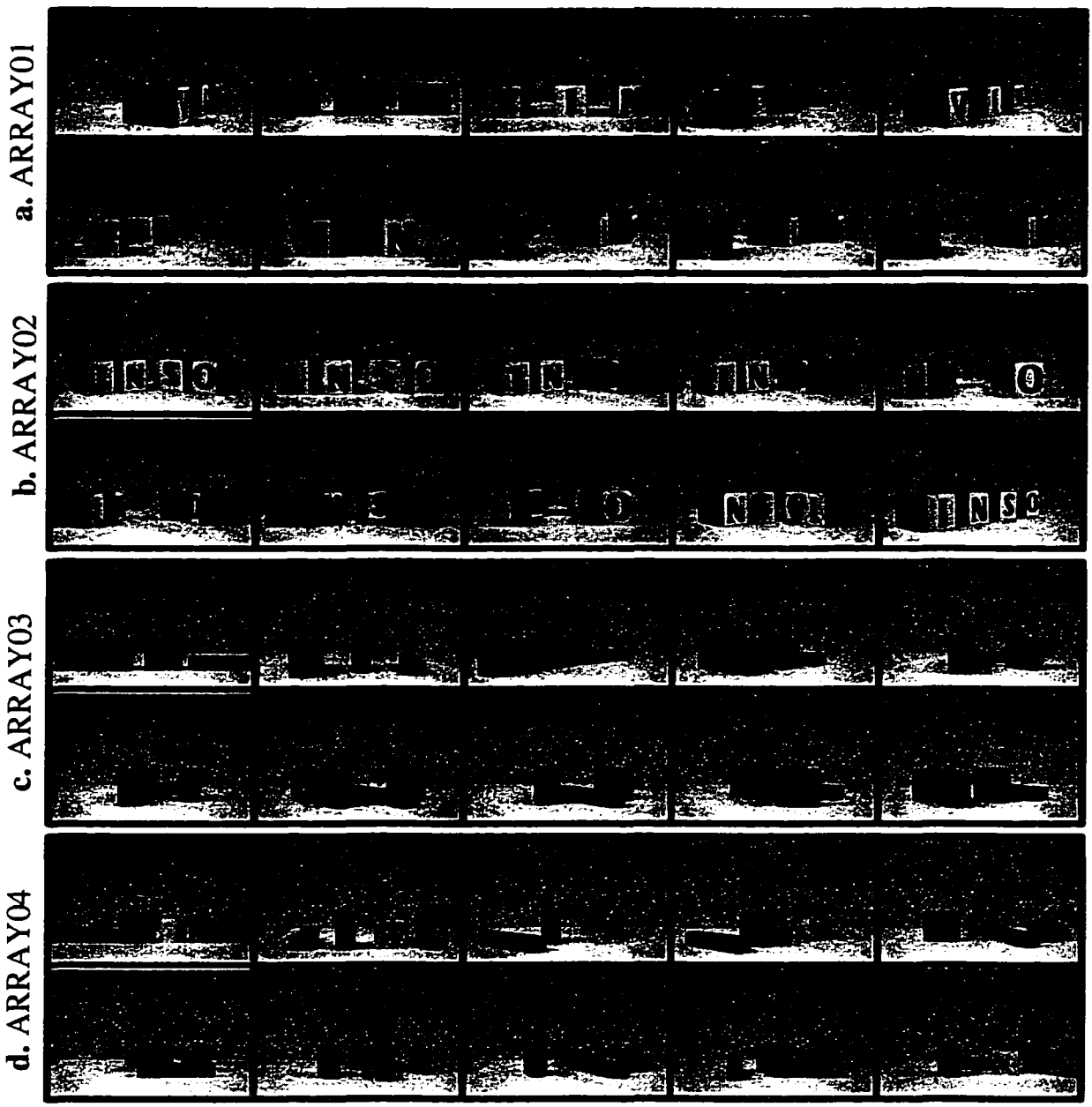

Figure 3.2: The blocks-world dataset. The dataset is organized into four different arrays each containing 10 scenes. There are a total of 10 different objects exhibiting five different surface colors: red, green, blue, yellow and orange.

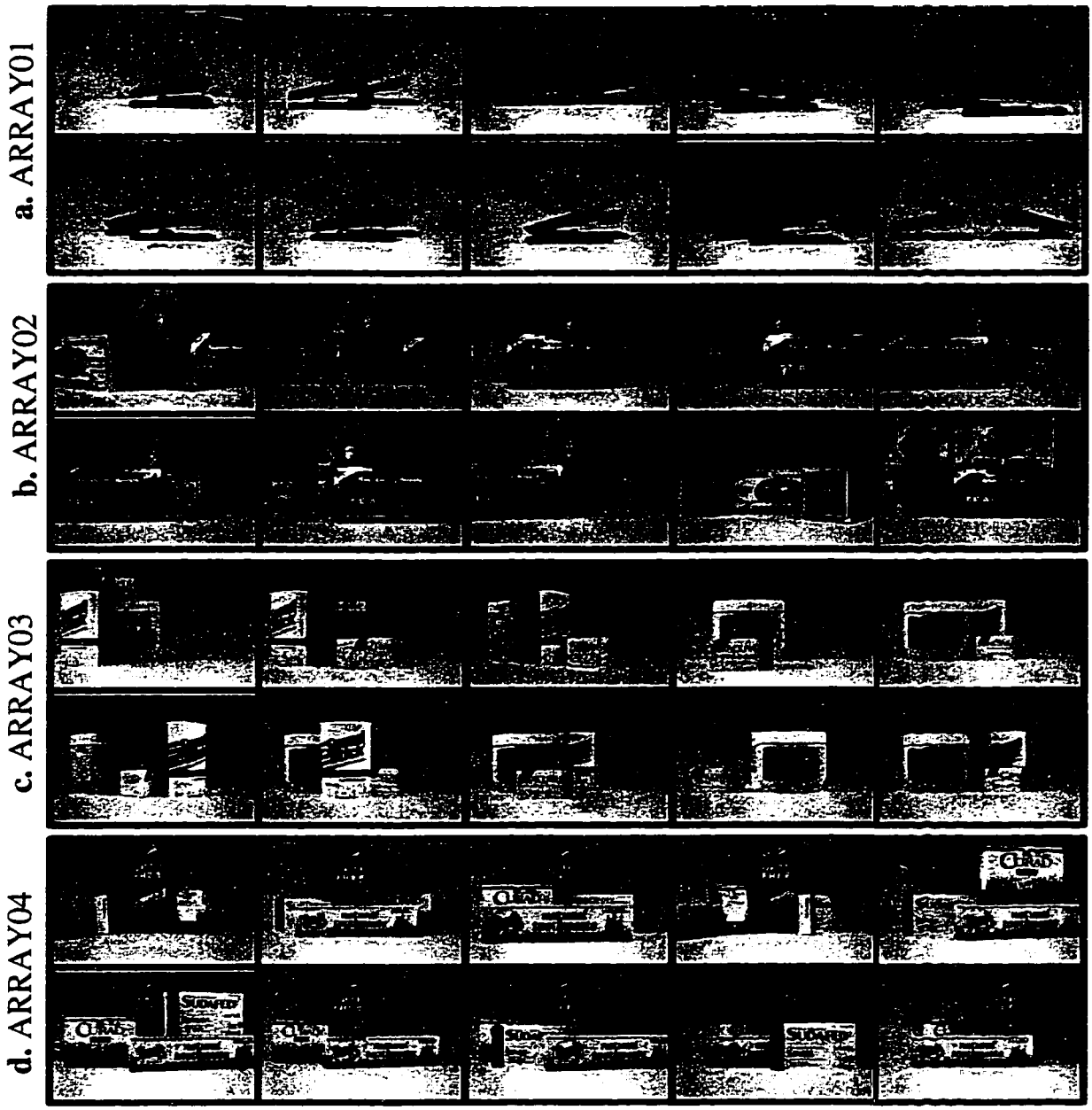

Figure 3.3: The products-world dataset. The dataset is organized into four different arrays each containing 10 scenes. There are a total of 12 different objects.

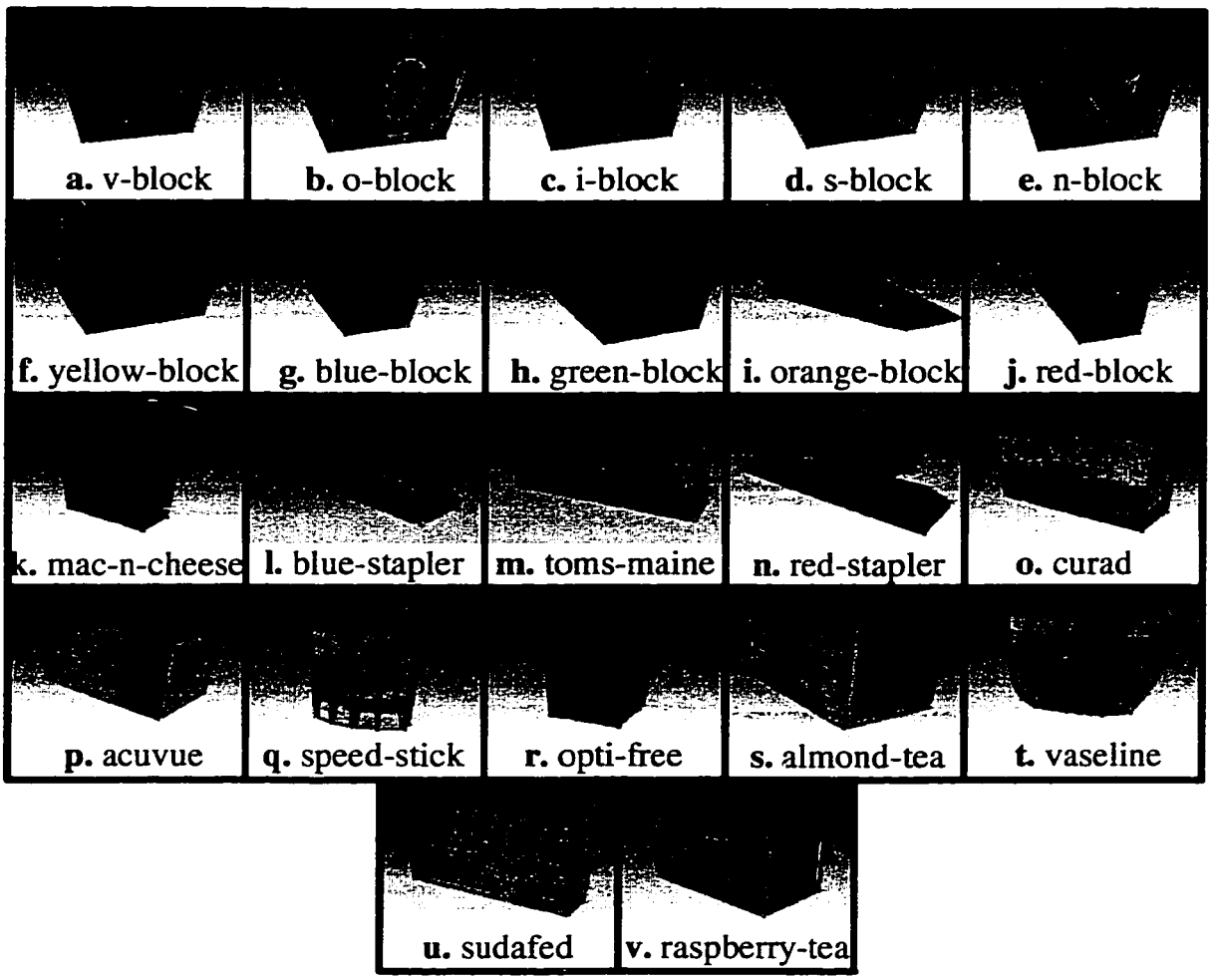

Figure 3.4: The various models in the database.

	Block Arrays			
	1	2	3	4
v-block	√			
i-block	√	√		
n-block	√	√		
s-block		√		
o-block		√		
red-block			√	
green-block			√	√
blue-block			√	√
yellow-block				√
orange-block				√

	Product Arrays			
	1	2	3	4
blue-stapler	√			
red-stapler	√			
mac-n-cheese		√		
almond-tea		√		
raspberry-tea		√		
acuvue			√	
vaseline			√	
speed-stick			√	
toms-of-maine				√
opti-free				√
sudafed				√
curad				√

Table 3.1: The models present in the database and the image array in which they appear.

3.2 Formally Defining the Scene Configuration

The position of an object in the scene is defined in terms of a set of pose parameters $\mathcal{F}$:

$$\mathcal{F}_{object} \tag{3.1}$$

where $\mathcal{F}_{object}$ is defined to map a point from model coordinates, to scene coordinates, then to camera coordinates and finally onto the image plane (Figure 3.5). The mapping functions for several objects comprise the scene configuration $\mathcal{S}$:

$$\mathcal{S} = \{\mathcal{F}_{object1}, \mathcal{F}_{object2}, \ldots, \mathcal{F}_{objectN}\} \tag{3.2}$$

The goal of model-based scene interpretation is to recover $\mathcal{F}$ for each object. Traditionally, the problem is solved by first determining $\mathcal{F}_{object1}$ and then $\mathcal{F}_{object2}$ and so on. When only dealing with a single object, the mapping of the object to the scene is typically neglected. Therefore, the problem of recovering the object pose with respect to the camera is equivalent to recovering the camera pose with respect to the object.

For multi-object scene interpretation, a discrepancy between single object recognition exists: with more than one object there must be both the concept of a scene coordinate system and the concept of a camera coordinate system. Furthermore, solving for the camera pose is not equivalent to solving for the object pose. Single object recognition algorithms also assume a simple projection model for transforming a $3D$ point onto the image plane. Given the $3D$ point $\{P_x, P_y, P_z\}$, the image point, $\{p_x, p_y\}$, is defined as:

$$\begin{vmatrix} p_x \\ p_y \end{vmatrix} = \begin{vmatrix} f\frac{P_x}{P_z} + t_x \\ f\frac{P_y}{P_z} + t_y \end{vmatrix} \tag{3.3}$$

where f is the focal length of the sensor, and (t_x, t_y) are translation parameters to correct for the situation where the optical axis is not the image center. Obviously, an infinite number of $3D$ points can project to the same pixel. Any basis for reasoning about occlusion is lost since the object point closest to the sensor cannot be determined.

A more complex projection model is needed which allows the projection of $3D$ points onto the image while still providing some mechanism for determining the closest point to the sensor. The more complex projection model is found by switching to a perspective

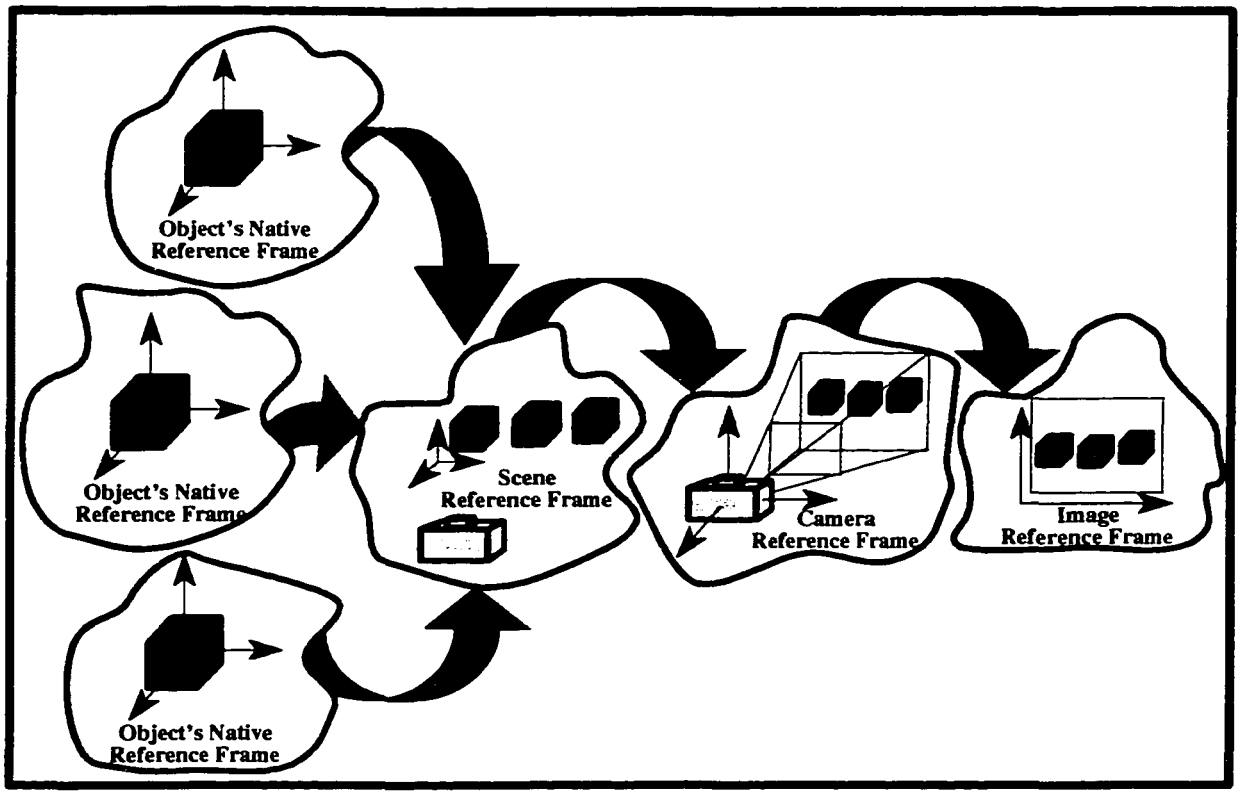

Figure 3.5: Computer Vision is typically concerned with four reference frames: model, scene, camera and image.

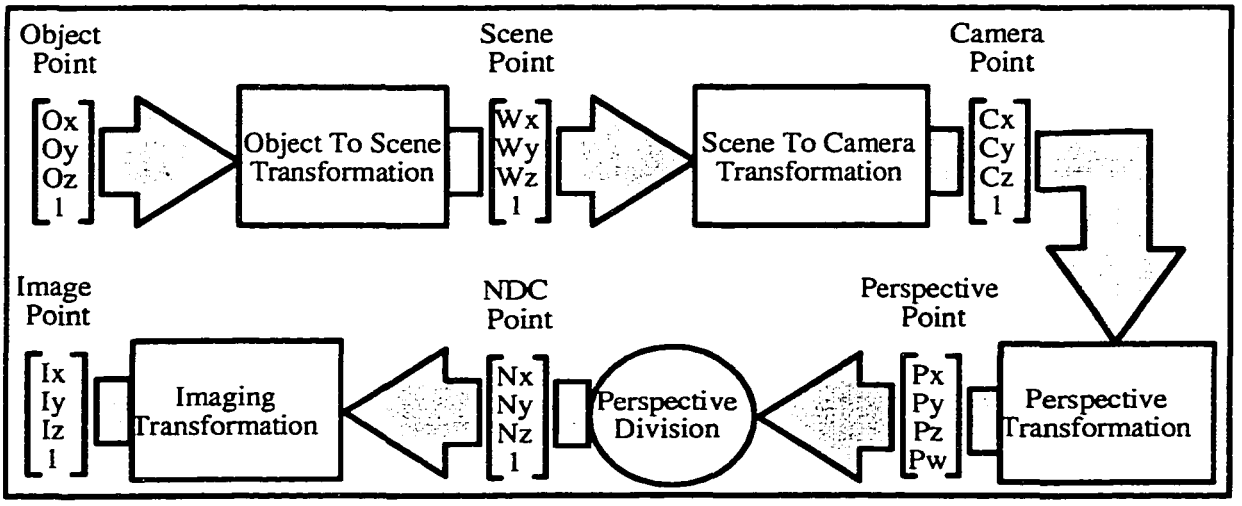

Figure 3.6: The rendering pipeline.

view volume representation (Foley and vanDam 1982). The object pose parameters are first converted into a matrix. This matrix, referred to as the *fundamental matrix*, is used to transform a point from one coordinate system to another. Figure 3.6 shows the various transformations used in computer graphics to render a scene. Notice that a point in the $3D$ world is represented as a homogeneous $4D$ vector. Homogeneous coordinates are convenient for performing both rotation and translation using a single matrix multiplication (Foley and vanDam 1982).

A $4D$ point in the model flows through the pipeline to produce a $4D$ point on the image plane. The first two coordinates of the image point represent the x and y pixel coordinates of the model point. The third coordinate is a *pseudo-depth* value. This value allows the sorting of points that project to the same pixel. The value is used in a form of hidden surface removal known as *z-buffering* (Foley and vanDam 1982). The z-buffer will correctly handle occlusion interactions between objects: only the closest object at that pixel appears in the predicted image. The fourth coordinate is used to handle the perspective view volume. Subsequent sections detail each phase of the pipeline.

3.2.1 Mapping an Object Point to a Scene Point

Given any $3D$ point $\{O_x, O_y, O_z\}$ in object coordinates, a matrix transformation is applied to map the point into the scene $\{W_x, W_y, W_z\}$. This transformation matrix defines the object to scene mapping M_{object}^{scene}:

$$W = M_{object}^{scene} \cdot O \tag{3.4}$$

The parameters used to construct M_{object}^{scene} are referred to as the object pose parameters. Typically pose space is defined in terms of a six degrees of freedom (DOF) transformation (three rotation and three translation parameters):

$$\mathcal{F}_{object} = \{\theta, \phi, \gamma, T_x, T_y, T_z\} \tag{3.5}$$

where θ is a rotation around the model's own X axis, ϕ a rotation about the Y axis, and γ around the Z axis. The other three parameters $\{T_x, T_y, T_z\}$ represent the translation of the model into the scene.

An added constraint on object pose exists for all of the scenes used in this dissertation (Figures 3.2 and 3.3): the objects must lie on a common ground plane or be stacked directly on top of one another. Using a variant of the ground-plane-constraint or GPC (Sullivan 1992; Linneanmaa, Harwood, and Davis 1988), the DOF of the problem is reduced. Using the GPC, the objects can translate in three dimensions as well as rotate about the Y axis:

$$\mathcal{F}_{object} = \{\phi, T_x, T_y, T_z\} \tag{3.6}$$

3.2.1.1 The Thesis Specific RMR Pose Parameterization

Up until now, the pose encoding has been discussed in general terms not specific to this dissertation. However, a problem arises when using the general pose encoding directly for RMR: the translation components appear to interact when the object is projected into the image. Figure 3.7 shows two images. The block outline is for a cube given two different sets of pose parameters. For the block in the image on the right, only a translation in the Z dimension has been made for the block in the left image. However, in image space there is a perceived translation in both the X, Y and Z dimensions.

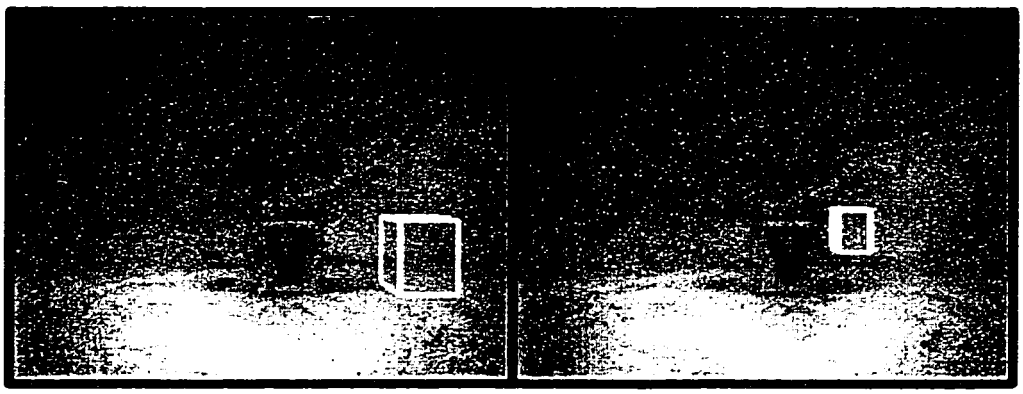

Figure 3.7: The perceived interaction between translation parameters.

This effect is due to perspective: as objects are moved away from the camera, they appear to move towards the vanishing point. Early experiments with this parameterization showed negative side effects during recognition. The search algorithms (as discussed in Chapter 5) had difficultly moving objects away from the camera since a translation in the Z dimension typically induced a perceived translation in the X dimension. To overcome this problem, we developed a slightly different object pose encoding for use in RMR.

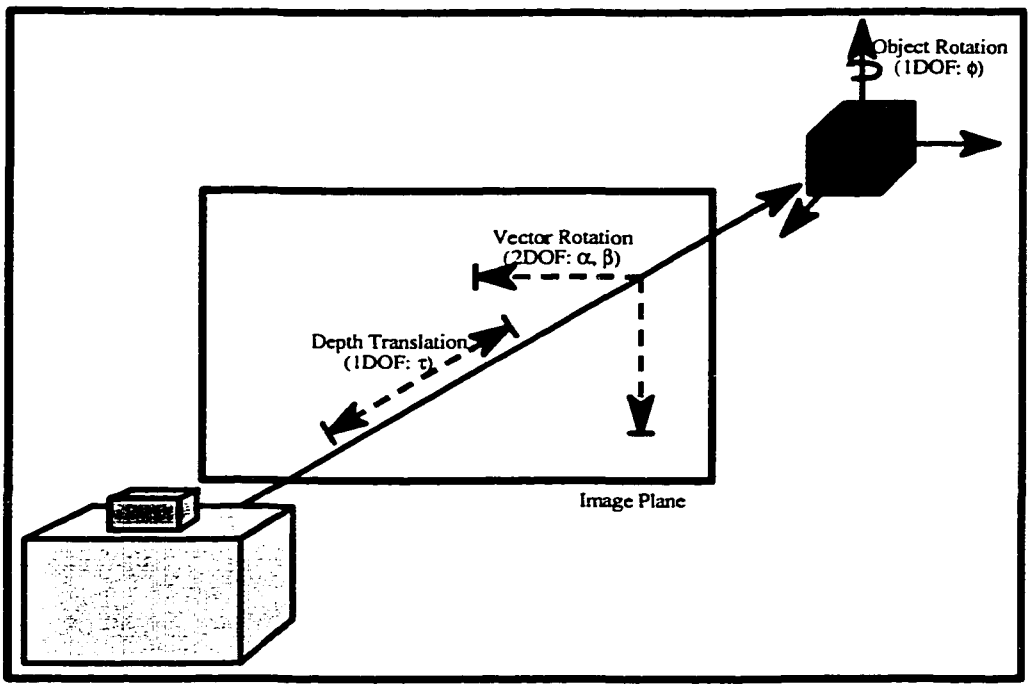

Figure 3.8: The vector based pose parameterization. Translations away from the sensor do not produce the false impression that the object is translating towards the vanishing point.

New translation components are defined in terms of the vector formed between the center of the object and the position of the camera lens. The new parameterization allows shifting of this vector in two dimensions, as well as movement along the vector (See Figure 3.8). The objects can move away from the sensor without the perceived translation towards the vanishing point. The new 4 DOF pose parameterization, referred to as *pseudo-polar*, is defined as:

$$\mathcal{F}_{object} = \{\phi, \alpha, \beta, \tau\} \tag{3.7}$$

where α and β define the infinite vector along which the object center will lie, and τ fixes the object centroid at a specific point on the vector. The new representation is converted to the original parameterization:

$$\begin{aligned} T_x &= -\sin(\alpha)\cos(\beta)\tau \\ T_y &= \sin(\beta)\tau \\ T_z &= -\cos(\alpha)\cos(\beta)\tau \end{aligned} \tag{3.8}$$

Using the form $W = M_{object}^{scene} \cdot O$, the entire object to scene transformation is:

$$\begin{vmatrix} W_x \\ W_y \\ W_z \\ 1 \end{vmatrix} = \begin{vmatrix} \cos\phi & 0 & \sin\phi & -\sin(\alpha)\cos(\beta)\tau \\ 0 & 1 & 0 & \sin(\beta)\tau \\ \cos\phi & 0 & \sin\phi & -\cos(\alpha)\cos(\beta)\tau \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} O_x \\ O_y \\ O_z \\ 1 \end{vmatrix} \tag{3.9}$$

where the upper left $3x3$ sub-matrix rotates the object around the Y axis before it is translated into the scene. Each object in the scene contains its own M_{object}^{scene} transformation which is formed directly from the corresponding parameters in $\mathcal{F}_{object}$.

3.2.1.2 Determining Ground Truth Parameters for the RMR Dataset

For performance evaluation of our RMR implementation, the absolute placement of the objects in the scene relative to the camera must be known. For each object in each scene of our dataset, several ground truth measurements are taken. The first step is to define an arbitrary point in the scene as the origin of the world. Next, a coordinate system is defined about that point. The rotation and translation of each object with respect to that coordinate system are recorded. These four values form the true pose parameters ($\mathcal{F}_{object}^{true}$).

3.2.2 Mapping a Scene Point to a Camera Point

The next phase of the rendering pipeline transforms the scene point $\{W_x, W_y, W_z\}$ into the coordinate system of the camera $\{C_x, C_y, C_z\}$:

$$O = M_{scene}^{camera} \cdot W \tag{3.10}$$

To compute M_{scene}^{camera} knowledge about the placement of the camera in the world is required. This knowledge may come from landmark navigation (Beveridge, Balasubrarian, and Whitley 1999) or global positioning systems (Lichten 1988). For this dissertation, the X, Y, and Z position of the camera lens with respect to the previously defined center of the world are measured by hand as a part of developing the ground truth for this dataset. These values are used to compute the scene to camera translation. For all of the images used, the z axis of the world coordinate system is aligned with the z axis of the camera.

Therefore, the transformation into camera coordinates is a translation:

$$\begin{vmatrix} C_x \\ C_y \\ C_z \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.056 \\ 0 & 0 & 1 & -0.4 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} W_x \\ W_y \\ W_z \\ 1 \end{vmatrix} \tag{3.11}$$

3.2.3 Mapping the Camera Point to an Image Point

Many different sensor models are available to determine where on the image plane a $3D$ point in front of the camera will project. The standard pin-hole camera representation (Nalwa 1993) for computer graphics is used to model the Kodak camera used in this dissertation. The pin-hole model may seem weak in light of the more advanced sensor models (Healey and Kondepudy 1992; Balasubrarian 1998; Boult and Wolberg 1992). However, the model was chosen for two reasons:

1. While the pin-hole model is only an approximation to the workings of an actual CCD camera, we found it to be highly reliable for generating images similar to those obtained by our sensor. In over 80 images used to validate the recognition strategy, none showed more than a one or two pixel mis-registration.

2. The pin-hole model can be used efficiently. During recognition, the prediction mechanism may be called thousands of times. Since the more advanced sensor models rely on ray tracing (Balasubrarian 1998), rendering images for RMR with these models is impractical. In addition, many application programming interfaces (API), such as OpenGL (Paul 1998), only support simple pin-hole camera models.

The current recognition paradigm, as presented, is highly modular. If there becomes a need for a more complex sensor model, a more accurate representation could easily be substituted.

The graphical pipeline of Figure 3.6 uses three steps to project a point in camera coordinates onto an image plane. The first step maps the camera point into a cube of fixed dimensions (see Figure 3.9). While a common practice in graphics, the mapping may not seem intuitive. The process is visualized as warping a set of perspective rays leaving the camera into a set of rays orthogonal to the image plane. The new z value of a point after being transformed by this matrix represents the distance of the object from the camera along

the orthogonal ray. Once the transformation is performed, hidden surface removal is trivial since the new pseudo-depth values show which point is closer to the image plane (Foley and vanDam 1982).

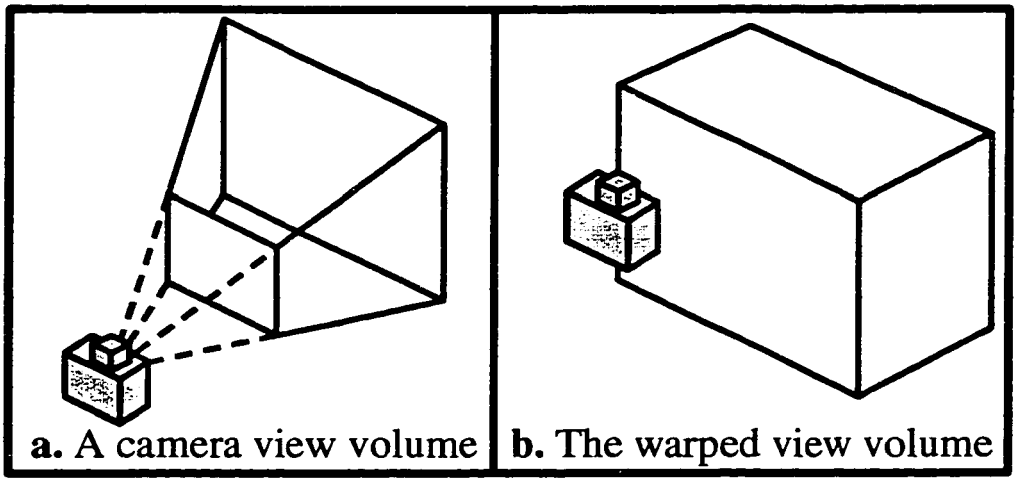

Figure 3.9: Projecting a camera point first involves warping the view volume from a pyramid into cube.

This transformation is accomplished with the perspective matrix. Using the form $P = M_{camera}^{perspective} C$ the entire camera to perspective transformation is:

$$\begin{vmatrix} P_x \\ P_y \\ P_z \\ P_w \end{vmatrix} = \begin{vmatrix} \frac{\cot(fov/2)}{w/h} & 0 & 0 & 0 \\ 0 & \cot(fov/2) & 0 & 0 \\ 0 & 0 & \frac{Far+Near}{Near-Far} & \frac{2*Far*Near}{Near-Far} \\ 0 & 0 & -1 & 0 \end{vmatrix} \cdot \begin{vmatrix} C_x \\ C_y \\ C_z \\ 1 \end{vmatrix} \tag{3.12}$$

where fov represents the field of view of the sensor, w and h are the width and height of the generated image, $Near$ represents the distance from the camera to the $Near$ clipping plane, Far the distance to the far clipping plane, and $\{P_x, P_y, P_z, P_w\}$ is the perspective point of the camera point $\{C_x, C_y, C_z\}$. A CCD camera does not actually have clipping planes, since all light traveling towards the sensor is absorbed. However, in computer graphics the $Near$ and Far planes are introduced to characterize the accuracy of hidden surface removal. As points in camera coordinates are transformed by the projection matrix, their relative distance to the sensor will be preserved to within the precision of the depth buffer and the ratio of Far to $Near$. Also note that the precision is non-linear with more accuracy for depth values farther from the camera.

After perspective projection, the final homogeneous component is no longer 1. Division is used to find the normalized device coordinate (NDC) of the point:

$$\begin{vmatrix} N_x \\ N_y \\ N_z \\ 1 \end{vmatrix} = \frac{1}{P_w} \cdot \begin{vmatrix} P_x \\ P_y \\ P_z \\ P_w \end{vmatrix} \tag{3.13}$$

A final step is necessary to project the NDC point onto the image plane. This step utilizes a viewport projection matrix and maps the points from the NDC cube into image coordinates. Using the form $I = M^{ndc}_{viewport} \cdot N$ the NDC to viewport transformation is:

$$\begin{vmatrix} I_x \\ I_y \\ I_z \\ 1 \end{vmatrix} = \begin{vmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} N_x \\ N_y \\ N_z \\ 1 \end{vmatrix} \tag{3.14}$$

where I_z is the pseudo-depth and is used for z-buffer hidden surface removal. The projection parameters of the Kodak CCD camera are taken directly from the technical specification (Kodak 1999). The only parameter not given is the far clipping plane, which is assigned a value of $1meter$ (which is far enough away from the camera to guarantee none of the objects are clipped). Table 3.2 shows the camera parameters recorded.

Parameter	Description	Kodak Technical Specification	Camera Calibration
w	Image Width	240	N/A
h	Image Height	180	N/A
$Near$	Near Clipping	$0.01m$	N/A
Far	Far Clipping	N/A	$1m$
fov	Field of View	40°	N/A

Table 3.2: The camera calibration parameters used for the Kodak DC25 digital camera.

3.3 The Model Representation

In the purest sense, a geometric model is a collection of geometric primitives which are grouped to form a solid shape having a closed volume. The work in this dissertation focuses on rigid objects stored in a representation known as the boundary representation or BREP (Mantyla 1990). The BREP captures the key elements easily rendered by the standard graphics API OpenGL: planar polygons, line segments, and Euclidean points. More specifically, the representation deals with *objects*, *faces*, *edges*, and *vertices* (See Figure 3.10).

Vertex: A vertex is a geometric point in Euclidean space (E^3).

Edge: An edge is the connection of two vertices which form a line. The line is continuous (i.e., there exist an infinite number of co-linear points between the end points).

Face: A face is a connected set of planar edges which create a closed area. Any point lying on the plane and *inside* the face cannot be connected to any co-planar *outside* point without crossing the edge boundary.

Object: An object is a connected set of planar faces which form a closed volume. None of the points lying *inside* the volume can be connected to any of those lying *outside* without first crossing a face boundary.

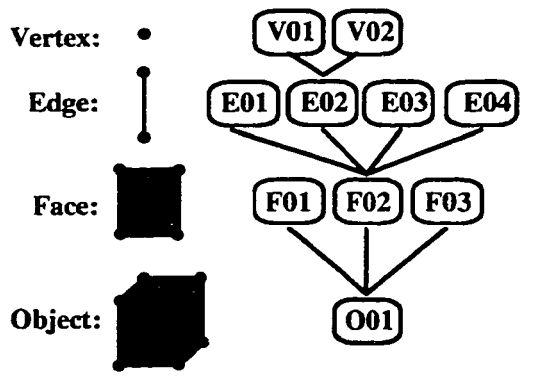

Figure 3.10: The BREP

Adding appearance information to a geometric model is accomplished with the use of texture maps. Texture mapping drapes a previously stored $2D$ image, referred to as a *texture map*, over a $3D$ object face at an arbitrary position and orientation with respect to the camera. Texture mapping provides many advantages for RMR:

1. Texture maps are readily supported by most rendering APIs. This allows efficient image generation of texture mapped objects. Efficiency is further enhanced when specialized hardware is used.

2. The underlying $3D$ geometry does not need to be complex. Since the variation within an object face is handled by the texture map, the object geometry only needs to represent the object boundary. For instance, many of the models in Figure 3.4 are simple cubes, and there is no structural difference between the model used for the *curad* box and that of the *v-block* model.

3. The number of training images required to generate an object texture map is quite small (one for each object face). This number can be compared to standard appearance matching techniques (Nayar, Nene, and Murase 1996; Huang, Camps, and Kanungo 1998), which rely on Eigenvector analysis of up to 72 training images per object (not even accounting for changes in object scale).

To generate a texture map for a new object, a training image for each face is needed (see Figure 3.11). Each of these images is a high-resolution view of the face, with the face being nearly parallel to the image plane. The resolution is needed so that a wide range of predictions can be made at a variety of scales. The parallel constraint is used to remove distortions due to perspective warping.

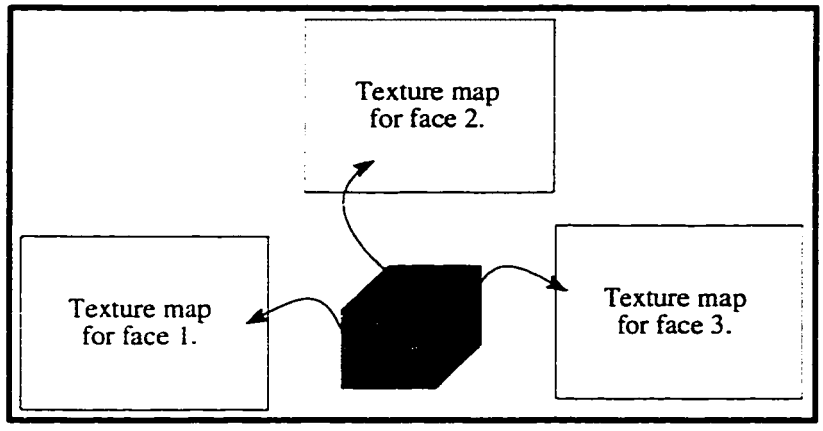

Figure 3.11: A Texture map for each face on an object.

These images should be taken under similar illumination conditions to what is expected during recognition. For instance, a training image taken with a red colored illuminant cannot be used for rendering the same object under a blue illuminant. For best results, each training image should also be well lit to avoid shadows and incorrect estimates of the underlying surface color.

For the simple objects in this thesis, four training images are required: left, right, front and back. Due to a ground plane constraint, training images for the top and bottom of the object are not needed. Figure 3.12 shows the layout of the camera for each of the different training images. When the training images are taken, the object position and orientation relative to the camera is recorded. Using this alignment information, the pixels pertaining to the object are *cut out* and placed into the new texture map. Knowledge about object position is used to project each vertex into the texture map. The texture coordinates for each vertex on each face of the object is recorded and stored within the BREP.

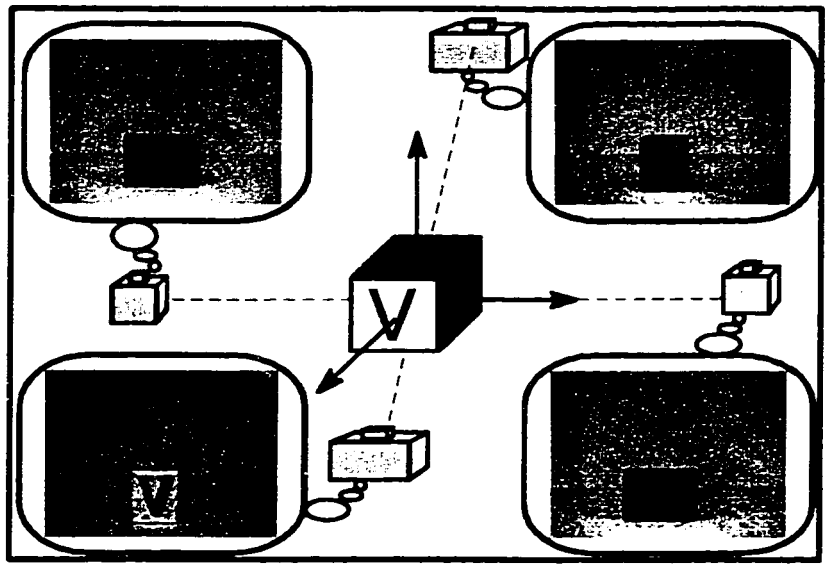

Figure 3.12: Four cameras viewing the object. Each of the four images is used to generate a texture map for the object.

3.4 Rendering Objects Using the Scene Configuration

So far, we have addressed the problems of 1) how to transform an object from its own reference frame onto an image plane and 2) how to represent an object in terms of both geometry and appearance. The final step is to present a method for using this information to predict the scene before the camera. Two types of predictions are generated: one based on object geometry and one based on object appearance.

3.4.1 Rendering Geometry

Many different algorithms exist for rendering geometry. The algorithm implemented for this dissertation is based upon (Stevens 1995) and begins with the assignment of a unique color to each object feature. Using these color labels and a set of pose parameters $\mathcal{F}_{object}$, the object is rendered. The result is an image where the pixel color represents an index into the $3D$ model. Using the index, the model feature inducing each pixel is quickly determined.

To form such an index, each model in the database is assigned a consecutive number beginning with one (zero is reserved for non-object pixels). This number is stored in the red band of the RGB color triple for each feature. Currently, that only allows for 254 different models assuming 8 bits per color channel. The next step assigns each model feature a

unique number. This number is stored in the remaining two color bands (GB). A lookup function takes the 8-bit model indicator and the 16-bit GB color and returns both the correct model and the corresponding feature. Figure 3.13 shows a sample camera image as well as a geometry rendering of the object faces.

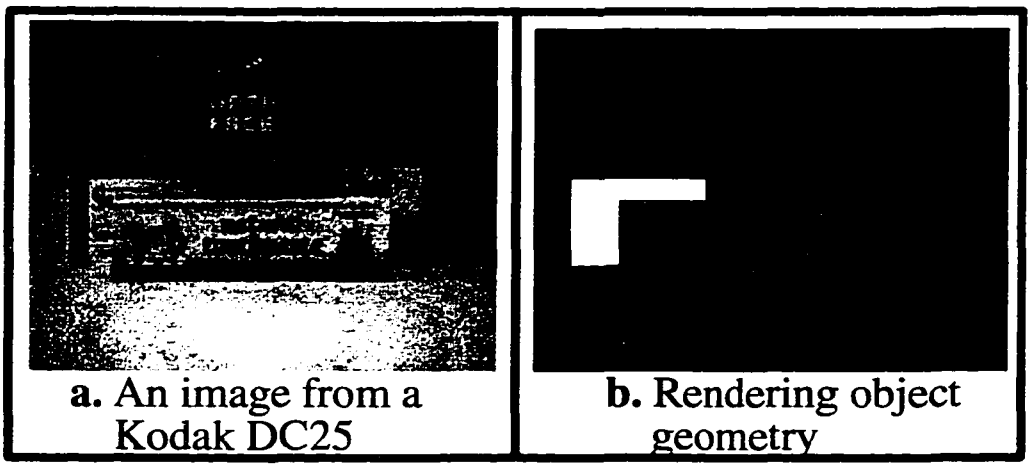

Figure 3.13: An example of rendering for geometry.

Using the current representation, there is a limit of 254 models and 65534 features for RGB color. The current representation could shift bits away from the features and into the model (thus allowing more models to be given indexes) should the need arise. Also, many rendering systems support the use of $RGBA$ color, where A represents an alpha channel used for transparency computations. These extra 8 alpha bits could be incorporated into the labeling scheme.

The main benefit of geometry rendering is the ability to create label images. A label image is an image at the resolution of the actual sensor viewing the real scene where each pixel represents the object or object feature closest to the camera. Label images are used to compute many important properties of a given scene (such as percent object occlusion, and visible object area). These properties are used as independent variables in the later analysis of the RMR approach.

To create a label image, a set of geometry renderings are generated for each new scene (see the left image of Figure 3.14). The first image contains all the objects rendered with their ideal pose parameters (see the second image in Figure 3.14): $\mathcal{F}^{true}_{v-block}$, $\mathcal{F}^{true}_{i-block}$, and

$\mathcal{F}^{true}_{n-block}$. The next three label images are rendered with each object in isolation. For each label image, a set $\mathcal{Q}$ is defined consisting of the pixels labeled object:

$$\mathcal{Q}^{true}_{object} = \left\{ \forall p \in \mathcal{F}^{true}_{object} \right\} \tag{3.15}$$

where p is a pixel in the image given the label for the associated *object* and *true* pose parameters. The cardinality of this set is the number of pixels labeled *object*. Different scene configurations produce different pixels in $\mathcal{Q}$. The pixels in any two sets can be compared. For instance, the cardinality of the set formed by intersecting $\mathcal{Q}^{true}_{object}$ and $\mathcal{Q}_{object}$ provides insight into how many face pixels are correctly identified in the image. A measure for comparing two scene configurations is:

$$D = \frac{\sum_{j \in objects} \left| \mathcal{Q}^{true}_{j} \cap \mathcal{Q}_{j} \right|}{\sum_{j \in objects} \left| \mathcal{Q}^{true}_{j} \cup \mathcal{Q}_{j} \right|} \tag{3.16}$$

The value of D should be 0 when the scene configuration shares no overlapping face features, and 1 when all face pixels completely align. The value of $1 - D$ will be used heavily in subsequent chapters as a means for comparing arbitrary configurations to the ground truth.

Figure 3.14: Generating a set of ground truth label images.

3.4.2 Rendering Appearance

Appearance rendering is rendering with texture maps. Figure 3.15a shows the same image as in Figure 3.13a, only now it is shown beside an appearance prediction. The naive approach to texture mapping begins by projecting the model points into camera coordinates and onto the image plane. If we retain the linking between the original model point and the corresponding texture point, we have two image plane points: the first represents the model point in the image, the second the point in the texture map.

To color the image pixels underlying the object face, one should interpolate between the projected vertices while simultaneously interpolating between the texture points (Heckbert

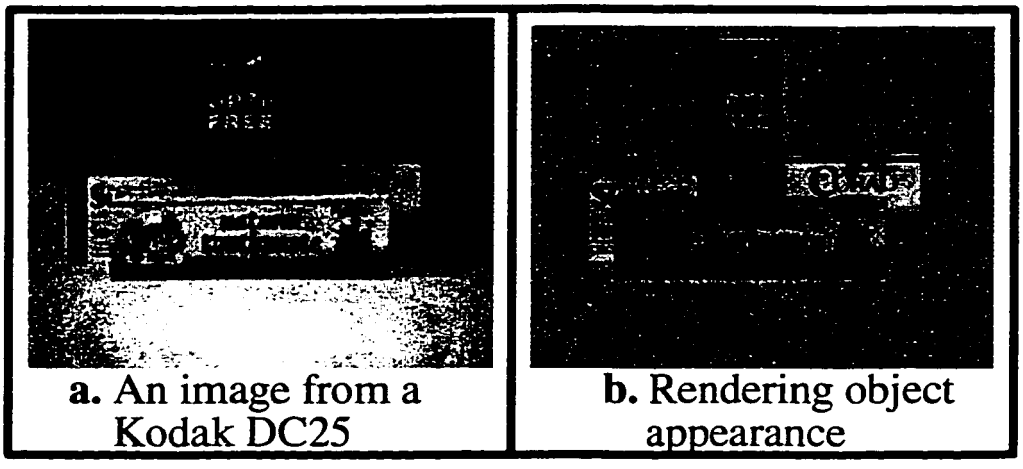

Figure 3.15: An example of rendering for appearance.

1989). Unfortunately, this naive interpolation mechanism does not properly deal with perspective distortions due to foreshortening. The cause of the problem is that interpolation in image space is a bilinear mapping while the true warping function is projective (Heckbert 1989).

A method for correcting this problem is to subdivide each $3D$ face into smaller sections. As each section gets finer in detail, the texture coordinates will be correctly computed and the distortions less noticeable (due to less interpolation). Determining the appropriate level of subdivision is problematic and can lead to either an inefficient over-subdivision of a face or incorrect texturing due to under-subdivision.

An accurate formulation of texture mapping takes into account projective image warping. The mapping between homogeneous image coordinates and homogeneous texture coordinates is viewed in terms of $\mathcal{M}_{image}^{texture}$. Using the form $T = \mathcal{M}_{image}^{texture} \cdot I$:

$$\begin{vmatrix} T_s \\ T_t \\ T_u \end{vmatrix} = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} \begin{vmatrix} I_x \\ I_y \\ I_z \end{vmatrix} \tag{3.17}$$

where T is the texture coordinate corresponding to image coordinate I and i is set to one. In order to solve for the values in this matrix, we first solve for each component:

$$\begin{aligned} T_s &= (a)(I_x) + (b)(I_y) + (c)(I_z) \\ T_t &= (d)(I_x) + (e)(I_y) + (f)(I_z) \\ T_u &= (g)(I_x) + (h)(I_y) + (I_z) \end{aligned} \tag{3.18}$$

Given three points on an image plane and three corresponding points in texture space, the warping matrix is computed. Determining such a matrix is extremely expensive since a matrix is computed for each polygon in the model. Every time the hypothesis about the pose of an object changes, $\mathcal{M}_{image}^{texture}$ must be re-computed.

A slightly different method, called *hyperbolic interpolation*, has been proposed by Blinn (Blinn 1996). Instead of interpolating in pixel space, a similar interpolation is done on the object in camera coordinates. With several acute observations about the effects of homogeneous coordinates, he is able to derive an efficient interpolation.

The question then becomes, which of the two approximation methods (bilinear and hyperbolic) more accurately approximate the effects of perspective foreshortening. Figures 3.16a, b and c show the three different texture mapping methods used to render a simple texture onto a square. The square has been rotated sharply away from the camera to induce perspective foreshortening. The bottom row shows the difference between the two interpolation mechanisms and the correct projective computation (color coded so that a white pixel represents the existence of a difference in pixels).

Under many conditions, the hyperbolic mechanism closely approximates the true projective mapping. While small deviations exist, the amount of computation time required for the proper projection matrix technique far exceeds that of the hyperbolic method. Since predictions need to be accomplished as quickly as possible, we have chosen to use the hyperbolic mechanism.

3.4.2.1 Implementation Considerations: Cost of Texture Maps

Since appearance prediction is used within the iterative RMR loop, rendering must be computationally efficient. Efficiency must not only be in terms of how quickly the objects can be rendered, but also how quickly the image pixels are moved into memory for processing. Remember that our goal is not to produce images to display on a screen, but rather to produce images for pixel-wise comparison with a sensor image.

A problem arises when using traditional OpenGL rendering mechanisms in conjunction with the X windows system. Most X window servers rely on the concept of client and

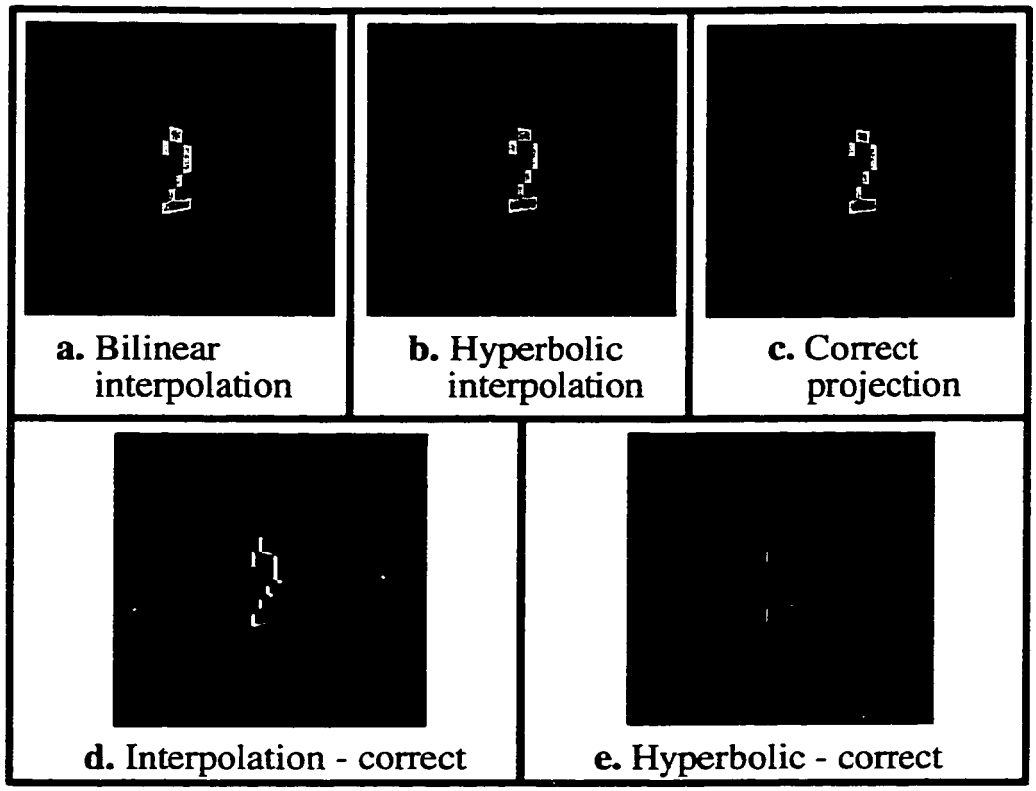

Figure 3.16: Effects of Perspective on Texture Mapping.

server side memory. The server side has low computational overhead for moving pixels to the screen, where as moving pixels from the client to the server has high computational overhead. The goal is to limit the amount of information transferred between the client and server so that images are displayed on the screen as quickly as possible. Most OpenGL packages rely on efficient server side hardware to quickly transfer rendered blocks of pixels, and thus perform most of the difficult rendering work on the server side. To access the pixels produced in such a framework, the information must make a trip back from the server to client memory.

For RMR, we do not need high frame rates in terms of the number of images displayed on a console, but rather demand fast access to images in memory. For this reason, we have chosen to utilize a free version of OpenGL called MESA (Paul 1998). The MESA package renders OpenGL calls directly into memory instead of passing information from client side memory to the server side and finally back to the client. While rendering may not be as efficient, the large penalty of moving data back and forth between client and server is not

incurred. Early timing results showed this extra communication greatly limited the number of images which could be generated and processed per second.

To generate timing information, random scene configurations are rendered. These configurations consist of anywhere from zero to five texture mapped blocks. The time required to render the scene and then visit each pixel and set the memory to zero is recorded. Figure 3.17 shows two separate plots. The first compares the time in machine clock cycles to the number of models. The second figure shows the same time values plotted against the number of object pixels rendered.

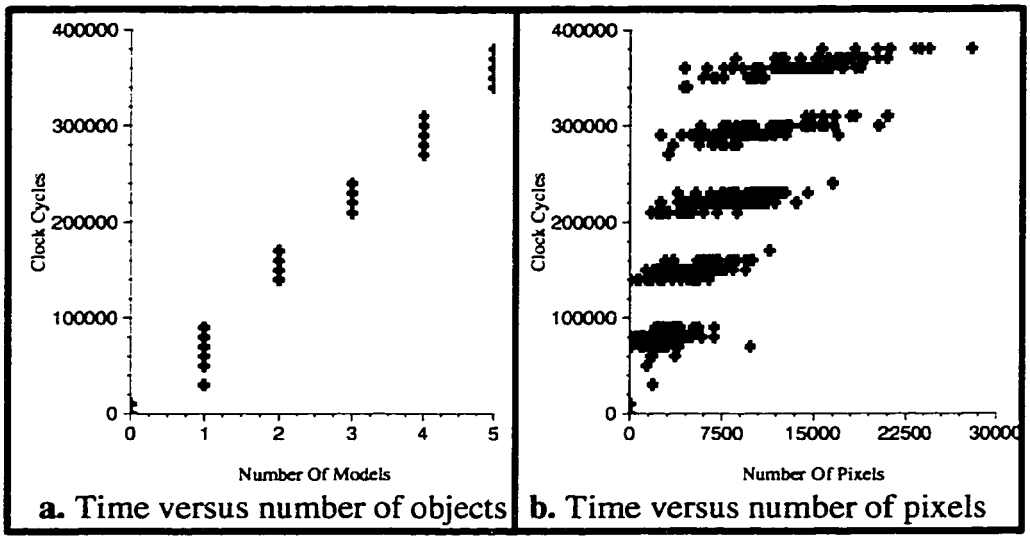

Figure 3.17: Time to render and process an image. All timings are recorded on a Pentium II/266 with $96M$ of RAM running Linux.

When five objects are occupying roughly 30% of the image, 1 prediction per second is processed (assuming a 400Mhz machine). When only three objects are present and 10% of the image is covered, roughly 2 predictions per second are processed. Notice that rendering time is more sensitive to the number of objects to be rendered than the number of pixels. This observation makes sense since rendering each model requires time to load the texture map and geometry into memory.

3.5 Predicting a Simple Background Model

The previous discussion on predicting the appearance of a scene focused on how to color pixels where an object is believed to be present. However, object related pixels usually

comprise only a small portion of the image. A dilemma arises as to what values to use for filling in the remainder of the image. At first glance, we may wish to just fill in non-object pixels with some arbitrary *don't care* label. We can then easily measure the fidelity of our prediction and the sensor data only over predicted object pixels. Unfortunately, this turns out to be a bad idea from a measurement standpoint. The problem stems from comparing sets of different sizes. Consider two configurations: one creates a rendering with 100 pixels, and another produces 1000 object pixels. Many measures of similarity at some point normalize out the effects of size leading to undesirable results (see Chapter 4, Section 4.1.2).

To avoid this problem, we seek to always compare the same number of pixels regardless of scene contents. To simplify the matter, we have chosen to incorporate a background into the prediction mechanism. Therefore, all measurements will always be over every pixel in the image. Although the background model will not be predicted in a robust manner, just having it prevents RMR from favoring one set of configuration parameters over another just because one of the configurations contains more pixels. For the moment, we assume a uniform background model. To estimate the background color, we simply average all of the pixels in the sensor image of the scene. We use this estimated color as the background of our prediction where no objects are present.

Chapter 4

Match: Comparing Images

"Love truth, and pardon error."

- *Voltaire*

After the rendering phase is complete, two images are available: the sensor image and the predicted image. The matching phase of Render-Match-Refine (RMR) is formulated in terms of the difference in pixel values between these two images. This difference or error indicates the quality of the set of scene parameters used to render the prediction. A low error indicates a good alignment of the objects to the imagery. A high error indicates an incorrect alignment. In this manner, the error measure provides insight into when one set of scene parameters provides a better explanation of the sensor image than another set. Typically the error $\mathcal{E}$ is normalized to the range $[0, 1]$:

$$\mathcal{E}(\mathcal{S}) \in [0, 1] \tag{4.1}$$

where $\mathcal{S}$ is a set of scene parameters defined in terms of the objects present:

$$\mathcal{S} = \{\mathcal{F}_{object1}, \mathcal{F}_{object2}, \ldots, \mathcal{F}_{objectN}\} \tag{4.2}$$

When the match is perfect (i.e., the pose parameters used to make the prediction are equivalent to the true values), the function should return 0. As the scene parameters begin to deviate from the ideal, the error values should increase. In practice it is not possible to exactly predict a scene, and therefore, the error will never reach zero. However, the absolute values of the error function are not important, only that 1) the function obtains a minimum at the ideal and 2) ranks competing configurations correctly.

Based on these two criteria, a list of desirable properties for an error function can be defined. These properties are based on the fact that the error function is going to be used

to guide heuristic search. For the moment, assume a simplistic definition of iterative search: given a set of parameters $\mathcal{S}$, a search process generates a new state $\mathcal{S}'$. If $\mathcal{S}'$ has a lower error value than $\mathcal{S}$, $\mathcal{S}'$ replaces $\mathcal{S}$ and search continues. This form of search is critically dependent on the error function: the search process will not converge to the optimal solution if $\mathcal{S}'$ and $\mathcal{S}$ are incorrectly ranked. Conversely, the error function does not depend upon how the state $\mathcal{S}'$ is generated. In fact, any number of search strategies can be defined to exploit the information provided by the error measures discussed here (see Chapter 5). We propose three desirable properties of an RMR error function for search:

1. **Error is minimized by the true scene configuration parameters:**

$$\forall b: \quad \mathcal{E}\left(\mathcal{S}^{true}\right) < \mathcal{E}\left(\mathcal{S}^{b}\right) \quad \text{where} \quad \mathcal{S}^{true} \neq \mathcal{S}^{b} \tag{4.3}$$

   If this criterion is not satisfied, search will often converge to a suboptimal solution.

2. **Error provides a partial ordering of solutions:** Given two solutions, $\mathcal{S}^a$ and $\mathcal{S}^b$, if $\mathcal{S}^a$ is closer to the true configuration than $\mathcal{S}^b$, the error for $\mathcal{S}^a$ should be lower than the error for $\mathcal{S}^b$:

$$\mathcal{E}\left(\mathcal{S}^{a}\right) < \mathcal{E}\left(\mathcal{S}^{b}\right) \Rightarrow \left|\mathcal{S}^{a} - \mathcal{S}^{true}\right| < \left|\mathcal{S}^{b} - \mathcal{S}^{true}\right| \quad \text{where} \quad \mathcal{S}^{a} \neq \mathcal{S}^{b} \tag{4.4}$$

   where $\left|\mathcal{S}^{a} - \mathcal{S}^{true}\right|$ represents some notion of distance between configurations (such as the L1 Norm).

3. **Error must be computationally efficient:** Since the error function is used within a heuristic search algorithm, the error will be evaluated numerous times before converging to a solution. The number of evaluations per second will dominate the total computation time required.

Based on these general properties, a variety of specific error functions can be defined. For many search problems, the first stage in defining an error measure is to create an analytic function of the underlying process. Solving for the best solution is accomplished by taking the derivative of the analytic function, setting the result to zero and solving for the appropriate set of parameters.

An analytic function would be impossible to derive for RMR due to the polygonal rendering used during prediction. In order to model the complex rendering process, either an overly simplistic function or an unwieldy and overly complicated function would be created. In both cases, the analytic function would be extremely weak for dealing with problems involving occlusion: occlusion will introduce a high degree of non-linearity.

Even though an analytic function does not exist, a process model does: rendering. While the explicit derivative of this rendering function does not exist, the error values can be used to guide heuristic search. Therefore, an empirical approach is used to see which of a set of error measures best meet the criteria stated above. Figures 4.1 and 4.2 show the 20 images used for this analysis.

4.1 Defining Specific Error Functions

This section defines two stages used to produce a measure representing the similarity between two images:

1. Define the feature types to use in the comparison.

2. Measure the disparity between feature types.

We have broken the error definition into two stages to clearly delineate what is being compared from how it is being compared.

4.1.1 Generating Features

An image is represented as a collection of pixels. In terms of a data structure, color images are viewed as $3D$ arrays. Two of the dimensions represent the x and y coordinates of the pixel in the image, and the third dimension represents the RGB color band (see Figure 4.3a). A less intuitive way to view an image is as a point embedded in a very high dimensional space. For instance, the images used in this dissertation are all $240x180$ with three color bands. Therefore, each image represents a point in $\Re^{(240)(180)(3)}$ or $\Re^{129,600}$. To form such a point, we unroll the image by copying each pixel from the $3D$ array into a $1D$ feature vector (see Figure 4.3b).

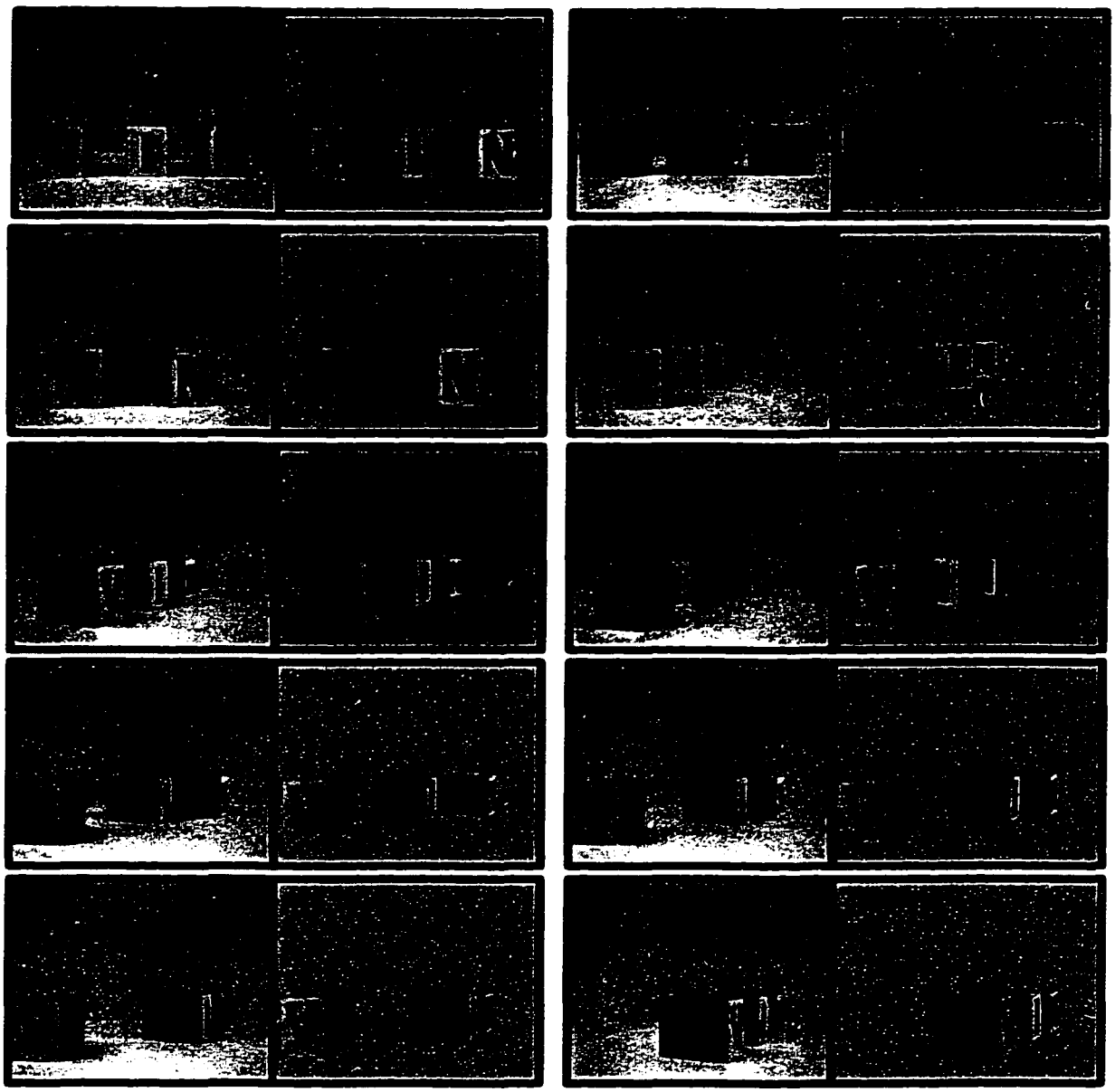

Figure 4.1: A set of example problems used for error evaluation and analysis. The left image of each pair shows the sensor data, and the right image a prediction using the ground truth pose estimates for the given objects.

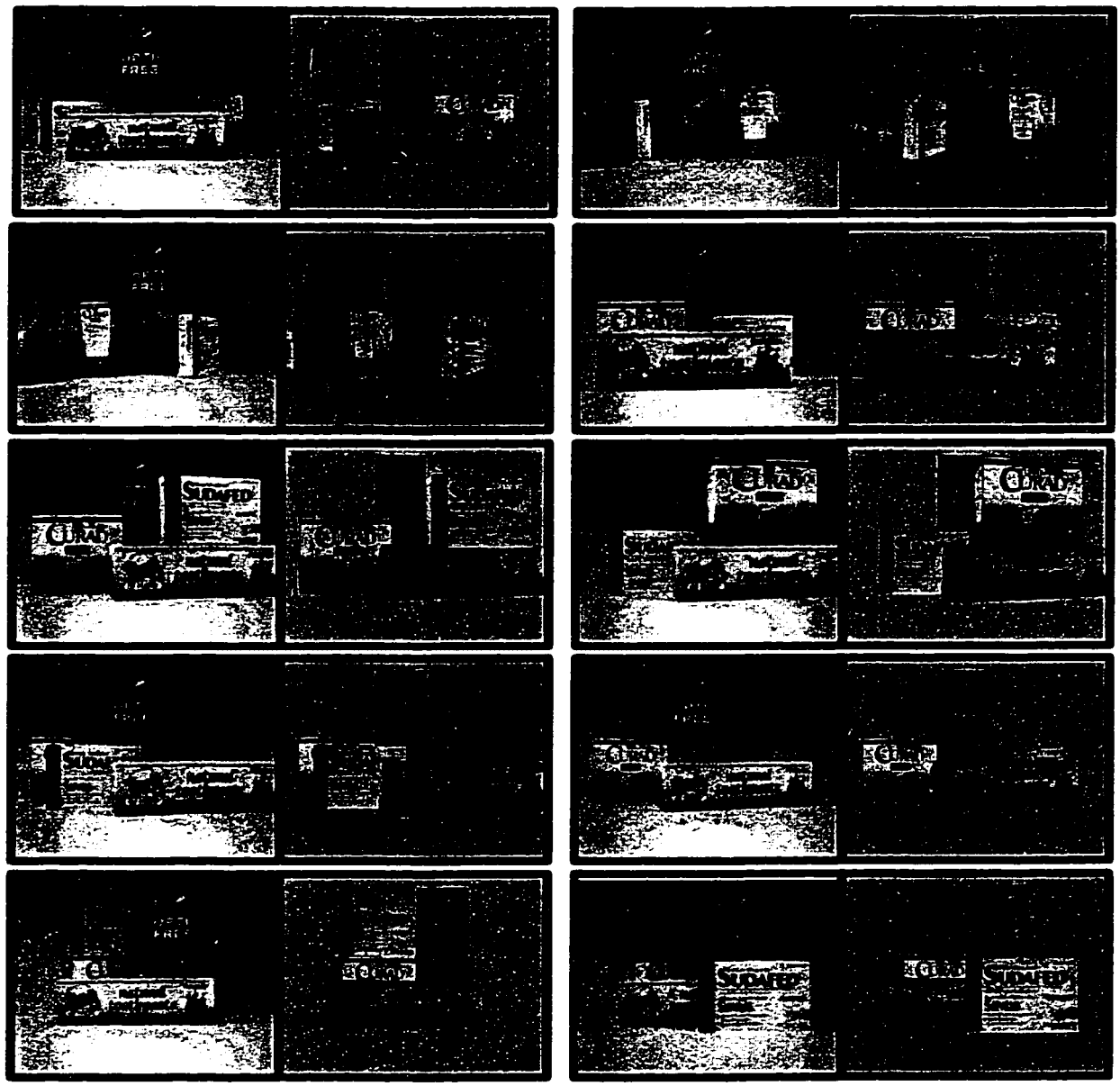

Figure 4.2: A set of example problems used for error evaluation and analysis. The left image of each pair shows the sensor data, and the right image a prediction using the ground truth pose estimates for the given objects.

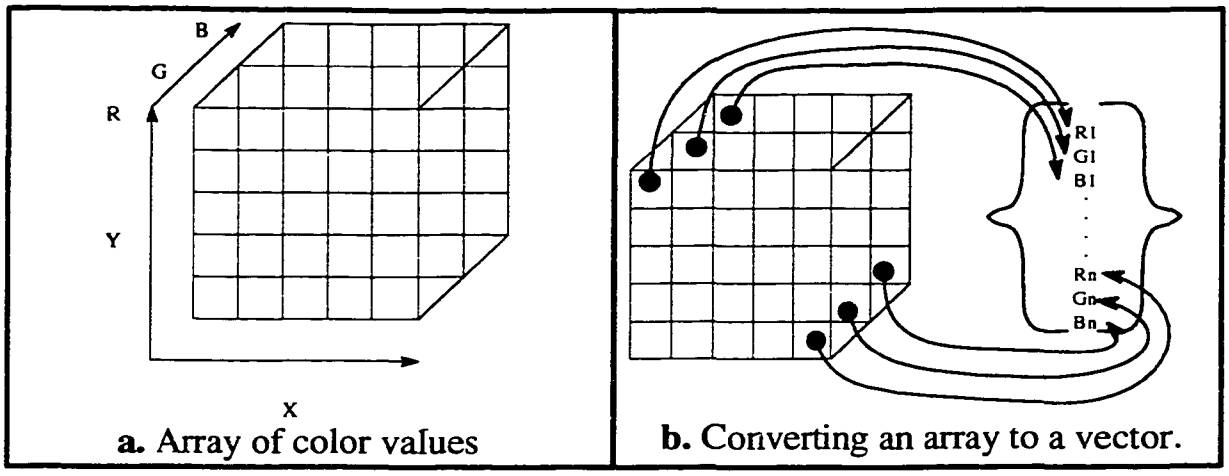

Figure 4.3: Converting from a 3D array to a high dimensional point.

One can view an error function in terms of a comparison of two $1D$ feature vectors:

1. An error function is the distance between the sensor image point and the predicted point in $\Re^{(240)(180)(3)}$. Thus, the goal of search is to find a prediction which lies closest to the sensor point.

2. An error function is the similarity between two $1D$ signals. Here, the goal of search is to make the signals as similar as possible modulo the noise, or discrepancy, within the prediction.

The feature vectors can consist of more than just raw color values. In fact, any sort of measurement about the pixels can be extracted and placed into a vector. Thus, the actual feature vector used by the error function may be larger or smaller than the dimensionality of the raw color vector.

Two categories of features types are considered in this dissertation: color and edge. Color features are based on individual pixel intensities. Edge features are based on gradient estimates which approximate where edges exist in the image. While many more complex features exist (Basri and Jacobs 1995; D.P. Huttenlocher and Rucklidge 1993; Shustorovich 1994), for simplicity we have chosen to implement the following eight:

- RGB: the raw RGB values from the image (Section 4.1.1.1).

- NRGB: normalized color (Section 4.1.1.3).

- GREY: the RGB values converted to grey scale (Section 4.1.1.2).
- HS: the Cartesian coordinates of the HS(V) polar coordinates (Section 4.1.1.4).
- HUE: the Hue component from HSV space (Section 4.1.1.5).
- EDGEI: the estimated partial derivatives of the GREY scale image (Section 4.1.1.6).
- EDGEM: the gradient magnitude of the EDGEI features (Section 4.1.1.8).
- EDGERGB: the estimated partial derivatives of the RGB image (Section 4.1.1.7).

Note that uppercase on each feature type is used to refer to the specific type of feature used as opposed to the concept. For instance, HUE represents hue features as opposed to the general definition of hue.

4.1.1.1 RGB: Red, Green and Blue Color Features

To extract RGB features, the color values in both the observed and predicted images (see Figure 4.4a and 4.4b) are unrolled. These features are represented in terms of the color cube (see Figure 4.4c). All possible colors in a 24bit image can be specified as a single discrete point within this cube (Figures 4.4d and 4.4e). Notice that the distribution of points in the two color cubes are fairly similar. Given the values shown, a feature vector is formed by collecting each ordered point into one of two vectors:

$$\vec{f}^{RGB}_{sensor} = \begin{vmatrix} R_{S1} \\ G_{S1} \\ B_{S1} \\ R_{S2} \\ G_{S2} \\ B_{S2} \\ \cdots \\ R_{Sn} \\ G_{Sn} \\ B_{Sn} \end{vmatrix} \quad \text{and} \quad \vec{f}^{RGB}_{predict} = \begin{vmatrix} R_{P1} \\ G_{P1} \\ B_{P1} \\ R_{P2} \\ G_{P2} \\ B_{P2} \\ \cdots \\ R_{Pn} \\ G_{Pn} \\ B_{Pn} \end{vmatrix} \tag{4.5}$$

where (R_{S1}, G_{S1}, B_{S1}) is the first RGB pixel in scan-line order from the sensor image, and (R_{P1}, G_{P1}, B_{P1}) is the corresponding RGB value from the predicted image. Since the order in which the pixel values are added is the same for both feature vectors, an explicit correspondence is being made. This correspondence implies (R_{S1}, G_{S1}, B_{S1}) should equal

(R_{P1}, G_{P1}, B_{P1}). We know the prediction will contain a certain amount of error, which can be thought of as local error, in the form of noise:

$$\vec{f}^{RGB}_{sensor} \approx \vec{f}^{RGB}_{predict} + \vec{G}^{RGB} \tag{4.6}$$

where $\vec{G}^{RGB}$ is an estimate of the noise function. In the controlled blocks-world and products-world domains, only three factors should influence $\vec{G}^{RGB}$: 1) variations in object face illumination due to the object rotating and translating with respect to the camera and light sources (observed colors are brighter or darker than predicted), 2) quantization errors due to the discrete nature of CCD cameras (observed colors are different due to sampling) and 3) reflections from other objects (observed colors are influenced by other objects). To better quantify the underlying noise distribution, $\vec{G}^{RGB}$ is examined:

$$\vec{G}^{RGB} \approx \vec{f}^{RGB}_{sensor} - \vec{f}^{RGB}_{predict} \tag{4.7}$$

Figure 4.4f shows a histogram plot of $\vec{G}^{RGB}$ across all twenty test images where $\vec{f}^{RGB}_{predict}$ comes from a prediction rendered using the ground truth scene parameters[1]. To better isolate the effect of the noise on each band, the R, G, and B bands have been separated. All of the histograms have very wide tails and are multi-modal. This effect can be accounted for based on the hypothesized factors influencing the difference between the predicted and the observed data. While quantization error should produce small errors in $\vec{G}^{RGB}$, the larger differences are due to changes in scene illumination: the sensor pixels are brighter than the prediction.

4.1.1.2 GREY: Grey Scale Color Features

The histograms in Figure 4.4f show each of the three color distributions have similar differences from the prediction (the shapes of the distributions are similar). This is due to the existence of redundant information in each color band. Therefore, a feature space based only on predicted image intensity, rather than color, is examined. Figures 4.5a and 4.5b

[1]The characterization of the noise function G across all feature types in this dissertation uses the ground truth parameters for determining $\vec{f}_{predict}$

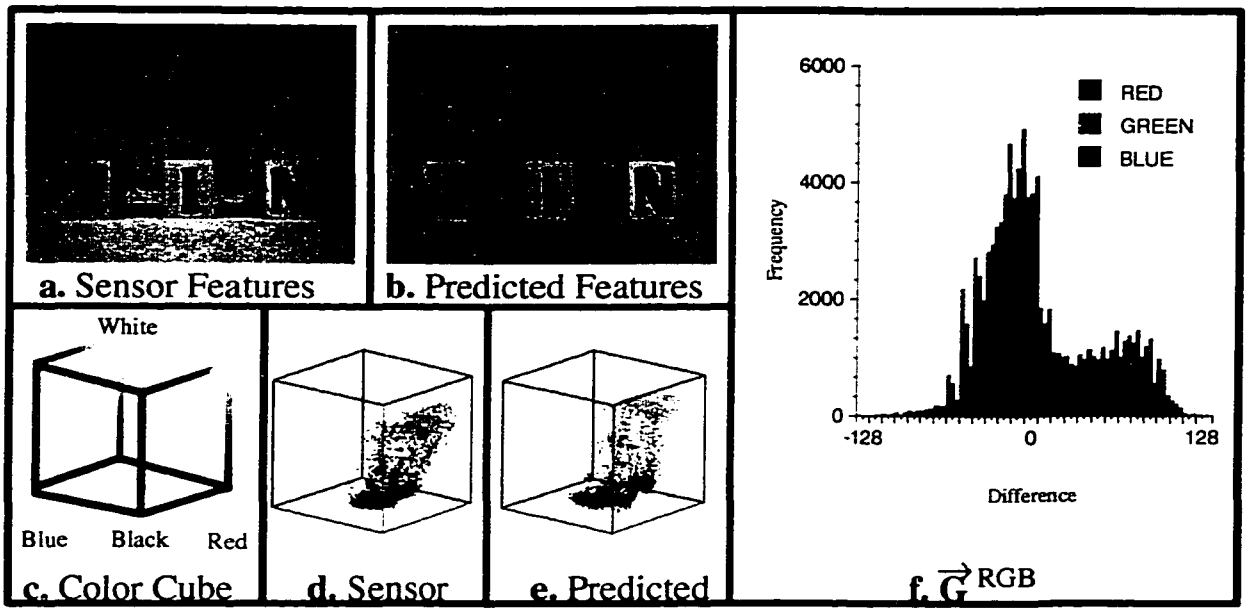

Figure 4.4: The RGB features.

show a grey scale sensor image and the corresponding prediction. Figure 4.5c shows the distributions of sensor and predicted intensities. This feature space, referred to as grey scale, sums each color band:

$$\vec{f}^{GREY}_{sensor} = \begin{vmatrix} R_{S1} + G_{S1} + B_{S1} \\ R_{S2} + G_{S2} + B_{S2} \\ \cdots \\ R_{Sn} + G_{Sn} + B_{Sn} \end{vmatrix} \quad \text{and} \quad \vec{f}^{GREY}_{predict} = \begin{vmatrix} R_{P1} + G_{P1} + B_{P1} \\ R_{P2} + G_{P2} + B_{P2} \\ \cdots \\ R_{Pn} + G_{Pn} + B_{Pn} \end{vmatrix} \tag{4.8}$$

The noise function $\vec{G}^{GREY}$ is:

$$\vec{G}^{GREY} \approx \vec{f}^{GREY}_{sensor} - \vec{f}^{GREY}_{predict} \tag{4.9}$$

which is shown as a histogram in Figure 4.5d for the twenty example problems. It is obvious from the plot that the same problem with using RGB is resurfacing: the noise distribution has both extremely wide tails and is multi-modal. The hypothesized reason is again due to illumination: the second peak implies the sensor pixels are much brighter than the predicted pixels.

4.1.1.3 NRGB: Normalized Red, Green and Blue Color Features

Scene illumination is an integral factor affecting the observed object color at any given pixel. To compensate for the effects of illumination, two approaches are proposed: 1) modify the

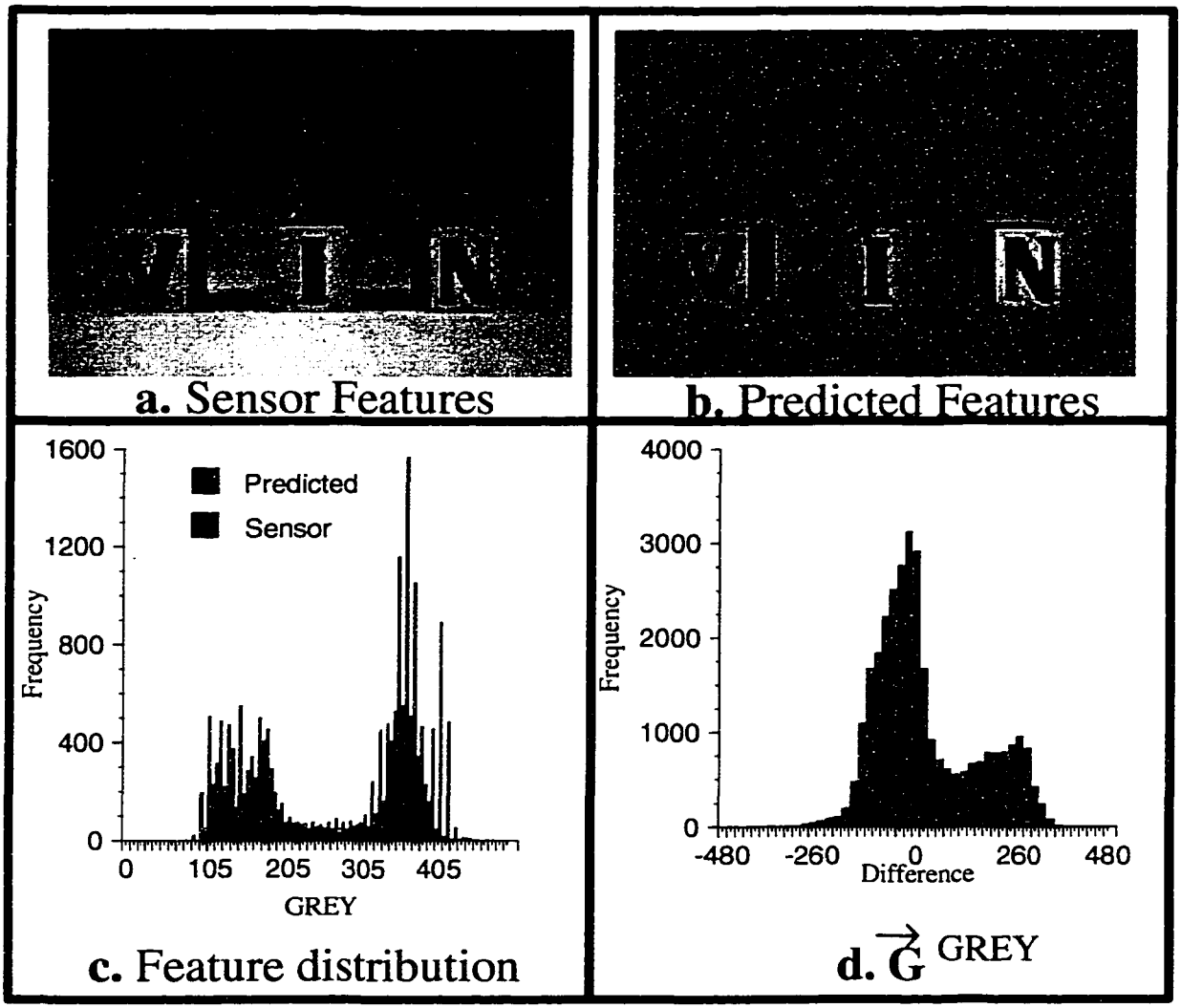

Figure 4.5: GREY.

prediction mechanism to account for changes in illumination or 2) switch to a different feature space. The former approach could be accomplished by hypothesizing a light source and incorporating that knowledge into the rendering. The parameter space for the error function would be defined over the hypothesized poses of each object as well as the location of the lights. While this approach may seem appealing, it is very difficult to model all possible light sources and lighting effects.

The practical solution is to define a new feature space which is less sensitive to changes in individual pixel intensities. One such feature space, which is referred to as *normalized color*, is formed by dividing each color pixel component by its total energy:

$$r' = \frac{R}{R+G+B} \quad g' = \frac{G}{R+G+B} \quad b' = \frac{B}{R+G+B} \quad \text{where} \quad r'+g'+b' = 1 \tag{4.10}$$

Note that the color black is undefined in this space; for that special case the obvious solution is to set each term to zero. Only two components are needed to represented a color since the three terms sum to unity.

Figures 4.6a and b show the working blocks-world example. The colors in the predicted and sensor image were converted to normalized color, and then the green band was discarded. Figures 4.6c and d show the projection into normalized RGB ($NRGB$) space. Again, note that only the r' and b' components were used to make this plot. The $NRGB$ feature vectors are:

$$\vec{f}^{NRGB}_{sensor} = \begin{vmatrix} r'_{S1} \\ b'_{S1} \\ r'_{S2} \\ b'_{S2} \\ \cdots \\ r'_{Sn} \\ b'_{Sn} \end{vmatrix} \quad \text{and} \quad \vec{f}^{NRGB}_{predict} = \begin{vmatrix} r'_{P1} \\ b'_{P1} \\ r'_{P2} \\ b'_{P2} \\ \cdots \\ r'_{PN} \\ b'_{PN} \end{vmatrix} \tag{4.11}$$

The choice of which two out of three bands to use is quite arbitrary. The r' and b' bands were chosen since they are farther apart in terms of wavelength. The difference between feature vectors is again characterized in terms of a noise function $\vec{G}^{NRGB}$:

$$\vec{G}^{NRGB} \approx \vec{f}^{NRGB}_{sensor} - \vec{f}^{NRGB}_{predict} \tag{4.12}$$

Figure 4.6e shows a histogram of $\vec{G}^{NRGB}$ for the twenty test images. Switching to $NRGB$ has effectively removed feature sensitivity to changes in pixel intensity.

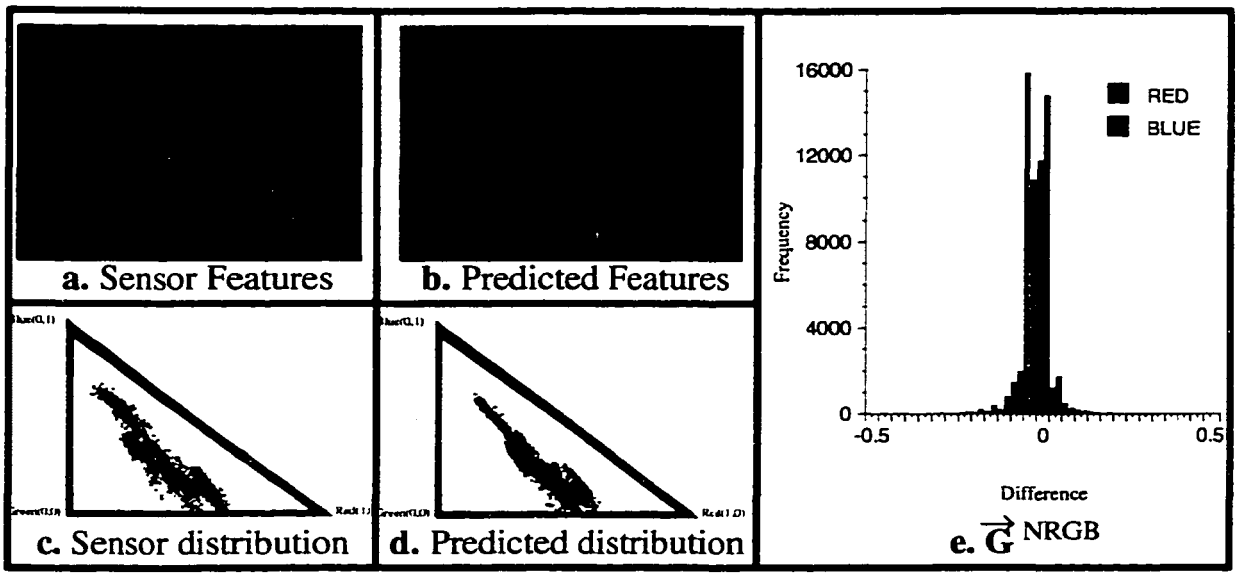

Figure 4.6: The NRGB features.

4.1.1.4 HS: Hue Saturation Color Features

$NRGB$ is only one possible feature which reduces sensitivity to intensity (brightness). Another possible color space, hue-saturation (HS), is also examined as a means for coping with illumination affects. HS features are derived from the HSV color space. HSV values are defined in terms of the conceptual definitions of color (Foley and vanDam 1982):

hue: used to distinguish among colors such as red, or blue.

saturation: defines the distance the color is from a similar grey of equal intensity.

value: refers to how bright the pixel appears.

HSV nicely separates the color information into two categories: what can (color) and cannot (intensity) be robustly predicted. Since the prediction mechanism is not able to reliably predict the v component (based on the histograms for $\vec{G}^{GREY}$), the term can be removed from the feature vector.

This feature space is conceptualized in terms of the HS plane. The HS plane normal is the vector from black to white in the RGB color cube. When the RGB values are projected downward along that axis, a hexacone is formed (see the HS plane in relation to the RGB color cube in Figure 4.7c). HSV space represents colors on this hexacone in terms of polar

coordinates. The hue value defines the angle around the vertical axis, and saturation is the distance along this line at which the desired color lies. The HS features are formed by discarding the v term. For instance, pure red has a hue angle of 0° and a saturation of 1, green has $(120°, 1)$, and blue $(240°, 1)$.

One of the major drawbacks in projecting the higher dimensional RGB values onto the lower dimensional HS is the loss of information[2]. This loss is most evident in the fact that pure white and pure black project to the same point. This contrasts with $NRGB$ where black and white project to different points. Besides data loss, hue features are known to become wildly unstable as the saturation value approaches zero. The stability problem is addressed by representing HS features in Cartesian coordinates as opposed to the original polar coordinates:

$$X = S\cos H \quad \text{and} \quad Y = S\sin H \tag{4.13}$$

The loss of information due to projection is unavoidable for HS features. Figures 4.7a and 4.7b show the working example color coded so that the X occupies the red band and Y the blue. Figure 4.7d and 4.7e show the object pixels in the HS coordinate system. The feature vector is formed by unrolling the HS Cartesian coordinates for each pixel in the predicted and sensor images:

$$\vec{f}^{HS}_{sensor} = \begin{vmatrix} X_{S1} \\ Y_{S1} \\ X_{S2} \\ Y_{S2} \\ \cdots \\ X_{Sn} \\ Y_{Sn} \end{vmatrix} \quad \text{and} \quad \vec{f}^{HS}_{predict} = \begin{vmatrix} X_{P1} \\ Y_{P1} \\ X_{P2} \\ Y_{P2} \\ \cdots \\ X_{Pn} \\ Y_{Pn} \end{vmatrix} \tag{4.14}$$

and the noise function is:

$$\vec{G}^{HS} \approx \vec{f}^{HS}_{sensor} - \vec{f}^{HS}_{predict} \tag{4.15}$$

Figure 4.7f shows the histogram for $\vec{G}^{HS}$ over the twenty test images. Again, errors due to changes in illumination have been effectively removed.

---

[2]Projection in this context defines the process of moving from the three dimensional color cube to the two dimensional plane.

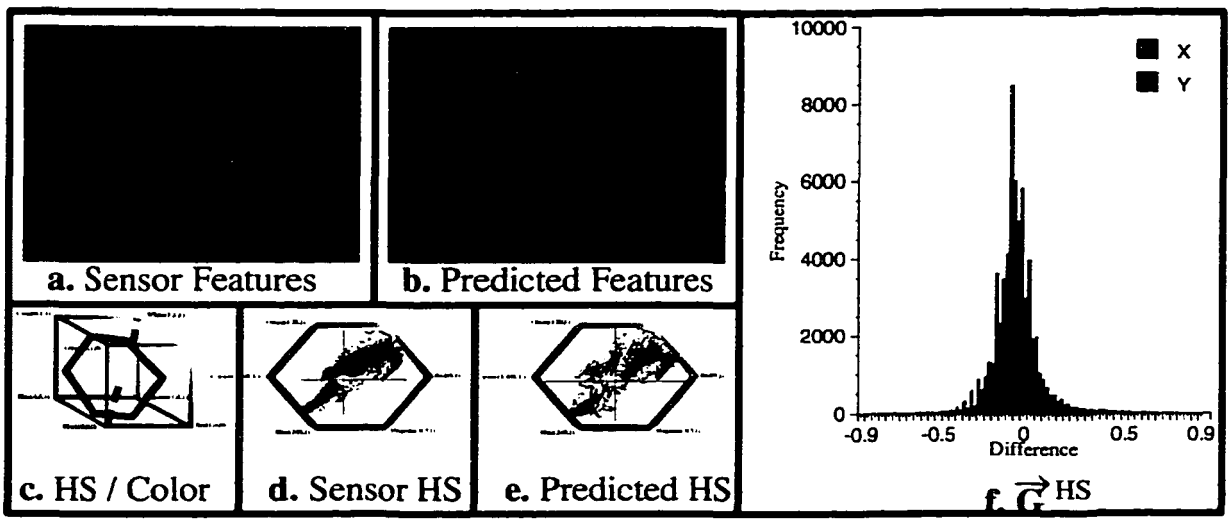

Figure 4.7: The HS features.

4.1.1.5 HUE: Hue Color Features

The previous four features have all characterized different aspects of color: raw values (RGB), intensity (GREY), and color with intensity removed (HS and $NRGB$). The final color feature examined is that of hue. Figures 4.8a and 4.8b show the hue values for the blocks-world working example. For visualization, the hue values have been converted to RGB by assuming a saturation and value of one for each computed hue. Figure 4.8c shows the hue distributions for the two images. Notice that there are dominant peaks around 0°, 50°, 95° and 240°. These correspond roughly to red, yellow, green and blue, which are the colors of the various faces of the blocks.

The feature vector for hue requires less effort to compute than the vector for HS space (no *sine* and *cosine* computations required) and amounts to just unrolling the hue values for each pixel:

$$\vec{f}^{HUE}_{sensor} = \begin{vmatrix} H_{S1} \\ H_{S2} \\ \cdots \\ H_{Sn} \end{vmatrix} \quad \text{and} \quad \vec{f}^{HUE}_{predict} = \begin{vmatrix} H_{P1} \\ H_{P2} \\ \cdots \\ H_{Pn} \end{vmatrix} \tag{4.16}$$

The noise function is:

$$\vec{G}^{HUE} \approx \vec{f}^{HUE}_{sensor} - \vec{f}^{HUE}_{predict} \tag{4.17}$$

Figure 4.8 shows the histogram for $\vec{G}^{HUE}$. Since intensity information has been removed from the HUE feature space, the other peaks in the distribution must be due to quantization effects or reflections.

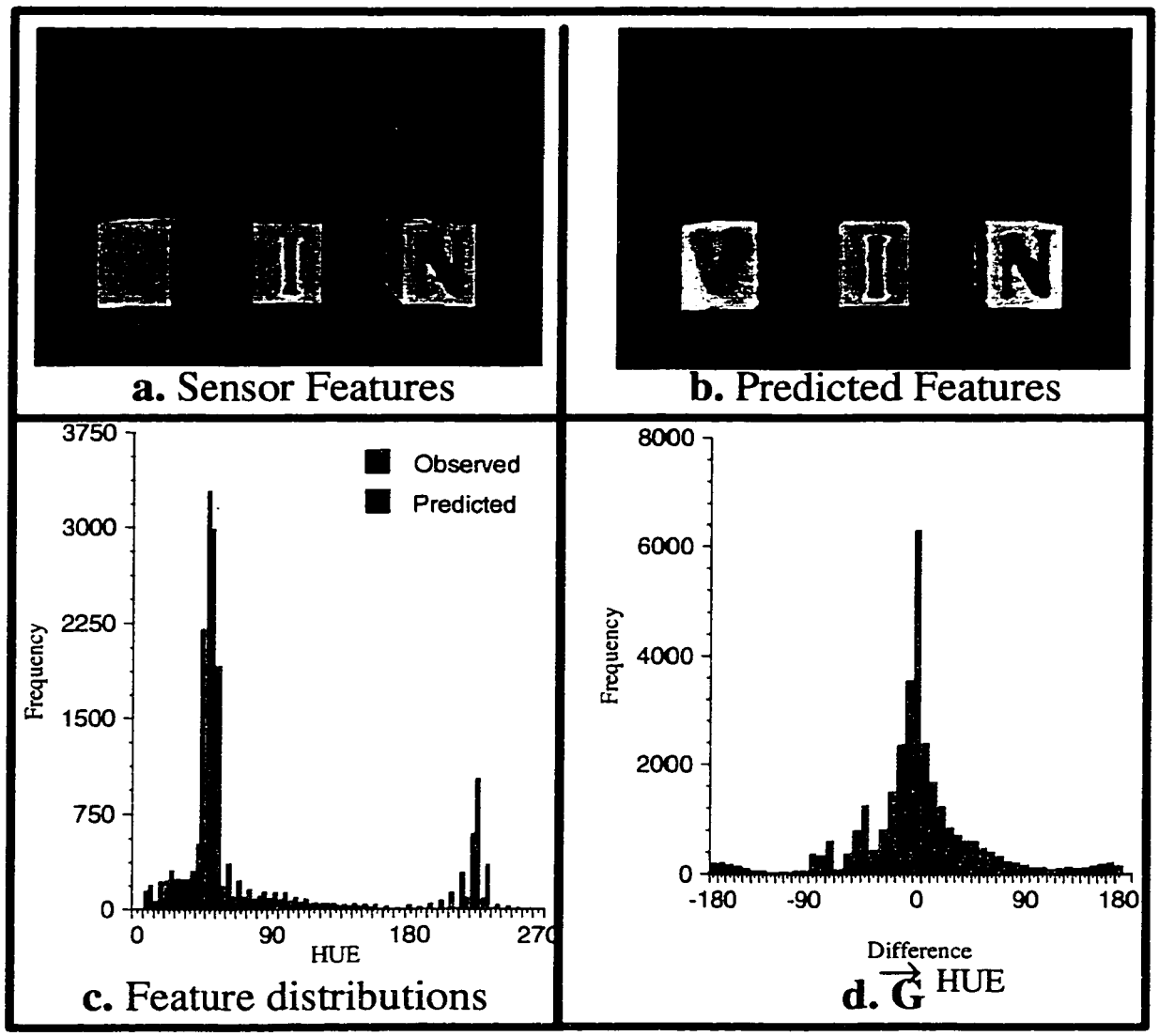

Figure 4.8: The HUE features.

4.1.1.6 EDGEI: Intensity Edge Features

Normalized color and HS features were introduced to deal with a simple observation about the prediction mechanism: predictions of color properties appear to be more reliable than predictions about the absolute brightness of the rendered pixels. Unfortunately, many factors outside of the simple texture-mapping prediction model can influence the observed

pixel color. For instance, the position of light sources, color of light sources, inter-reflections among objects, and specular reflections all could alter the color of an object. To overcome the limitations of color, others have turned to edge detection (Marr and Hildreth 1980; Hildreth 1983; Canny 1986). An edge is defined as the border between two dissimilar image regions. Edges in an image are caused by several physical events in the scene (Nalwa 1993):

1. **Surface normal discontinuities:** orientation changes between adjacent surfaces.
2. **Depth discontinuities:** one surface is farther away from the camera than the neighboring visible surface.
3. **Surface reflectance discontinuities:** neighboring surfaces have different reflectance properties.
4. **Illumination discontinuities:** one face receives more light than its neighbor (either from different lights or by reflection from other objects).

Often the practice of edge detection boils down to image differentiation (Peli and Malah 1982). Since a digital image is a discrete collection of pixels, the derivative is ill-defined. Therefore, when referring to image gradients, there is an implication that a discrete derivative is being used to estimate the underlying continuous image irradiance function (Horn 1986; Nalwa 1993). To estimate the discrete derivative of an image, the image is convolved with a set of masks. These masks are designed to measure the rate of change in pixel intensity in a certain direction. Masks contain a set of weights describing how much each pixel contributes to the gradient estimate. Typically, pixels farther from the edge are weighted less. While many masks have been proposed (Chaudhuri and Chanda 1984; Nevatia and Babu 1980), the following $1D$ mask $\vec{A}$ is used:

$$\vec{A} = \left| \begin{array}{ccccccccccc} -1 & -2 & -3 & -4 & -5 & 0 & +5 & +4 & +3 & +2 & +1 \end{array} \right| \tag{4.18}$$

This mask was chosen because it provides gradient information for a large area about an edge: a wide range of pixels about the true edge will have a non-zero response which will increase in magnitude closer to the true edge. Therefore, even when the pose of the object

is off by several pixels, there should still be some overlap of gradient values between the predicted and observed images. The estimated derivatives are:

$$\frac{\partial I}{\partial X}_{i,j} \approx \vec{X}_{i,j} \cdot \vec{A} \tag{4.19}$$

and:

$$\frac{\partial I}{\partial Y}_{i,j} \approx \vec{Y}^{\top}_{i,j} \cdot \vec{A} \tag{4.20}$$

For $EDGEI$ features, the $\vec{X}_{i,j}$ and $\vec{Y}^{\top}_{i,j}$ vectors are formed from the grey scale values of the image:

$$\vec{X}_{i,j} = \begin{vmatrix} R_{i-5,j} & + & G_{i-5,j} & + & B_{i-5,j} \\ & & \cdots & & \\ R_{i,j} & + & G_{i,j} & + & B_{i,j} \\ & & \cdots & & \\ R_{i+5,j} & + & G_{i+5,j} & + & B_{i+5,j} \end{vmatrix} \tag{4.21}$$

and $\vec{Y}^{\top}_{i,j}$ is defined as:

$$\vec{Y}^{\top}_{i,j} = \begin{vmatrix} R_{i,j-5} & + & G_{i,j-5} & + & B_{i,j-5} \\ & & \cdots & & \\ R_{i,j} & + & G_{i,j} & + & B_{i,j} \\ & & \cdots & & \\ R_{i,j+5} & + & G_{i,j+5} & + & B_{i,j+5} \end{vmatrix} \tag{4.22}$$

Figures 4.9a through 4.9d shows the edges detected for the sensor and predicted images of the blocks-world scene. The left column represents $\frac{\partial I}{\partial X}$ and the middle column is $\frac{\partial I}{\partial Y}$. Grey regions in the images represent weak gradient response and dark or white regions represent strong negative or positive responses respectively.

Two observations can be made about these images. First, the gradient estimates for the predicted images match well where there is an edge event. However, where there is no edge the sensor image contains small gradient responses due to the non-uniform image background (the prediction does not since the uniform background is used during rendering). Second, notice that the derivatives in the X direction (Figures 4.9a and 4.9c) contain long vertical regions of strong mask response while the derivatives in the Y direction (Figures 4.9b and 4.9d) contain long horizontal regions of strong mask response. This effect results from the edge detection mask being designed to detect horizontal and vertical edges.

The $EDGEI$ feature vector is the collection of estimated image derivatives:

$$\vec{f}^{EDGEI}_{sensor} = \begin{vmatrix} \frac{\partial I}{\partial X} S1 \\ \frac{\partial I}{\partial Y} S1 \\ \frac{\partial I}{\partial X} S2 \\ \frac{\partial I}{\partial Y} S2 \\ \cdots \\ \frac{\partial I}{\partial X} Sn \\ \frac{\partial I}{\partial Y} Sn \end{vmatrix} \quad \text{and} \quad \vec{f}^{EDGEI}_{predict} = \begin{vmatrix} \frac{\partial I}{\partial X} P1 \\ \frac{\partial I}{\partial Y} P1 \\ \frac{\partial I}{\partial X} P2 \\ \frac{\partial I}{\partial Y} P2 \\ \cdots \\ \frac{\partial I}{\partial X} Pn \\ \frac{\partial I}{\partial Y} Pn \end{vmatrix} \tag{4.23}$$

where $S1, S2, \ldots, Sn$ represent sensor pixels and $P1, P2, \ldots, Pn$ are pixels from the predicted image. Again, the noise function is characterized in terms of the difference between feature vectors:

$$\vec{G}^{EDGEI} \approx \vec{f}^{EDGEI}_{sensor} - \vec{f}^{EDGEI}_{predict} \tag{4.24}$$

Figure 4.9 shows the histogram of the noise function accumulated across all images in the two test suites.

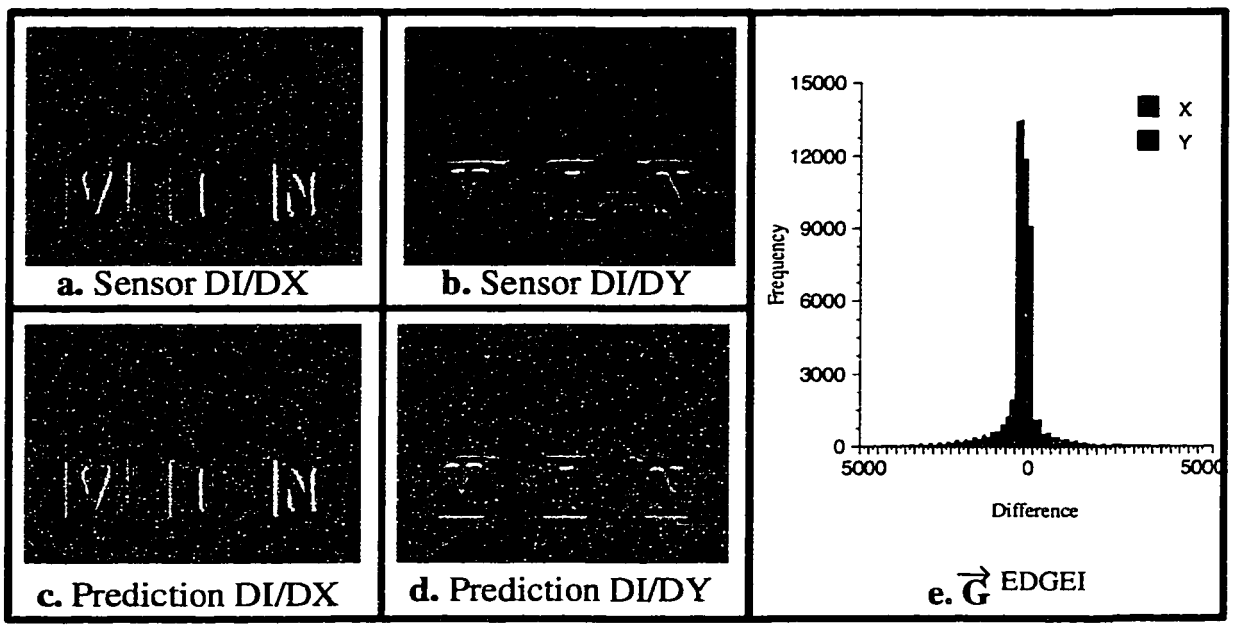

Figure 4.9: The $EDGEI$ features.

4.1.1.7 EDGERGB: Color Edge Features

An obvious problem exists with using gradient estimates based on the total energy absorbed at each pixel: if a purely red surface is adjacent to a purely blue surface in the scene, the previous edge detection mechanism will not find an edge between the two surfaces. To

overcome this deficiency, the derivatives are estimated for each color band independently. Using such a feature space allows the detection of gradient changes across individual bands, but not for combinations. For instance, a completely yellow surface adjacent to a completely red surface will show very little response to the gradient masks in the red band. However, a stronger response should show up in the green band.

The same mask as was used for intensity edge detection can now be used to estimate the derivative across each band:

$$\left\{\frac{\partial R}{\partial X}, \frac{\partial R}{\partial Y}, \frac{\partial G}{\partial X}, \frac{\partial G}{\partial Y}, \frac{\partial B}{\partial X}, \frac{\partial B}{\partial Y}\right\} \tag{4.25}$$

Instead of using the total pixel energy, the raw color values are used. For instance, the $\vec{X}_{i,j}$ and $\vec{Y}_{i,j}$ vectors for $\frac{\partial R}{\partial X}$ and $\frac{\partial R}{\partial Y}$ are:

$$\vec{X}_{i,j} = \begin{vmatrix} R_{i-5,j} \\ \cdots \\ R_{i,j} \\ \cdots \\ R_{i+5,j} \end{vmatrix} \tag{4.26}$$

and:

$$\vec{Y}_{i,j}^{\top} = \begin{vmatrix} R_{i,j-5} \\ \cdots \\ R_{i,j} \\ \cdots \\ R_{i,j+5} \end{vmatrix} \tag{4.27}$$

For the other two bands, one should replace the R components with either the G or B values. Figures 4.10a through 4.10l show the 12 images of the partial differential for each band in both the X and Y directions for the working example. Notice that the blue band seems to have less recognizable gradient structure than either the green or the red. Similar to the edges computed over the pixel intensities, long horizontal regions of strong gradient response appear for the derivative estimates in the Y direction and long vertical regions appear for the X derivative estimates.

The $EDGERGB$ feature vector is defined in terms of the mask response for each band in both directions. Unfortunately, this vector is now twice as large as the vector formed directly from the RGB intensity values:

$$\vec{f}_{sensor}^{EDGERGB} = \begin{vmatrix} \frac{\partial R}{\partial X}S1 \\ \frac{\partial R}{\partial Y}S1 \\ \frac{\partial G}{\partial X}S1 \\ \frac{\partial G}{\partial Y}S1 \\ \frac{\partial B}{\partial X}S1 \\ \frac{\partial B}{\partial Y}S1 \\ \dots \\ \frac{\partial R}{\partial X}Sn \\ \frac{\partial R}{\partial Y}Sn \\ \frac{\partial G}{\partial X}Sn \\ \frac{\partial G}{\partial Y}Sn \\ \frac{\partial B}{\partial X}Sn \\ \frac{\partial B}{\partial Y}Sn \end{vmatrix} \quad \text{and} \quad \vec{f}_{predict}^{EDGERGB} = \begin{vmatrix} \frac{\partial R}{\partial X}P1 \\ \frac{\partial R}{\partial Y}P1 \\ \frac{\partial G}{\partial X}P1 \\ \frac{\partial G}{\partial Y}P1 \\ \frac{\partial B}{\partial X}P1 \\ \frac{\partial B}{\partial Y}P1 \\ \dots \\ \frac{\partial R}{\partial X}Pn \\ \frac{\partial R}{\partial Y}Pn \\ \frac{\partial G}{\partial X}Pn \\ \frac{\partial G}{\partial Y}Pn \\ \frac{\partial B}{\partial X}Pn \\ \frac{\partial B}{\partial Y}Pn \end{vmatrix} \tag{4.28}$$

The noise function is the difference between feature vectors:

$$\vec{G}^{EDGERGB} \approx \vec{f}_{sensor}^{EDGERGB} - \vec{f}_{predict}^{EDGERGB} \tag{4.29}$$

Figure 4.10m shows the histogram of the noise function for the twenty test problems. Again, for clarity, the histogram was generated for each differentiated color band. Notice the large discrepancy between the predicted and observed $\frac{\partial B}{\partial Y}$ which is partially due to the large amount of salt and pepper noise in Figure 4.10k.

4.1.1.8 EDGEM: Magnitude Edge Geometric Features

The edge detection process may be viewed as fitting a local plane to the underlying pixel intensities (Burns, Hanson, and Riseman 1986). Using the partial derivative estimates for $\frac{\partial I}{\partial X}$ and $\frac{\partial I}{\partial Y}$, the normal of this plane can be determined. This vector has both a magnitude:

$$mag = \sqrt{\left(\frac{\partial I}{\partial X}\right)^2 + \left(\frac{\partial I}{\partial Y}\right)^2} \tag{4.30}$$

and an orientation:

$$orn = \tan^{-1}\left(\frac{\frac{\partial I}{\partial Y}}{\frac{\partial I}{\partial X}}\right) \tag{4.31}$$

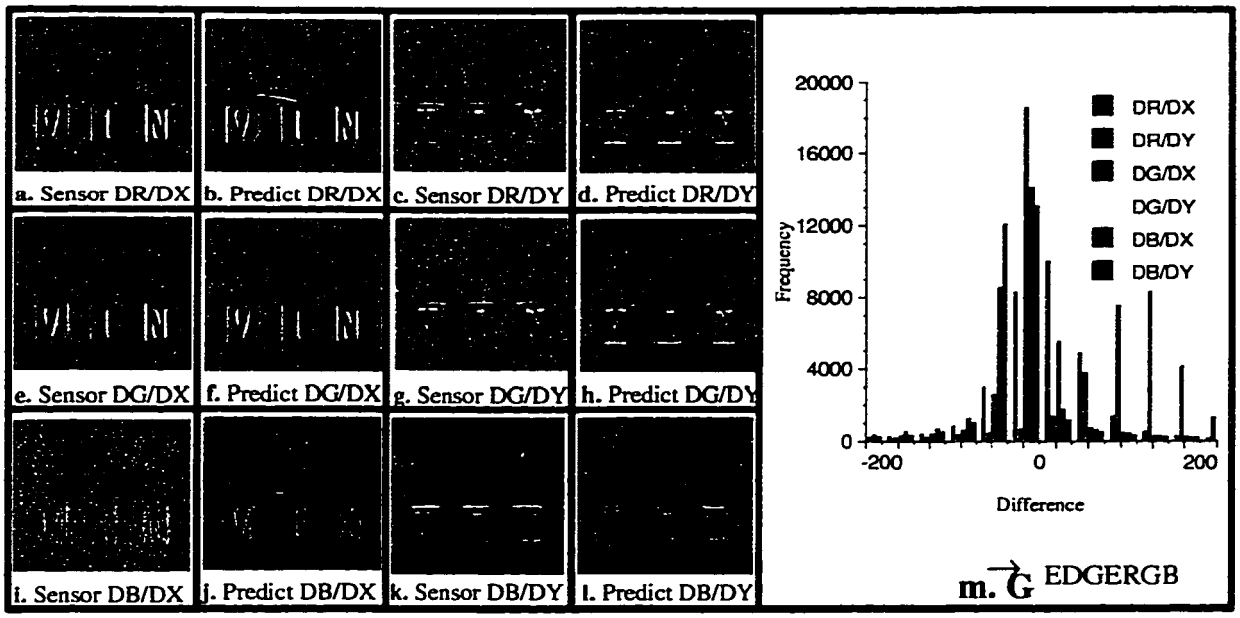

Figure 4.10: Edge gradients based on the RGB values.

Since others have observed that the orientation component is often difficult to estimate (Davies 1987), the final feature examined discards the orientation information. This feature vector is constructed from the magnitude estimates for each pixel (see Figure 4.11):

$$\vec{f}^{EDGEM}_{sensor} = \begin{vmatrix} mag\ S1 \\ mag\ S2 \\ \dots \\ mag\ Sn \end{vmatrix} \quad \text{and} \quad \vec{f}^{EDGEM}_{predict} = \begin{vmatrix} mag\ P1 \\ mag\ P2 \\ \dots \\ mag\ Pn \end{vmatrix} \tag{4.32}$$

with the noise function estimated by:

$$\vec{G}^{EDGEM} \approx \vec{f}^{EDGEM}_{sensor} - \vec{f}^{EDGEM}_{predict} \tag{4.33}$$

Figures 4.11c and 4.11d show a histogram plot for this noise function.

4.1.2 Measuring Feature Similarity

Formulating the error function began with the problem of comparing two images. After the introduction of features, the error function becomes the comparison of two feature vectors. Now the comparison of features must be mapped to a real number:

$$\mathcal{E}(\mathcal{S}) \mapsto \mathcal{E}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) \mapsto [0,1] \tag{4.34}$$

where $\vec{f}_{sensor}$ represents a generic type of feature vector from the sensor image and $\vec{f}_{predict}$ a generic vector from the prediction. To compare feature vectors, three similarity measures

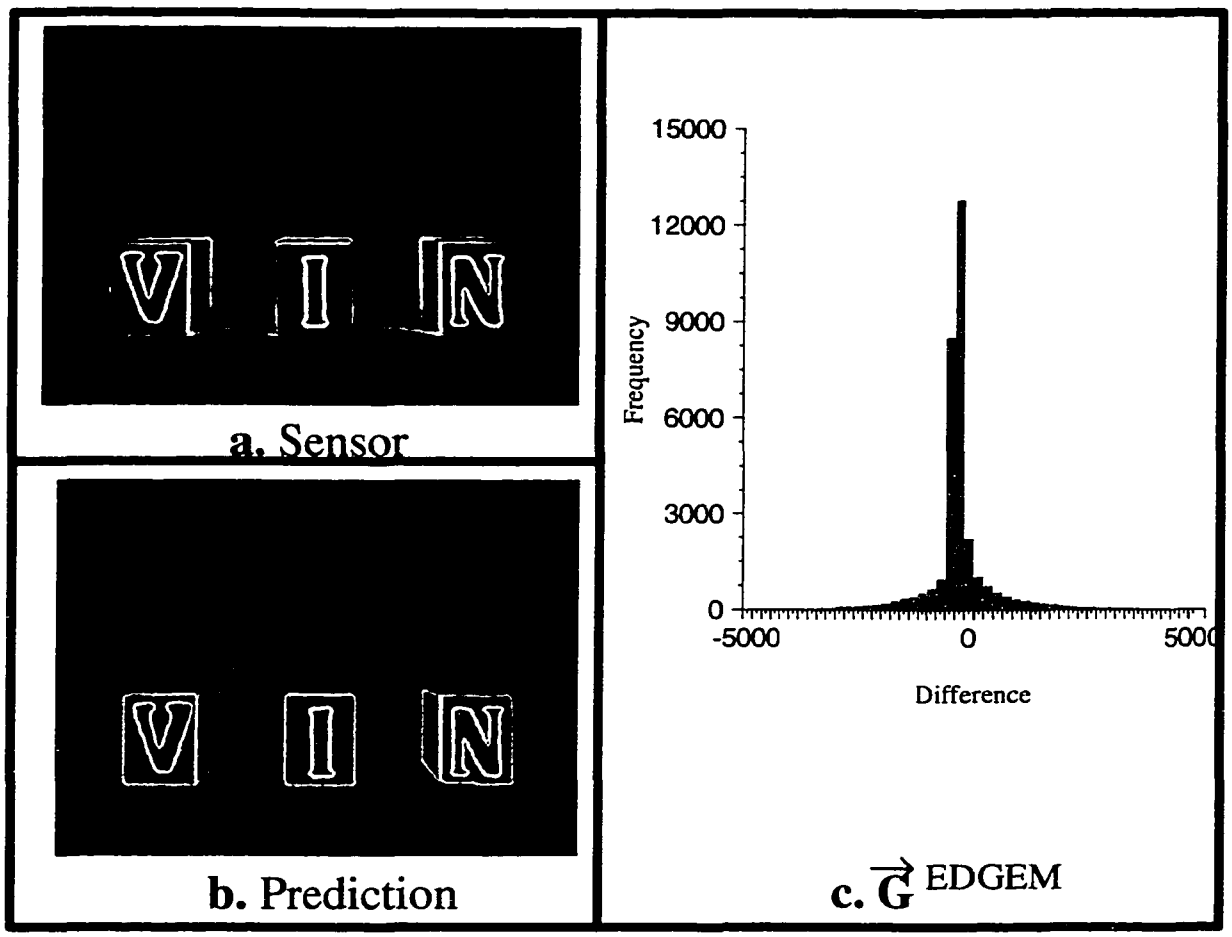

Figure 4.11: Edge Magnitude features.

were selected from the literature (Crowley and Martin 1995; Duda and Hart 1973) and implemented within RMR:

1. **Euclidean Distance:** Since each feature vector represents a point in a very high dimensional space, perhaps the easiest comparison mechanism is to measure the Euclidean distance between the two points as in the L^2norm (Section 4.1.2.1).

2. **Correlation:** Normalized correlation measures the rate of change of one vector with respect to the other using Pearson's r statistic (Section 4.1.2.2).

3. **Covariance:** Measures the angle between two unit feature vectors using the inner product (Section 4.1.2.3).

In addition to these measures, a measure based on histogram intersection was originally considered (Swain 1990). The measure, while proven to be effective for object indexing, is absurd for pose recovery in this domain. Once the background is added to the prediction, translations of the object in the image plane will not change the histogram of the predicted image. In addition, the spatial correspondence of the pixels has been discarded, which is essential for pose recovery.

4.1.2.1 L^2norm Measure

The L^2norm computes the Euclidean distance between two feature vectors (Duda and Hart 1973):

$$\mathcal{E}_{L^2norm}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \frac{1}{K}\sqrt{\sum_{j=1}^{N}\left(\vec{f}_{sensor}[j] - \vec{f}_{predict}[j]\right)^2} \tag{4.35}$$

where N is the length of each feature vector, and K is a scalar constant used to normalize the distance to the range $[0, 1]$. For instance, with raw 8-bit RGB features, the maximum distance between any two pixel values is 255. Therefore, K^{RGB} is:

$$K^{RGB} = \sqrt{\sum_{j=1}^{N}\left(\frac{1}{255}\right)^2} = \sqrt{\frac{N}{255^2}} \tag{4.36}$$

While the theoretical maximum of the color range is easily determined, that maximum is not usually observed in CCD sensor data. Therefore, K^{RGB} is a poor normalization constant since it will produce error values in a much smaller range than $[0, 1]$. However, K is a linear

scale factor which will preserve the ranking of solutions: if scene configuration $\mathcal{S}^A$ is larger in error value than $\mathcal{S}^B$ before scaling, than it will also be larger after scaling (provided they are normalized by the same K).

Crowley *et al.* have observed that the L^2norm is the optimal function for comparing two signals when the difference between them is due to Gaussian noise (Crowley and Martin 1995). It may be the case that when the true object pose parameters are used the difference between the sensor and prediction is Gaussian. However, as the configuration parameters deviate from the ideal, it is unlikely that the noise function will remain Gaussian.

4.1.2.2 Correlation Measure

Correlation is a common technique for comparing two signals in digital signal processing (DSP). The technique can be viewed as converting each signal to a standard randomized variable (Spiegel 1996). Instead of each signal having a different mean and standard deviation, they are both re-scaled to a $[0,1]$ distribution. The general form for Pearson's r-correlation is:

$$r_{xy} = \frac{1}{N-1}\sum_{j=1}^{N}\frac{(x_j-\overline{x})}{\sigma_x}\frac{(y_j-\overline{y})}{\sigma_y} \tag{4.37}$$

This statistic is in the range $[-1,1]$ where -1 represents a perfect negative correlation and 1 a perfect positive correlation. Correlation is easily converted into an error term by normalizing to the range $[0,1]$:

$$\mathcal{E}_{corr}\left(\vec{f}_{sensor},\vec{f}_{predict}\right) = \frac{1}{2}-\frac{1}{2}r'_{xy} \tag{4.38}$$

where:

$$r'_{xy} = \frac{1}{N-1}\sum_{j=1}^{N}\left(\frac{\vec{f}_{sensor}[j]-\overline{\vec{f}_{sensor}}}{\sigma_{\vec{f}_{sensor}}}\right)\left(\frac{\vec{f}_{predict}[j]-\overline{\vec{f}_{predict}}}{\sigma_{\vec{f}_{predict}}}\right) \tag{4.39}$$

One non-intuitive piece of information must be provided about computing the means and standard deviations of the feature vectors for the cases where multiple bands of data are used. In the case of RGB, one might expect the mean to be computed for each band separately. Problems can result if this is done. Take a region in an image which is mostly white and compare that against a mostly red region. Using the mean for each band, the white values are all shifted towards zero when normalized. Similarly, for the red region,

each band is independently shifted towards zero. Taking the mean over all three bands prevents these vastly different signals in color space from appearing similar in correlation space.

4.1.2.3 Normalized Covariance Measure

The final similarity measure examined is normalized covariance. Given two feature vectors, covariance measures the inner product between the two:

$$\mathcal{E}_{cov}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \frac{1}{2} - \frac{1}{2} \cdot \frac{\vec{f}_{sensor}}{\left|\vec{f}_{sensor}\right|} \cdot \frac{\vec{f}_{predict}}{\left|\vec{f}_{predict}\right|} \tag{4.40}$$

Intuitively, the measure computes the inverse cosine of the angle formed between the two unit feature vectors. The normalization is done to shift the dot product which is bounded at $[-1, 1]$ (where 1 is *good*) to the range $[0, 1]$ (where now 0 is *good*).

4.1.3 A Combined Measure

Throughout the discussion on image features, we have hinted that color and edge features measure distinct events in an image: color measures are based on the internal properties of a face, and edges are based upon the face boundaries. For instance, observe the different objects shown in Figure 4.12a. Using an edge based error function, the match shown in Figure 4.12b would receive a good score. Similarly using a color based measure, Figure 4.12c would also match well. While color and edge based measures have different strengths and weaknesses, they are both important.

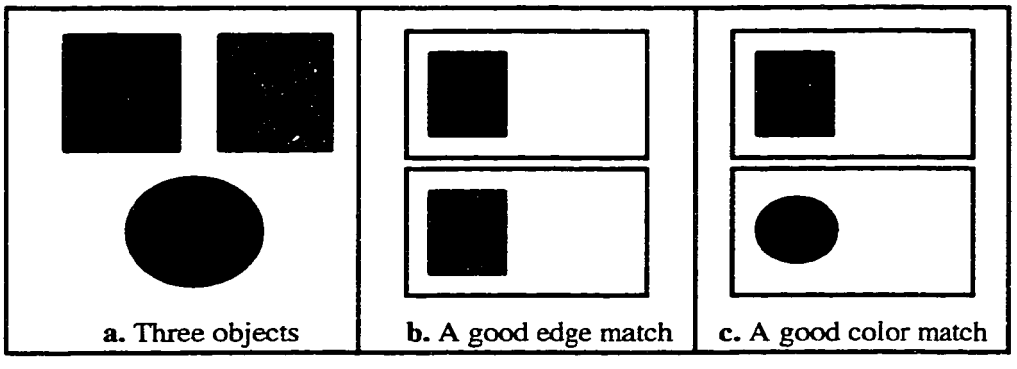

Figure 4.12: A simple matching example.

Therefore, choosing one feature over the other may not provide the best error function for RMR. Perhaps an error measure based on multiple features would provide a better measure of match quality than any single error function based solely on one feature type. The issue then becomes how to combine error functions based on different features in a reasonably intelligent and principled manner. Since the information measured by color and edge features is independent, the easiest combination method is to just add any two error functions:

$$\mathcal{E}(\mathcal{S}) = \mathcal{E}^{color}(\mathcal{S}) + \mathcal{E}^{edge}(\mathcal{S}) \tag{4.41}$$

While 552 ($24 \times 24 - 24$) possible error functions could be constructed, only a subset were used for the evaluation in the next section. For comparison purposes, three new error functions are introduced:

$$\mathcal{E}^{COMB}_{L^2norm}(\mathcal{S}) = \mathcal{E}^{NRGB}_{L^2norm}(\mathcal{S}) + \mathcal{E}^{EDGEM}_{L^2norm}(\mathcal{S}) \tag{4.42}$$

$$\mathcal{E}^{COMB}_{Correlation}(\mathcal{S}) = \mathcal{E}^{NRGB}_{Correlation}(\mathcal{S}) + \mathcal{E}^{EDGEM}_{Correlation}(\mathcal{S}) \tag{4.43}$$

$$\mathcal{E}^{COMB}_{Covariance}(\mathcal{S}) = \mathcal{E}^{NRGB}_{Covariance}(\mathcal{S}) + \mathcal{E}^{EDGEM}_{Covariance}(\mathcal{S}) \tag{4.44}$$

We chose these three functions out of the 552 possible based on the $\vec{G}$ histograms presented for the $NRGB$ and $EDGEM$ features. Each of these histograms had very small standard deviations, thus indicating a good match between predicted and observed. Future work should examine more elaborate error functions based on a wider range of feature combinations.

4.1.4 The Effects of Using the Background in the Comparison

Chapter 3 introduced the uniform background model used during rendering. Incorporating the background model into the measurement phase allows all error function evaluations to be made over feature vectors of fixed length: issues of normalization are neglected. Normalization is an issue seldom discussed in the object recognition literature, partly because of the focus on single instance object recognition. When dealing with a single object in a scene, normalization is not an issue since the obvious heuristic is to always match as much of the object as possible.

Had the background not been incorporated into the error function, a normalization mechanism would be required. One possible normalization technique is based on Bayesian statistics (Wells 1993). Intuitively, Bayes rule allows the error measure to be specified in terms of the likelihood that each model feature is found in the image given some expectation that it will appear. This expectation is referred to as the *a priori* probability. A Bayesian error measure takes the form:

$$P(\alpha_i|D) = \frac{P(D|\alpha_i)P(\alpha_i)}{P(D)} \tag{4.45}$$

where $P(\alpha_i|D)$ is read as the probability of feature α_i given the data D. This is equal to the probability of the data given the feature $P(D|\alpha_i)$ weighted by the expectation $P(\alpha_i)$ and the data $P(D)$. $P(D)$ is used to normalize the ratio and can be expanded using the law of total probabilities:

$$P(D) = \sum_{j=1}^{n} P(D|\alpha_j)P(\alpha_j) \tag{4.46}$$

The likelihood of the model based on the likelihood of the features is estimated by:

$$P(M|D) = \prod_{i=1}^{|M|} P(\alpha_i|D) \tag{4.47}$$

If the cardinality[3] of the dataset remains constant, then $P(D)$ is constant across all matches and can be discarded. However, if the number of features in the model varies, the success or failure of the comparison will depend on $P(D)$. The reason is simple: without the denominator, adding more features will always cause the numerator to decrease (due to multiplying together many numbers less than 1). The role of the denominator is to prevent such an imbalance.

How accurately and reliably $P(D)$ can be computed across all models for any given dataset defines the stability of the error measure. Such computations can be difficult when the individual measurements for a given feature are not independent of other features in the same model. Future work should consider ways to match over only portions of the image where objects are predicted. However, as this discussion suggests, doing so successfully will

[3]Cardinality represents the count of the number of elements in the set

require considerable care. In particular, image pixels are not independent events so a simply approach for estimating $P(D)$ will likely fail.

4.2 Comparing Error Functions

During the definition of the feature vectors, three factors were hypothesized as introducing discrepancy between predicted and observed images:

- **Quantization:** image quantization results from the discrete sampling of the continuous image irradiance function (light from the scene entering the sensor). For every digital signal, no matter how highly sampled, there exists a Nyquist rate such that events above the Nyquist rate (i.e., higher frequency) cannot be reliably detected and reconstructed. Since there always exists a Nyquist rate, unless the signal is low-pass filtered before sampling artifacts will be introduced.

- **Reflectance:** for most domains, quantization will not be the dominant factor affecting object appearance. In fact, changes in the direction of a light source with respect to an object face will account for most of the discrepancy between the rendered and sensor images. How such a change in surface illumination influences the perceived color of the object surface is dictated by three properties: 1) the diffuse properties of the surface material, 2) the specular properties of the surface material, 3) the surface orientation with respect to the light source and the camera. Diffuse material properties accurately model the effects of lighting on Lambertian, or matte surfaces. Specular surface properties account for bright patches on these shiny objects.

- **Reflection:** besides reflectance, inter-object reflection will change the appearance of an object in a scene. Reflection is a difficult event to model and incorporate into the reasoning process. Not only are all of the material properties required, but additional reflection parameters are also needed. These reflection parameters influence how light is re-emitted into the scene from each object surface. Accounting for reflections requires exact knowledge of scene geometry including a better representation of the background (light will reflect of the background surfaces and onto the objects). Fur-

thermore, ray tracing is needed to render the scene at a considerable computational cost.

It stands to reason that error functions that are less sensitive to these three factors should provide a more robust measure of match quality. Therefore, the expected performance of each error function is intuitively compared based on these factors. The following comparison is partially derived from 1) an initial pilot study used to compare just $\mathcal{E}^{NRGB}_{Correlation}(\mathcal{S})$ to $\mathcal{E}^{EDGEM}_{Correlation}(\mathcal{S})$ and 2) our intuition and domain knowledge. The first part of the comparison addresses the implemented features, and the second part addresses the similarity measures:

- **Both HS and $NRGB$ features compensate for changes in intensity either through normalization or representation.** Unless the color of the light source changes between when the texture maps are obtained and the new scenes are captured, observed changes in diffuse illumination should only be in terms of brightness. Therefore, error functions based on HS and $NRGB$ should outperform error functions based on other color features which do not normalize out intensity (RGB and $GREY$).

- **Edge based features normalize out changes in intensity along geometric object boundaries.** Edge features estimate local differences between neighboring pixels: the absolute value of neighboring pixel intensities is not important, only the relative difference. Therefore, changes due to diffuse illumination will be removed along boundaries between object surfaces of different orientation. The $EDGEI$ and $EDGERGB$ features should achieve comparable performance since they both normalize for intensity changes in the same manner. An interesting question is whether the orientation provided by the gradient computation is accurate. If orientation cannot be reliably computed, the $EDGEM$ features may be more robust.

- **Edge features measure local image properties, whereas color features are based upon regions in the image.** Edges have a smaller region of influence in which the measurements are meaningful. Conversely, the color measure should have a larger region of influence. This intuition can be explained with a simple example: imagine a

block sitting on a table with no other objects around. When the scene parameters are close to the true settings, most of the object edges will align with edges in the sensor image (similarly, most of the object faces will also align). If the block is slid to the left half the object width, fifty percent of the object faces remain overlapping but only twenty five percent of the object edges still align. Therefore, when a hypothesized scene configuration is close to the true configuration, the edges should provide a substantial amount of information for search. As the scene configuration diverges from the true, color will provide more information.

In addition to the intuitive comparison of features, performance predictions about the combination mechanisms are given. The analysis of combination mechanisms is based on several observations about the similarity of the equations used by the comparison functions. Since the cardinality of the feature vectors is fixed, the normalization constants (used to bound the error to the range $[0, 1]$) are no longer necessary. Without normalization, the L^2norm measure can be re-written as:

$$\mathcal{E}_{L^2norm}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \sqrt{\sum_{j=1}^{N}\left(\vec{f}_{sensor}[j] - \vec{f}_{predict}[j]\right)^2} \tag{4.48}$$

Note that the square root function does not change the shape of the error surface, only the magnitude of the error values: the lowest value on the error surface will be the same with or without the square root:

$$\mathcal{E}_{L^2norm}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \sum_{j=1}^{N}\left(\vec{f}_{sensor}[j] - \vec{f}_{predict}[j]\right)^2 \tag{4.49}$$

which can be expanded to:

$$\mathcal{E}_{L^2norm}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \sum_{j=1}^{N}\left(\vec{f}_{sensor}[j]^2 + \vec{f}_{predict}[j]^2 - 2\vec{f}_{sensor}[j]\vec{f}_{predict}[j]\right) \tag{4.50}$$

and finally:

$$\mathcal{E}_{L^2norm}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \sum_{j=1}^{N}\vec{f}_{sensor}[j]^2 + \sum_{j=1}^{N}\vec{f}_{predict}[j]^2 - 2\sum_{j=1}^{N}\vec{f}_{sensor}[j]\vec{f}_{predict}[j] \tag{4.51}$$

Since the $\vec{f}_{sensor}[j]^2$ term is constant for all error evaluations of a specific image, it will also not affect the shape of the error surface and can be discarded. The next term, $\vec{f}_{predict}[j]^2$,

is the total picture energy of the prediction (Duda and Hart 1973). Assuming there will be very little variance in the picture energy across predictions, then the function will be minimized when $\sum_{j=1}^{N} \vec{f}_{sensor}[j]\vec{f}_{predict}[j]$ is maximized (Duda and Hart 1973) (the constant 2 can also be discarded as it is a linear scale factor). Notice that when the normalization values are neglected from the covariance measure (Section 4.1.2.3):

$$\mathcal{E}_{covariance}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \frac{\vec{f}_{sensor}}{\left|\vec{f}_{sensor}\right|} \cdot \frac{\vec{f}_{predict}}{\left|\vec{f}_{predict}\right|} \tag{4.52}$$

the covariance will be minimized when the dot product between the predicted and observed feature vectors are maximized:

$$\mathcal{E}_{covariance}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \frac{\sum_{j=1}^{N} \vec{f}_{sensor}[j]\vec{f}_{predict}[j]}{\left|\vec{f}_{sensor}\right|\left|\vec{f}_{predict}\right|} \tag{4.53}$$

In essence, modulo the division to make both vectors of unit length and the reflection so that the error term is minimized, the general shapes of the error surfaces should be equivalent. This implies that a minima using the $L^2 norm$ for a specific set of features and parameters should have a corresponding maxima in the covariance error function when using the same features and parameters.

A set of similar observations applies to the normalized correlation function (Section 4.1.2.2). Assuming that normalization by the standard deviation of the data is not important (they are approximately constant across predictions):

$$\mathcal{E}_{correlation}\left(\vec{f}_{sensor}, \vec{f}_{predict}\right) = \sum_{j=1}^{N} \left(\vec{f}_{sensor}[j] - \overline{\vec{f}_{sensor}}\right)\left(\vec{f}_{predict}[j] - \overline{\vec{f}_{predict}}\right) \tag{4.54}$$

When it is the case that $\overline{x} = \overline{y} = 0$, the correlation will be exactly equivalent to covariance. This zero mean case occurs when signed edge gradients are used (the edge mask weights sum to zero). For the gradient magnitude features, this is not a valid assumption. For the case of signed edges, the three error functions should have equivalent predictive power.

Another common case arises when using color based features. Often the means of each feature vector are nearly equal: $\overline{x} \approx \overline{y}$. This gray world assumption derives from the use of the uniform background model to predict non-object pixels. Since the background occupies more of the image than the objects in general and the background is the mean of the sensor

image, the means of both images are approximately equal. In this case, the means can also be neglected from the measure.

Thus under a variety of conditions, the error surface shapes for correlation, covariance and Euclidean distance will be similar. This does not imply that they are equivalent. For instance, when comparing two cases using correlation, normalizing by the standard deviation may be essential: search should favor configurations with lower variance. Often, however, we should expect similar results.

4.2.1 An Empirical Comparison

To empirically compare the 27 combinations of 3 measures, 8 features, and 3 combined functions, the desirable error function properties discussed in the introduction to this chapter are used. Underlying these criteria is the notion that error should decrease in value as the parameters move from an arbitrary setting towards the true configuration. If an error function does not exhibit this property, iterative search will fail to locate the correct solution. Another way to view this phenomena is in terms of an error trajectory (Rana 1999). The goal is to have the optimal point, in terms of the true scene configuration, appear as the lowest point on an error surface. A step along any path leading away from that point should result in an increase in error.

Adding noise to any parameter in the true configuration should produce a configuration having a higher error: more noise equates to higher observed error. Examining collections of randomly perturbed configurations and their error values across different scenes it is possible to characterize the overall stability of the error function. Intuitively, an equivalent ranking of configurations should be produced when sorting by either distance from the true configuration or by error value.

4.2.1.1 Randomly Sampling an Error Surface

Instead of first choosing a fixed number of random points and then measuring the distance to the true configuration, a structured sampling method is used. The idea behind this sampling is to obtain groups of configurations all of similar distance from the optima. The

structured sampling can be viewed in terms of contours, or rings, which are of the same conceptual distance from the true configuration (see Figure 4.13). In order to satisfy the ranking criterion, the error values for all of the configurations within a given ring must be of similar magnitude. The distance distance of a given configuration to the true is

$$D = \frac{\sum_{j \in objects} \left| \mathcal{Q}_j^{true} \cap \mathcal{Q}_j \right|}{\sum_{j \in objects} \left| \mathcal{Q}_j^{true} \cup \mathcal{Q}_j \right|} \tag{4.55}$$

This measure was discussed in Section 3.4.1 of Chapter 3 is used. The D measure compares the ground truth configuration to an arbitrary set of pose parameters (remember $\mathcal{Q}_j$ is defined as the set of pixels given the object face label in a geometry rendering based on the current pose). D counts the number of overlapping object face pixels given two configurations. The measure is adapted to indicate the distance of an arbitrary configuration from the true:

$$\left| \mathcal{S} - \mathcal{S}^{true} \right| = 1 - D \tag{4.56}$$

where $\left| \mathcal{S} - \mathcal{S}^{true} \right|$ will be 0 when each face in each object perfectly overlaps the correct region in the image, and 1 when no face pixels overlap.

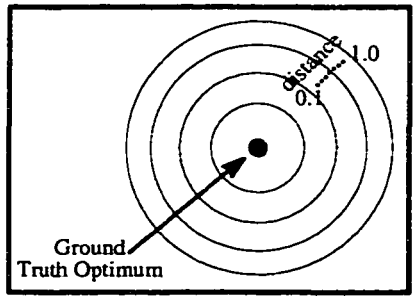

Figure 4.13: Distance from the optimum is visualized in terms of contours. Each sampled point on the contour is the same distance from the ground truth parameterization.

It may seem at first inappropriate to use a $2D$ image measure to evaluate how well $3D$ pose is recovered. However, in this chapter, the goal is not to assess pose recovery but rather characterize the underlying error surface topology. A more traditional $3D$ pose analysis will take place in Chapter 6 when the overall performance of our RMR implementation is evaluated (the recovery of each pose parameter is considered as opposed to one single

measure of distance). Here, the $2D$ measure was selected over several other measures which were considered:

- We considered using the Euclidean distance between two configurations in parameter space. This measure is not practical since the pose parameters are of different types and units (rotation is in angles, translation is in meters). Any single distance value computed would be heavily biased towards the distance between rotations.

- We considered using the σ from the random number generator: as σ increases, so does the amount of perturbation. Again, this measure is not practical due to the difference in pose parameter units (each parameter type has its own σ).

- We considered using the average $3D$ Euclidean distance between each vertex on each model in scene coordinates as it is first projected using the true configuration and then projected using the perturbed configuration. This measure seems plausible but it is very scene specific and does not allow the combination of information across scenes. For instance, imagine two scenes containing the same object. In one scene, the object is close to the camera, and in the other, the object is far from the camera. Translating this object a small amount will have a drastic change in the appearance of the scene when the object is close to the camera. Conversely, a small translation will have little effect when the object is far from the camera. While both of these perturbations will be measured as close to the true configuration, wildly different error values will be recorded. The face feature overlap prevents such problems: the pixel measure shows the far object as having less change than the closer object.

In light of these issues, D is the appropriate measure. The measure remains stable across scenes since it compares normalized counts of pixels: 50 percent of overlapping face pixels has the exact same meaning in all scenes. Similarly, both D and the error function features compare pixels.

In this empirical comparison, 250 configurations were examined for each of 20 scenes ($5,000$ configurations total). 50 different rings were defined, with five configurations per ring. The configurations in the first ring have nearly 100 percent face overlap between

object features rendered by the true configuration and those rendered with the perturbed configuration. Each successive ring represents a two percent decrease in face overlap. Since the $1st$ ring represents perfect feature alignment, the $50th$ ring represents the case where none of the object face features are correctly aligned with the data.

The new perturbations are generated by adding noise to each parameter in the true configuration. The appropriate ring into which the configuration falls is computed using D. If that ring already has five samples, the configuration is discarded. If the ring has less than five, the error evaluations for all 27 error functions are recorded. The process continues until all of the rings have been filled.

4.2.1.2 Does the Error Function Indicate the True Solution?

Perhaps the most important attribute of an error function is whether it ranks the true solution as better than other solutions. If:

$$\forall b: \quad \mathcal{E}\left(\mathcal{S}^{true}\right) < \mathcal{E}\left(\mathcal{S}^{b}\right) \quad \text{where} \quad \mathcal{S}^{true} \neq \mathcal{S}^{b} \tag{4.57}$$

then the error function can differentiate the true state from an arbitrary configuration. To measure how often this is satisfied for the 20 example problems, two counts are made. The first count c^1_{image} is the number of times the true configuration has a lower error than all of the corresponding perturbations for a single image:

$$c^1_{image} = \sum_{b=1}^{N} \begin{cases} 1 & \mathcal{E}\left(\mathcal{S}^{true}\right) < \mathcal{E}\left(\mathcal{S}^{b}\right) \\ 0 & \text{otherwise} \end{cases} \tag{4.58}$$

where N represents the number of perturbations (250). The next count is the sum of c^1_{image} across images which is then normalized by the total number of configurations examined:

$$C^1 = \frac{1}{N \cdot M} \sum_{i=1}^{M} c^1_{image} \tag{4.59}$$

where M is the number of images (20). The percentage represents the number of times $\mathcal{E}\left(\mathcal{S}^{true}\right)$ is less than all other perturbations. Table 4.1 shows C^1 for each of the 27 error functions. From the table, several observations are obvious:

1. The HUE feature combined with the *Correlation* mechanism poorly identifies the true configuration in both domains (shown in bold). As a side note, the HUE features

have an important implementation detail not previously discussed: HUE values are circular so wrap around must be handled (360 is next to 0 in HUE space). For *Euclidean* comparisons, the wrap around is handled by doing a simple difference test. When the predicted and sensor HUE for each pixel are being compared, the signed difference between them is computed. If that difference is greater than either 180° or less than $-180°$, the predicted pixel is shifted by 360°. For *Correlation*, the means for the predicted and sensor HUE are computed in terms of the XY points on the HS plane. The HUE for that HS mean is computed and used to compute the standard deviation (with the difference adjustment).

2. Similarly, most of the other color features (*GREY*, *NRGB*, *RGB* and *HUE*) with *Covariance* also poorly indicate the true configuration as the optimum (shown in bold).

3. All of the other error functions have comparable performance in terms of ranking the true solution with the lowest error value (most of the values are in the high 90s).

	Euclidean	Correlation	Covariance
GREY	0.96	0.96	**0.77**
RGB	0.96	0.96	**0.80**
NRGB	0.95	0.95	**0.89**
HUE	0.92	**0.83**	**0.78**
HS	0.95	0.95	0.95
EDGEI	0.96	0.96	0.95
EDGEM	0.94	0.94	0.91
EDGERGB	0.96	0.96	0.95
COMB	0.94	0.95	0.91

Table 4.1: Determining how often the true configuration has the lowest error for the twenty test problems.

4.2.1.3 Partially Ordering Configurations?

The previous criteria only dealt with $\mathcal{E}(\mathcal{S}^{true})$ and did not analyze the ranking of solutions as they deviated from the true configuration. The ability to assign configurations closer to

the optima with a lower error is an extremely important attribute for heuristic search (Jones and Forrest 1995):

$$\mathcal{E}(\mathcal{S}^a) < \mathcal{E}\left(\mathcal{S}^b\right) \Rightarrow \left|\mathcal{S}^a - \mathcal{S}^{true}\right| < \left|\mathcal{S}^b - \mathcal{S}^{true}\right| \quad \text{where} \quad \mathcal{S}^a \neq \mathcal{S}^b \tag{4.60}$$

A given list of scene configurations can be sorted in one of two ways. First, the list can be sorted in terms of the distance of each configuration to the true configuration: $|\mathcal{S}^a - \mathcal{S}^{true}|$. Second, that list can be sorted in terms of error: $\mathcal{E}(\mathcal{S}^a)$. If the error function provides a good partial ordering, the position of each configuration in both lists should be nearly the same.

For both of the sorted lists, two numbers between 0 and 250 can be assigned to each configuration (there were 250 randomly sampled configurations per image). These numbers represent the position of the configuration in either list. Figures 4.14 through 4.16 show where each configuration was placed in both lists. A good ranking of configurations by an error function produces a point cloud close to the diagonal. For instance, $\mathcal{E}^{NRGB}_{correlation}$ in Figure 4.15 produces points fairly near the diagonal as compared to $\mathcal{E}^{EDGERGB}_{L^2norm}$ of Figure 4.14.

Also, the error functions based on either $\mathcal{E}^{EDGERGB}_{correlation}$ or $\mathcal{E}^{EDGEI}_{correlation}$ features have desirable behavior (points are near the diagonal) when the samples are close to the true configuration (meaning near zero). Samples farther from the true configuration appear completely random. This behavior can be explained by the narrow radius of influence for which the edge based measurements make sense. This drop-off may be prevented by convolving the image with a wider edge mask. One could easily envision scale space techniques (Witkin 1983) being applied to measure edge gradient over different size areas.

In order to statistically compare the information shown in Figures 4.14 through 4.16, the two rankings (distance and error) of each configuration are converted to a single number. This number is computed by taking the absolute value of the difference between the two rankings: for each point in Figures 4.14 through 4.16, the y value (representing the position as sorted by the error) is subtracted from the x value (representing the position as sorted by the distance). Thus, for no difference in ranking, the metric would be 0. The distributions formed by these differences over all of the perturbed configurations can be compared using

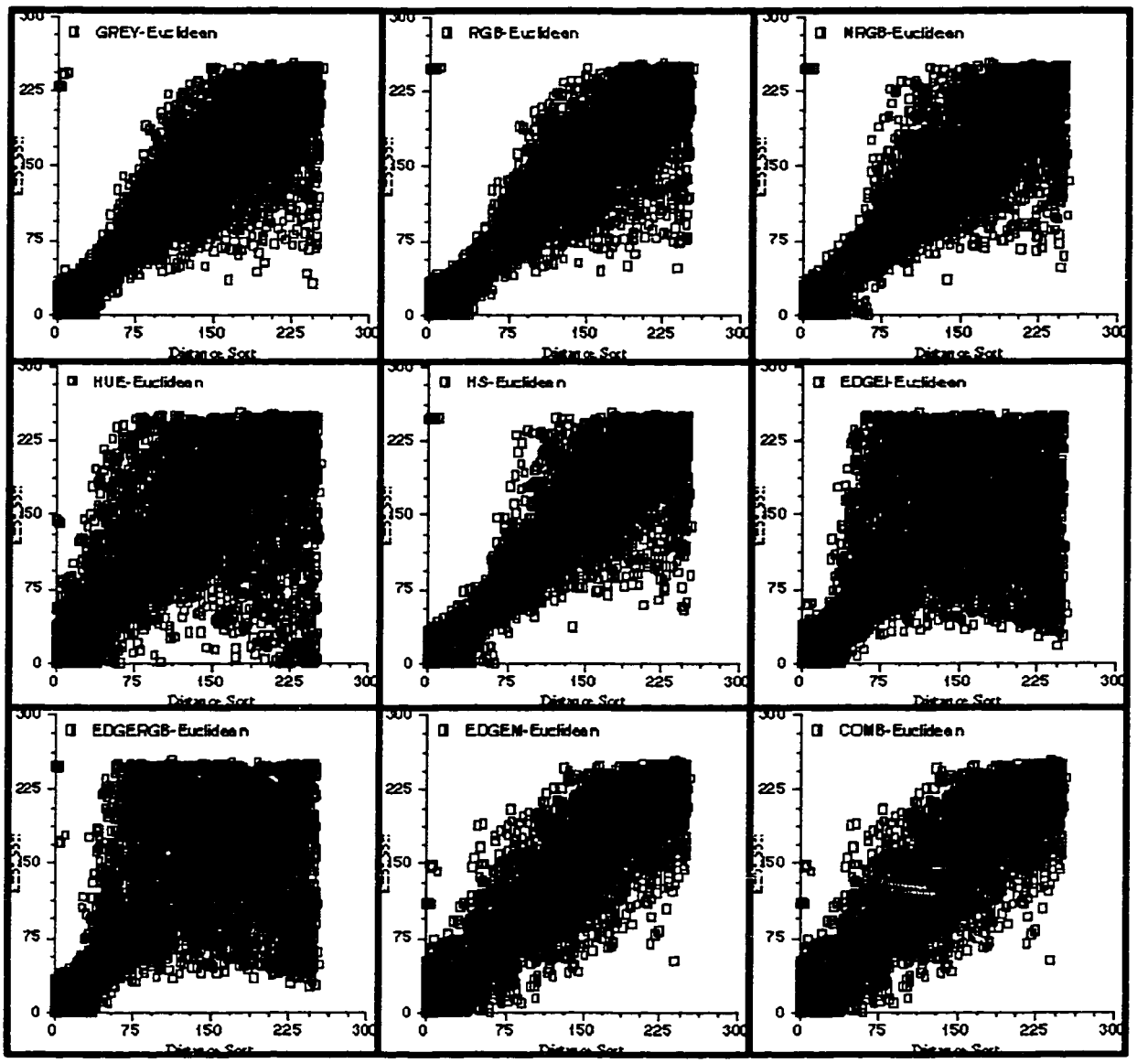

Figure 4.14: Visualizing the ranking supplied by the error functions based upon Euclidean ($L^2 norm$) combination.

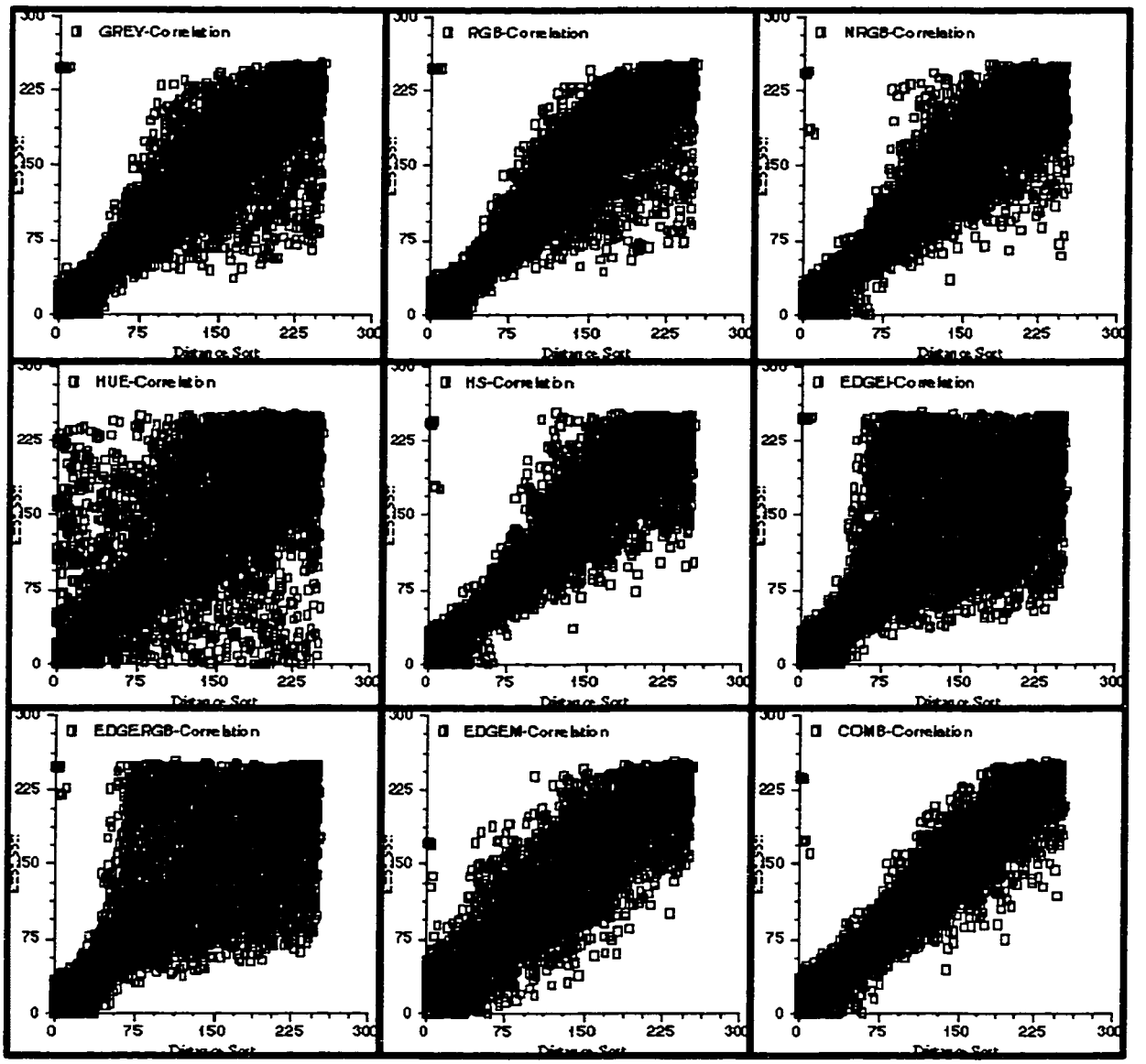

Figure 4.15: Visualizing the ranking supplied by the error functions based upon Correlation combination.

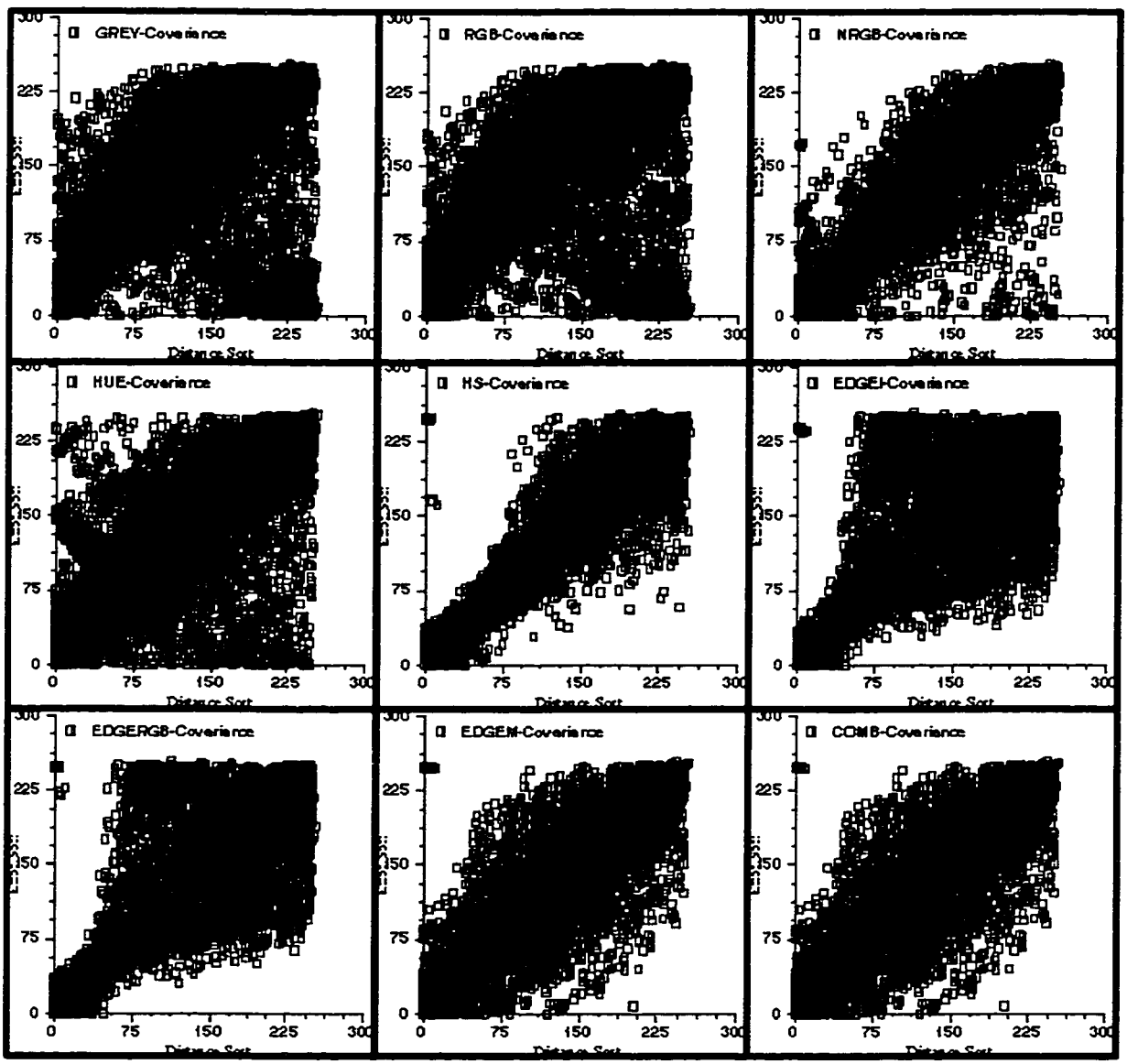

Figure 4.16: Visualizing the ranking supplied by the error functions based upon Covariance combination.

the two way ANOVA with the feature type and combination mechanism as the independent variables.

The two way ANOVA test[4] is used to determine if there is a relationship between the independent variables (feature type and combination mechanisms) and the dependent variable (ranking). Table 4.2 shows the ANOVA results on all the distributions for all of the error functions. The table can be interpreted as follows: the between features category represents the test across feature types and has DOF equal to one minus the number of features. The between combination mechanisms represents the test across the three different similarity techniques. The interaction category represents the variation across all of the evaluations for all error functions. The F distribution is used to determine if there is a significant difference between any of the distributions. If F is large enough with respect to the degrees of freedom, then the distributions are not assumed to be equivalent: the distance from the ground truth is predicted by the error function evaluation.

	Degrees of Freedom	Mean Square	F
Between Features	8	1500950	1200 with 8 degrees of freedom ($p \leq 0.001$)
Between combination mechanisms	2	2403829	1922 with 2 degrees of freedom ($p \leq 0.001$)
Interaction	16	560618	448 with 16 degrees of freedom ($p \leq 0.001$)
Residual or Random	136053	1250	

Table 4.2: Using the ANOVA test to compare the ranking dependence on either the combination mechanism or the features.

[4]The ANOVA test tells whether sample means of various factors are significantly different from one another and whether they interact significantly with each other. The test is usually used to compare differences between three or more means (or groups). The null hypothesis for the ANOVA is that the means of all groups are equal, and the alternative hypothesis is that they are not.

Based on the table, the ANOVA shows there is a statistically significant difference in the ranking provided by different error functions (both in terms of features and combination mechanisms). Since there is a significant interaction effect, the feature type and combination mechanism must be chosen together: the combination mechanism cannot be separated from the features being measured. The goal is now to select the error function which provides the best ranking of solutions. To find the best error function, the Kendall's Tau statistic was used to compare the ranking provided by error to the ranking based on distance.

Kendall's Tau statistic is a non-parametric test used to compare two rankings. Instead of using the numerical difference of ranks (as was done for the ANOVA), it uses only the relative ordering of ranks: higher in rank, lower in rank, or the same rank (Press, Flannery, Teukolsky, and Vetterling 1988; Sokal and Rohlf 1981). Since the relative ranking is all that is required, this statistic operates on the raw error and distance values: the ordinal scale used by the ANOVA is not needed as Kendall's Tau works with interval data.

To use the statistic, the rank given by the distance sort for each configuration is compared to the rank given by the error function. For all pairs of configurations, the relative ranking will either be correct (concordant: the same ranking by distance and error) or incorrect (discordant: a different ranking by distance and error). The counts of the number of concordant and discordant pairs is used to compute τ:

$$\tau = \frac{concordant - discordant}{\sqrt{concordant + discordant + extra_y}\sqrt{concordant + discordant + extra_x}} \quad (4.61)$$

where $extra_x$ represents a tie in rank by distance and $extra_x$ represents a tie in rank by error. The statistic is bound to the range $[-1, 1]$ where 1 represents complete agreement in ranking. Table 4.3 shows the Kendall's Tau statistic for all of the 27 different error functions. Also shown is the significance of each test (Z and p).

Figure 4.17 shows a plot of the Kendall's Tau statistic (the information shown in Table 4.3). Based on Figures 4.14 through 4.16, the ANOVA, and the Kendall's Tau statistic, several observations about error function ranking can be made:

		τ	Z	p
Euclidean	GREY	0.251	26.747	0.0001
	RGB	0.300	31.898	0.0001
	NRGB	0.619	65.901	0.0001
	EDGEI	0.176	18.732	0.0001
	EDGEM	0.263	27.961	0.0001
	EDGERGB	0.191	20.325	0.0001
	HUE	0.198	21.079	0.0001
	HS	0.634	67.497	0.0001
	COMB	0.263	27.961	0.0001
Correlation	GREY	0.501	53.369	0.0001
	RGB	0.573	60.969	0.0001
	NRGB	0.625	66.500	0.0001
	EDGEI	0.434	46.221	0.0001
	EDGEM	0.664	70.721	0.0001
	EDGERGB	0.467	49.690	0.0001
	HUE	0.331	35.244	0.0001
	HS	0.658	70.005	0.0001
	COMB	0.712	75.785	0.0001
Covariance	GREY	0.056	5.930	0.0001
	RGB	0.083	8.792	0.0001
	NRGB	0.174	18.527	0.0001
	EDGEI	0.471	50.131	0.0001
	EDGEM	0.265	28.231	0.0001
	EDGERGB	0.496	52.769	0.0001
	HUE	0.130	13.869	0.0001
	HS	0.524	55.828	0.0001
	COMB	0.265	28.231	0.0001

Table 4.3: Using the Kendall's Tau statistic to compare the ranking of configurations by the distance from ground truth to the ranking of configurations by error.

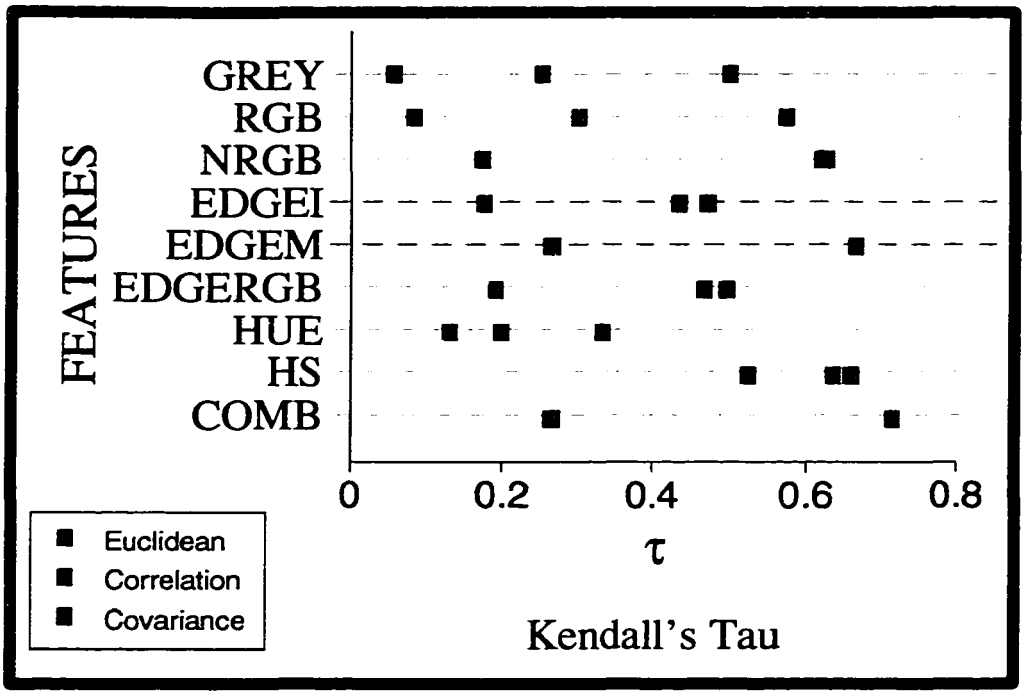

Figure 4.17: Displaying the Kendall's Tau statistic as a plot.

1. All of the tests are significant. Part of the computation for p involves the number of elements in the list. Since there are $5,000$ elements in each list, the confidence in τ is always high.

2. The *Euclidean* combination mechanism provides a poor ranking (when compared to the other measures) for all features except *NRGB* and *HS*. A plausible explanation for this effect is that the *NRGB* and *HS* features are already normalized (the *Euclidean* metric does not provide anything other than a magnitude normalization).

3. For *EDGEI* and *EDGERGB* features, performance with the *Correlation* versus *Covariance* is nearly equivalent. Since the features are computed based on signed gradients, the mean is roughly zero.

4. The *HS* and *NRGB* features with *Correlation* perform extremely well when compared to the other error functions based solely on color information (*GREY*, *RGB*, and *HUE*). The ability to normalize out changes in intensity appears to greatly benefits rankings based on these two features.

5. The error function based on *COMB* features with *Correlation* has the highest Kendall's Tau score.

4.2.1.4 Computation Time Required for an Error Evaluation

One of the key issues in determining the value of an error function is computation speed. When an error function is embedded inside an iterative search algorithm, it will need to be evaluated thousands of times before a final solution is found. The longer an evaluation takes, the more time that is required for the recognition cycle. Therefore, the computational demands of the error function dictate the feasibility of the RMR methodology.

Figure 4.18 shows a timing experiment run to compare each error function. For the 20 example images, each error function was run 10 times using the true configuration. For each of these 10 evaluations, the time to render and then compute each feature vector was recorded. Since roughly constant time is required to compare features with either *Euclidean*, *Covariance* or *Correlation*, only the time to compute two feature vectors is recorded[5].

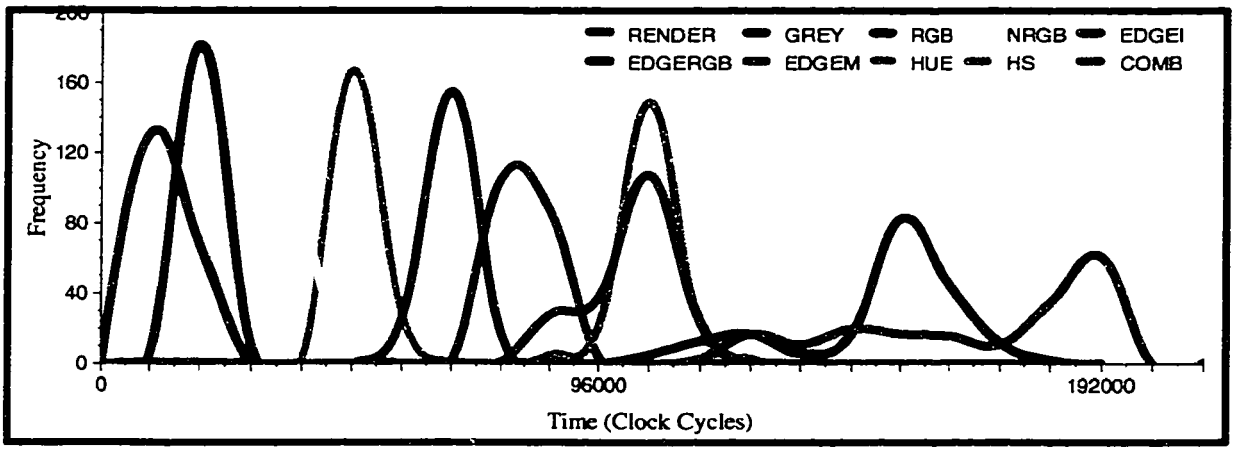

Figure 4.18: Comparing performance of the features using time. All timings were recorded on the same Pentium II/266 with 96M of RAM running Linux.

[5]Each algorithm requires one pass through the feature vectors to determine the score. Removing computationally expensive operations, such as square root, implies the computation time for comparing two vectors is similar for all of the measures examined.

Figure 4.18 shows computation times vary considerably. The fastest features to compute are the ones based directly on the raw color values: $GREY$, RGB, $NRGB$. The HUE and HS features take much longer than the other color features as the projection for each pixel from RGB to HSV is computationally demanding. In some cases, this conversion can be done in hardware or the HSV values can be received directly from the sensor. The HS takes even longer than HUE since a final sine and cosine computation must be done to convert from polar HSV to the Cartesian coordinates. Similarly, the edge based features require additional computation to compute the image gradients. $EDGERGB$ requires the most time since the feature vector is six times as long as the one for RGB. Also notice that the $COMB$ feature computation is, not surprisingly, roughly the sum of the time required to compute both $NRGB$ and $EDGEM$ features.

4.2.2 Which Error Function to Use When

The empirical comparison of error functions provides insight into which measures are robust under varying conditions. Based on that analysis, a few *rules of thumb* are proposed to decide which error functions are most appropriate in a given situation:

- When the scene configurations are fairly close to the optimum, the functions based on edge features should be favored.
- When the scene configurations are far from the optimum, the functions based on color features should be favored.
- The HUE features tend to be fairly weak for the two chosen RMR recognition domains. This does not preclude their use in other domains where they may be more appropriate.
- The $NRGB$ and HS color features provide a good ranking of solutions (the error provides insight into the configuration's distance from the optimum). Since the two features are similar in predictive power, $NRGB$ should be favored due to the reduced computation time.

- The $COMB$ mechanism provides an excellent balance of color and edge features and worked well in both blocks-world and products-world.

For the RMR implementation developed in this thesis, we are not as concerned with computation time as we are with quality of the results achieved. Since we do not want to constrain the expected distance of the solutions from the optimum, the $COMB$ measure based on:

$$\mathcal{E}^{COMB}_{correlation} = \mathcal{E}^{NRGB}_{correlation} + \mathcal{E}^{EDGEM}_{correlation} \tag{4.62}$$

is selected as the error function used in the subsequent chapters of this dissertation.

Chapter 5

Refine: Iterative Search

"The search for truth is more precious than its possession."

- Albert Einstein

The role of search is to find the optimal set of pose parameters $\mathcal{S}^*$ which minimize the error function of a given scene:

$$\forall b: \quad \mathcal{E}(\mathcal{S}^*) < \mathcal{E}\left(\mathcal{S}^b\right) \quad \text{where} \quad \mathcal{S}^* \neq \mathcal{S}^b \tag{5.1}$$

Chapter 4 formulated an error function $\mathcal{E}$ which is characterized by the following conditions:

1. The error function parameters are continuous and real valued. The number of possible scene configurations is therefore infinite.

2. An explicit analytic function for the error does not exist: no derivative of the function with respect to the pose parameters can be computed and traditional gradient descent methods, such as Newton-Raphson, are not appropriate.

3. The error surface for a given scene is not smooth and contains local minima.

4. Since pose parameters interact, search cannot optimize one parameter in isolation. For instance, solving for the depth of an object is not appropriate if the orientation is incorrect.

These characteristics imply search is difficult. Since traditional optimization techniques based upon analytic function optimization cannot be applied, heuristic techniques are examined.

Heuristic search can be defined in terms of a state space: a graph is formed where the nodes (states) represent scene configurations and the connections are transitions, or moves,

through the search space. In this context, search for the optimal solution equates to graph traversal (Cohen and Feigenbaum 1982). For Render-Match-Refine (RMR), the graph has an infinite number of nodes (since the parameters are real valued). It is also fully connected: given any two states, $\mathcal{S}^a$ and $\mathcal{S}^b$, a transition vector $\vec{\mathcal{S}}$ exists which maps one state into another:

$$\vec{\mathcal{S}} = \mathcal{S}^b - \mathcal{S}^a \tag{5.2}$$

therefore:

$$\mathcal{S}^b = \vec{\mathcal{S}} + \mathcal{S}^a \tag{5.3}$$

The error functions defined in Chapter 4 usually satisfied the following relationship between $\mathcal{S}^a$ and $\mathcal{S}^b$:

$$\mathcal{E}(\mathcal{S}^a) < \mathcal{E}\left(\mathcal{S}^b\right) \Rightarrow \left|\mathcal{S}^a - \mathcal{S}^{true}\right| < \left|\mathcal{S}^b - \mathcal{S}^{true}\right| \quad \text{where} \quad \mathcal{S}^a \neq \mathcal{S}^b \tag{5.4}$$

In words, states with a lower error are closer to the true configuration. The goal of RMR search is to find a series of transitions which lower the scene error and thereby move configurations towards the optimum. The objects are thus aligned with the imagery. Given that the parameters are real valued and the nodes in the search graph are fully connected, a path exists from any configuration to the optimal state:

$$\mathcal{S}^a + \vec{\mathcal{S}} = \mathcal{S}^* \tag{5.5}$$

where vector addition is used to combine transitions:

$$\vec{\mathcal{S}} = \vec{\mathcal{S}^1} + \vec{\mathcal{S}^2} + \ldots + \vec{\mathcal{S}^n} \tag{5.6}$$

Recall from the previous chapter that if the configuration in the graph with the lowest error $\mathcal{S}^*$ is found, that configuration is most likely the desired solution: $\mathcal{S}^* \approx \mathcal{S}^{true}$.

Using local forms of heuristic search, graph traversal is reduced to exploring the immediate neighbors and finding slightly better solutions: $\vec{\mathcal{S}}$ is iteratively constructed from a series of transitions. Eventually, the algorithm will converge to either the optimal configuration or a locally good solution (one in which no set of small transitions lowers the error).

Five different search algorithms are examined consisting of three essentially local methods (Simplex search, Hill Climbing, and Mutation based search), one form of global search

(Evolution Strategies) and a fifth method that combines local and global techniques. Each algorithm has been molded into a similar framework where a collection of initial starting configurations are provided. Each algorithm is then allowed to explore the space of possible configurations. After a fixed time, the best solution found by each algorithm is returned. This uniformity allows the search algorithms to be compared empirically.

5.0.3 Desirable Properties of a Search Algorithm

Before an empirical comparison can take place, a set of criteria for evaluating search performance must be defined. These criteria are cast in terms of a Binomial process (Beveridge 1993). For any given set of initial conditions, search will either locate a globally optimal solution (succeed) or it will find a local sub-optimal solution (fail). If the probability of success p can be estimated, then the Binomial probability density function (pdf) allows an estimate of the chance of seeing k successes in n trials:

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{5.7}$$

Since the true configuration only needs to be visited once in a series of trials, the value of $P(k \geq 1)$ (the likelihood of seeing at least one success in a series of trials) can be computed based on the cumulative density function of the Binomial equation:

$$P(k \geq 1) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{5.8}$$

Since the function is discrete:

$$P(k \geq 1) = 1 - P(k = 0) \tag{5.9}$$

and:

$$P(k = 0) = (1-p)^n \tag{5.10}$$

A nice property of viewing search trials in this manner is that the total number of trials to run can be estimated given knowledge of p. Table 5.1 shows the relationship of p and n for $P(k \geq 1)$ where n is the number of trials, and p is the likelihood of converging to the correct solution. Assuming that each trial is independent, even with a low probability of locating the optimum on a single trial, the chance of finding the true configuration can always be

increased by running more trials[1]. Based on this Binomial model of search, three criteria for evaluating the performance of any given search algorithm are:

		n									
		10	20	30	40	50	60	70	80	90	100
p	0.01	0.09	0.18	0.26	0.33	0.39	0.45	0.51	0.55	0.59	0.63
	0.05	0.40	0.64	0.78	0.87	0.92	0.95	0.97	0.98	0.99	0.99
	0.1	0.65	0.87	0.95	0.98	0.99	0.99	~1.00	~1.00	~1.00	~100
	0.2	0.89	0.98	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00
	0.3	0.97	0.99	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00
	0.4	0.99	0.99	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00
	0.5 to 1.0	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00	~1.00

Table 5.1: The probability of observing at least one success in n trials based on the likelihood of success, p.

1. **A search algorithm should maximize p:** A success can be defined in terms of a distance from the converged configuration to the true configuration. If:

$$\left|\mathcal{S}^a - \mathcal{S}^{true}\right| \leq \epsilon \tag{5.11}$$

then $\mathcal{S}^a$ is a success. Once a value of ϵ has been chosen, p can be estimated from a series of trials. A desirable search algorithm should have a high value of p, thus requiring fewer trials to find the true configuration. Any non-zero estimate of p implies the search algorithm will eventually find the true configuration; however, the number of trials required may be prohibitive.

2. **A search algorithm should maximize solution quality:** the binary label of success or failure does not capture how close each configuration labeled as success is to the true configuration. When two different search algorithms produce two different states, $\mathcal{S}^a$ and $\mathcal{S}^b$, for the same starting point, and:

$$\left|\mathcal{S}^a - \mathcal{S}^{true}\right| < \left|\mathcal{S}^b - \mathcal{S}^{true}\right| \tag{5.12}$$

[1]Assuming the random trials are independent is not always correct. However, it is usually a good idea to bootstrap each search trial with diverse starting points to reduce the portions of the error surface redundantly searched.

then the algorithm which produced $\mathcal{S}^a$ did better on that trial. Average algorithm performance can be viewed in terms of the mean distance from the true for a given set of successful trials:

$$\mu_{alg} = \frac{1}{n}\sum_{a=1}^{n}\left|\mathcal{S}^a - \mathcal{S}^{true}\right| \tag{5.13}$$

where n is the number of trials. To compare performance for any two algorithms, statistical tests are used to accept or reject the null hypothesis that the performance of the algorithms is equal.

3. **A search algorithm should minimize the amount of work required to obtain a solution:** The first criterion stated that: for two search algorithms, the one with the higher p is preferred. However, no estimate of the amount of work required to achieve that solution was specified. Therefore, the algorithm with the higher p may be computationally intractable for real problems. A better statement is that an algorithm should minimize the work required to find the true solution.

   For this dissertation, the amount of work w for a given trial is defined as the number of error function evaluations expended before convergence. Over a series of trials, an estimate of the average amount of work $\overline{w}$ per trial is used to compute W:

$$W = \overline{w} \cdot n \tag{5.14}$$

   where W represents the total number of error evaluations needed to find the optimal solution with a given probability.

   A subtle relationship exists between p and an estimate of the total amount of work required by a given search algorithm. Using p, the total number of trials n needed to find the solution with a certain probability can be determined (from Table 5.1). Since n is based upon p, maximizing p will lower the required number of trials. Simultaneously, minimizing $\overline{w}$ will lower the total amount of work expended by the search algorithm. However, minimizing just w without consideration of p will provide inaccurate estimates of how much effort a given algorithm requires. In essence, a tradeoff exists between finding an algorithm that performs acceptably while remaining computationally tractable.

5.1 Defining Specific Search Algorithms

Given a configuration $\mathcal{S}^a$, heuristic search explores the space of possible solutions by generating a transition vector $\vec{\mathcal{S}}$:

$$\mathcal{S}' = \vec{\mathcal{S}} + \mathcal{S}^a \tag{5.15}$$

How each of the five implemented search algorithms generates these transition vectors dictates overall performance. Briefly these five algorithms are:

1. Simplex: Simplex search developed by Nelder and Mead (Section 5.1.1).
2. Hill Climbing: Local search similar to gradient descent (Section 5.1.2).
3. Evolution Strategies: An evolution inspired search proposed by Back and Schwefel (Section 5.1.3).
4. Mutation-Only: a quasi-random search (Section 5.1.4).
5. Combined: fuses Simplex and Evolution Strategies (Section 5.1.5).

These five algorithms were selected based on their performance in a set of small pilot studies previously run over a subset of the data used in the empirical evaluation for this chapter. Furthermore, several of these algorithm have already been demonstrated in the object recognition literature: Beveridge has used Hill Climbing (Beveridge 1993), Whitley has used Evolution Strategies (Whitley, Beveridge, Graves, and Mathias 1996), and Simplex has been used by both Hoff and Fua (Hoff and Sarojak 1998; Fua and Leclerc 1994).

5.1.1 Simplex Search

Nelder and Mead (Nelder and Mead 1965) first introduced the Simplex search algorithm. It uses a collection of points to explore a search space. In our domain, a single scene configuration is viewed as a point on an error surface. Two of these points, or two scene configurations, forms a line. Three configurations define a plane, and M points define a hyper-plane. Simplex search uses a specific hyper-plane called a *simplex*. In an N dimensional search space, the simplex is defined by $N + 1$ vertices. For our problem, there are 4 dimensions to optimize for each object (using the ground plane consistency constraint

discussed in Section 3.2.1 of Chapter 3). For two objects, there are 8 dimensions, and thus the simplex contains 9 points.

5.1.1.1 The Simplex Transition Vectors

Four different changes can be made to the current simplex. These moves are defined with respect to three configurations and two parameters: $\mathcal{S}^{high}$ is the configuration with the highest error, $\mathcal{S}^{low}$ is the configuration with the lowest error, $\mathcal{S}^{center}$ is the average configuration, and α and β are control parameters set to 1.5 and 0.5 respectively.

Reflection: moves the worst point $\mathcal{S}^{high}$ past the centroid and out the other side of the simplex (Figure 5.1b). The transition vector $\vec{\mathcal{S}} = 2\left(\mathcal{S}^{center} - \mathcal{S}^{high}\right)$ is added to $\mathcal{S}^{high}$ to produce $\mathcal{S}'$.

Expansion: moves the worst point away from the centroid along the vector formed between the centroid and the worst point (Figure 5.1c). The transition vector $\vec{\mathcal{S}} = \alpha\left(\mathcal{S}^{high} - \mathcal{S}^{center}\right)$ is added to $\mathcal{S}^{high}$ to produce $\mathcal{S}'$.

Contraction: is the exact opposite of expansion. The worst point is pulled towards the centroid (Figure 5.1d) by adding the transition vector $\vec{\mathcal{S}} = \beta\left(\mathcal{S}^{center} - \mathcal{S}^{high}\right)$ to $\mathcal{S}^{high}$ and thereby producing $\mathcal{S}'$.

Shrinkage: contracts all points in the simplex towards the best point (Figure 5.1e). The transition vector $\vec{\mathcal{S}}^i = \beta\left(\mathcal{S}^{low} - \mathcal{S}^i\right)$ is added to all configurations $\mathcal{S}^i$ except for $\mathcal{S}^{low}$.

Various combinations of these moves are used to adjust the points on the simplex and move the entire population towards a region of the search space with lower error. The operators are applied in the following manner:

```
while (Not Converged) {
    1. Attempt to reflect the worst point.
    2. If the reflection makes an improvement, the algorithm expands the
       previously reflected point farther away from the center of the
```

```
        simplex.
    3. If a plateau is reached (reflection did not cause any
       improvement), the points are contracted in an attempt to escape
       along a dimension of the plateau.
    4. If the contraction fails to produce a better solution, shrinkage
       is applied.
}
```

Each type of move is guaranteed not to produce a degenerate simplex (one where some of the points in the simplex do not contribute to the convex hull). Since the simplex is initially non-degenerate, at no time during a search trial should the simplex ever become degenerate. Nelder and Mead have presented some convergence criteria for Simplex search based on this convex criteria (Nelder and Mead 1965). The reasoning is as follows: over time the volume of the simplex will decrease in a non-degenerate manner. Eventually all points in the simplex will become equivalent.

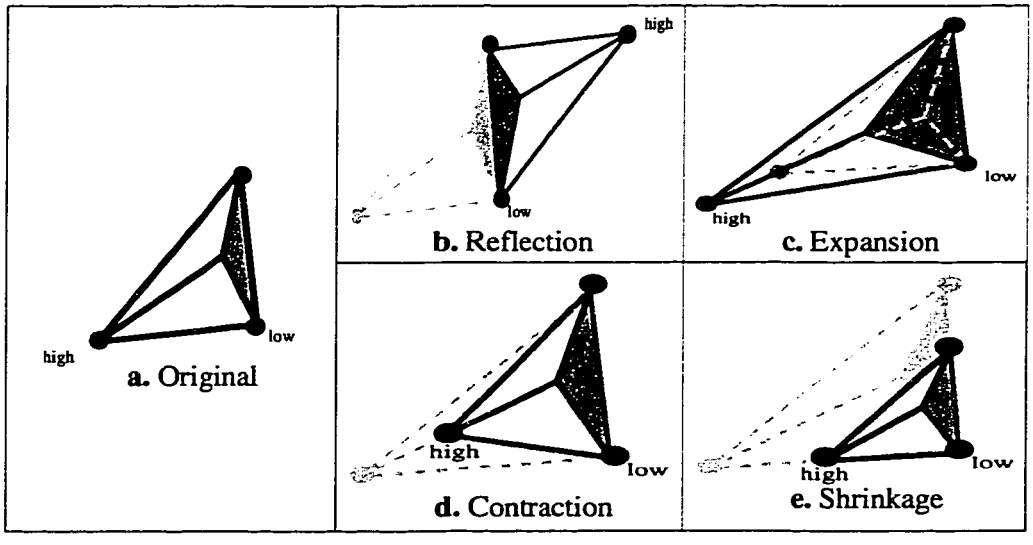

Figure 5.1: Defining the Simplex search algorithm operators. The original simplex is shown in light grey.

5.1.1.2 Determining Simplex Convergence

Determining a search termination condition in a multi-dimension optimization problem can be difficult (Press, Flannery, Teukolsky, and Vetterling 1988). One method is to test whether all the points in the simplex are within some ϵ of the centroid. This is computationally demanding ($O(n)$ where n is the number of points in the simplex) and must be computed after each iteration. Furthermore, an ϵ threshold must be determined for each of the different parameter types (rotations and translations).

Another approach examines convergence in terms of differences in error. The search convergence test is based on the difference in magnitude between the vertex in the simplex with the highest error $\mathcal{S}^{high}$ and the one with the lowest error $\mathcal{S}^{low}$ (Press, Flannery, Teukolsky, and Vetterling 1988):

$$ftol = \frac{\mathcal{E}(\mathcal{S}^{high}) - \mathcal{E}(\mathcal{S}^{low})}{\mathcal{E}(\mathcal{S}^{low}) + \mathcal{E}(\mathcal{S}^{high})} \tag{5.16}$$

where $ftol$ should represents the least significant change in error value ($ftol$ was set to 0.0001). Once all terms are within this tolerance, no further improvement can be found.

Even with this check for convergence, it is possible that the Simplex algorithm will not converge. To prevent such an occurrence, a hard limit is set on the total number of error evaluations (5,000) used by the search algorithm. Once that limit is reached, the configuration with the lowest error is returned as the best solution found. Using this method, RMR search could be viewed as an any-time algorithm: at any point in the search process, the best solution found could be returned.

5.1.2 Hill Climbing

Hill Climbing is based on the notion of greedy ascent (Kernighan and Lin 1972; Lin and Kernighan 1973) (or in this case descent, as the error function is being minimized). Given any state in the search space, a region of neighboring configurations is examined. If any of these configurations has a lower error, the current state is replaced and the process continues. However, if none of the neighbors have a lower error, the current point is returned as a local optimum. Figure 5.2 shows a simple one-dimensional search space. Given the initial starting point shown, the algorithm would converge to the local optimum. Allowing for multiple

trials, another starting point would be chosen. If the new point falls within the basin of attraction for the global optimum, the best solution is found.

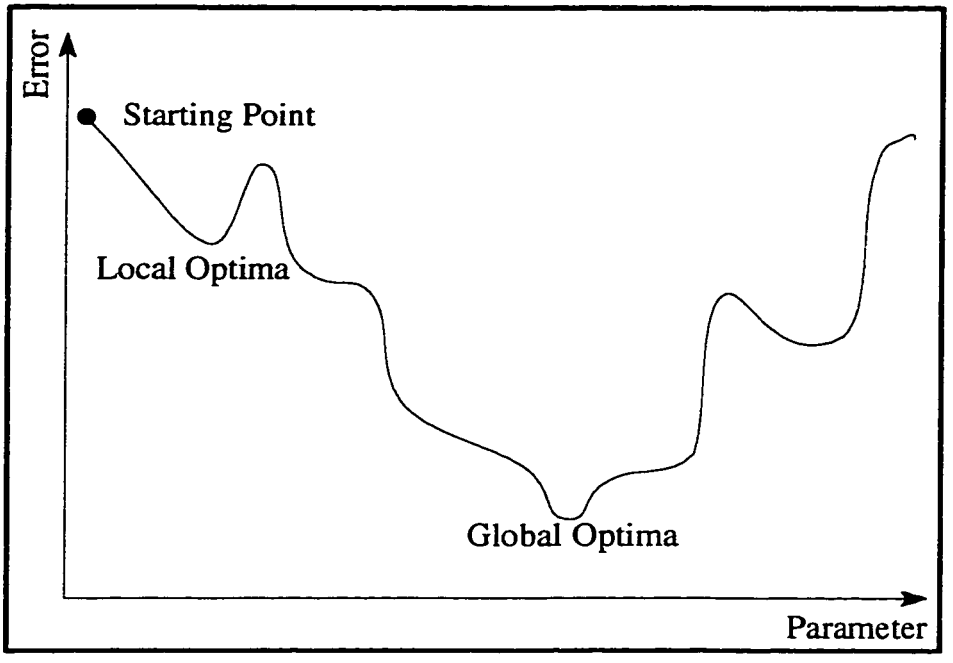

Figure 5.2: A Hill Climbing Algorithm Example.

For our implementation of Hill Climbing, a method for determining a local neighborhood is needed. For the continuous parameters used by RMR, each parameter has an infinite number of nearby configurations. For instance, imagine all of the possible unit transition vectors possible for a 12 parameter search problem (with 3 objects and 4 DOF per object). Out of all the possible heuristics which could be used to form a transition vector, we have selected one of the easiest to implement. Each parameter is evaluated independently, and greedy moves are taken each time a single parameter change results in a better state. To make the Hill Climbing algorithm directly comparable to the Simplex method, a population based Hill Climber is used. This technique can be viewed as running M Hill Climbing algorithms in parallel on each of the M points in the population. For consistency, M is set to $N+1$ where N is the number of pose parameters in the scene configuration. The explicit steps in the Hill Climbing algorithm are:

```
while (Not Converged) {
   foreach (configuration in the population) {
      foreach (parameter in that configuration) {
         1. Add a small amount to that parameter
            (2.5 degrees for rotations, and 2mm
            for translations). Record the error
            for that configuration.
         2. Subtract the same amount from the original
            parameter setting and record the error value.
         3. Determine if the added or subtracted configuration
            had a lower error than the current parameter. Update
            the parameter with the best setting.
      }
   }
}
```

Each configuration requires $2M$ evaluations of the error function for an M parameter problem: each of the M parameters requires an examination of the neighbor to the left (addition) and right (subtraction).

After Hill Climbing has completely updated each configuration, a simple check is made to see if any of the parameters were changed. If a complete pass through all of the configurations in the population failed to make any changes, the population is said to have converged. A fixed number of error evaluations ($5,000$) was also set to prevent the algorithm from continuing indefinitely.

5.1.3 Evolution Strategies

The next search technique implemented is based on a class of algorithms called Evolution Strategies (Bäck and Schwefel 1993; Bäck and Schwefel 1995). These techniques are

biologically inspired and employ terms such as population, generation, reproduction, and mutation. In the case of our RMR implementation, a population is a collection of points on the error surface. Search constructs successive generations by selecting two configurations and producing a new state. This new state is mutated so that it does not share exact components of either parent. When a new point is produced with a lower error than the worst point in the population, that worst point is replaced. Figure 5.3 shows a single generation.

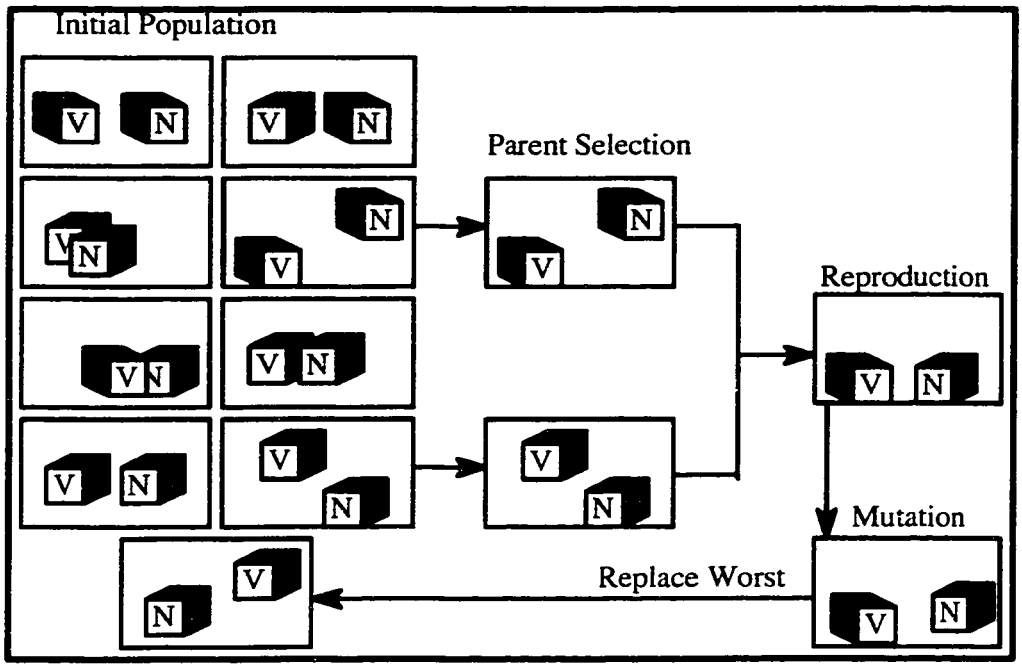

Figure 5.3: Our implementation of an Evolution Strategies Algorithm.

For consistency, the population is the same size as the number of points in the simplex (the population contains $N+1$ points for N pose parameters). Thus, when comparing search across iterations, all three algorithms described so far were given the exact same starting conditions. Many different forms of Evolution Strategies have been developed (Goldberg 1989; Holland 1975; Fogel 1993). While some variants have been successfully applied to object recognition (Whitley, Beveridge, Guerra-Salcedo, and Graves 1997; Swets, Punch, and Weng 1995), we have chosen to implement and evaluate the specific techniques discussed in (Bäck and Schwefel 1993).

5.1.3.1 Evolution Transition Vectors: Reproduction

In our implementation, the transition vectors are formed in two stages: reproduction and mutation. The reproduction operator, referred to as *panmictic*, is fully discussed in (Bäck and Schwefel 1993). This operator picks one configuration at random. Values from that configuration are copied into a new offspring using a simple constraint. At each parameter a uniform random number $[0, 1]$ is produced. If that random number is less than 0.5, the value is copied directly from the original configuration. If the number is greater than 0.5, a completely different configuration is randomly selected from the population, and the corresponding pose parameter is placed into the offspring.

Using this reproduction mechanism, offspring contain 50% of their information from a single parent and the other 50% drawn randomly from the entire population. As more and more generations are used to produce stronger and stronger offspring, search will focus on specific areas of the search space. Since only the best offspring are retained, new offspring are confined to areas of the scene where the error is lower.

5.1.3.2 Evolution Transition Vectors: Mutation

Once a new configuration is produced by reproduction, each parameter is mutated. Under certain conditions, the goals of mutation can be compared to those of simple Hill Climbing. In essence, mutation adds small random perturbations to each pose value. These perturbations allow the new configuration to sample slightly different areas outside of the current population. Mutation adds scaled Gaussian noise to each pose value, where the scaling parameters are linked to each pose parameter:

$$\vec{S} = \begin{vmatrix} \sigma_{\phi_{object1}} \cdot G(0,1) \\ \sigma_{\alpha_{object1}} \cdot G(0,1) \\ \sigma_{\beta_{object1}} \cdot G(0,1) \\ \sigma_{\tau_{object1}} \cdot G(0,1) \\ \cdots \\ \sigma_{\phi_{objectN}} \cdot G(0,1) \\ \sigma_{\alpha_{objectN}} \cdot G(0,1) \\ \sigma_{\beta_{objectN}} \cdot G(0,1) \\ \sigma_{\tau_{objectN}} \cdot G(0,1) \end{vmatrix} \tag{5.17}$$

where a mutation rate σ exists for each pose parameter and $G(0,1)$ is a mean zero standard deviation one Gaussian random variable. The added noise will allow new points to sample a neighborhood closely bounded by $[-3\sigma, +3\sigma]$.

The individual mutation parameters are also modified during recombination. When the parent configuration is chosen at random, 50 percent of the mutation parameters are copied directly to the new offspring from the parent (in exactly the same manner as the pose parameters). The other 50 percent are determined by averaging the mutation parameters of the original parent and another parent selected at random. Back and Schwefel (Bäck and Schwefel 1993) observe good performance when the σ values are reduced over time. Thus initially allowing a large amount of mutation, or local exploration, and less mutation over time. The reduction schedule decreases each σ after a generation:

$$\sigma' = \sigma \exp(\tau_p G(0,1) + \tau G(0,1)) \quad \text{where} \quad \tau = \frac{1}{\sqrt{2\sqrt{N}}} \quad \text{and} \quad \tau_p = \frac{1}{\sqrt{2N}} \tag{5.18}$$

N is the number of configurations in the entire population. To prevent mutations from becoming too large or too small in magnitude, σ is clipped by an upper and lower bound.

5.1.3.3 Survival of the Fittest

A survival of the fittest method is applied to determine if the new configuration is retained. If the error for the new configuration is lower than the worst configuration in the population (the one with the largest error), that worst configuration is replaced. This selection mechanism guarantees that the average error of the population monotonically decreases.

5.1.3.4 Determining Evolution Strategies Convergence

As subsequent generations are produced, the error values tend to converge. Once they become nearly equivalent (using the Simplex criteria for convergence discussed in Section 5.1.1.2), the best configuration in the population is returned. The same cap of 5,000 on the number of error evaluations is used to prevent the algorithm from running indefinitely.

5.1.4 Mutation-Only Search

Currently, there is no mechanism to gauge the difficulty of the RMR search problems. The previously described algorithms may be more complex than what is actually needed to recover an optimal scene configuration. An obvious *straw man* algorithm for demonstrating the complexity of the search space is Mutation-Only search. Mutation-Only search provides a baseline: if the other algorithms are worse than the random choices made by mutation, they are not based on good heuristics.

A variant of Mutation-Only search, called random sampling, was also considered. Random sampling iterative picks pose parameters completely at random and retains the best configurations visited. Random sampling is a powerful technique if a massive number of error evaluations are permitted and there is very little structure to the underlying error surface. Since neither of these two cases hold in the current evaluation, Mutation-Only search was chosen instead. In addition, Mutation-Only search has a strong relationship to Hill Climbing: the mutation equates to a variant of gradient descent where the neighborhood size is not fixed.

In our implementation of Mutation-Only search, scene configurations are explored in a stochastic fashion. If the best states found are retained, eventually the true configuration will be recovered. Mutation only search begins with the same population generated for the three previous algorithms. At each time step of each search trial, a configuration is randomly drawn from the population. Small random perturbations are made to each parameter (see Equation 5.17), and if improvements are found they are retained. A cooling schedule is not used to reduce the amount of mutation. A survival of the fittest algorithm is used (if the new configuration is better than the current worst, the worst is replaced). Convergence is tested using the Simplex criteria discussed in Section 5.1.1.2, and the same cap of $5,000$ on the number of evaluations is used to prevent the algorithm from running indefinitely.

5.1.5 A Combined Search

The final search mechanism examined is actually a combination of two of the search methods previously discussed: Simplex and Evolution Strategies. These two methods could be viewed in terms of local and global operators. The Simplex search algorithm searches in a more local fashion, exploring neighborhoods that are spatially close to the vertices in the existing simplex. The Evolution Strategies algorithm searches in a more global manner, exploring neighborhoods based on combinations of parameters in the existing population.

Fusing the local and global methods together has the potential to create an algorithm capable of escaping local minima while at the same time quickly exploring locally optimal solutions (Beveridge, Balasubrarian, and Whitley 1999; Whitley, Beveridge, Graves, and Mathias 1996). The algorithm first does 10 iterations of Evolution Strategies (reproduction and mutation) followed by 10 iterations of Simplex (exploring all combination of operators). Unfortunately, the Combined search algorithm cannot guarantee that a degenerate simplex will not be introduced.

This new algorithm is given the same initial starting configurations. Once again convergence is tested using the Simplex criteria discussed in Section 5.1.1.2, and the same cap of 5,000 on the number of evaluations is used to prevent the algorithm from running indefinitely.

5.2 Comparing Search Algorithms

Each of the five search algorithms described uses a different mechanism for producing a new scene configuration based on the current population or set of simplex points[2]. The task is to analyze Hill Climbing, Simplex search, Mutation-Only search, Combined and Evolutionary Strategies in terms of how well each explores the error landscapes created by the error function defined in Chapter 4.

---

[2]The terms population and simplex are used interchangeably since the same mechanism is used to generate both, and they represent the same conceptual hyper-plane.

Three factors are expected to influence search performance within both the products-world and blocks-world domains:

1. The object pose parameters are not independent. For instance, imagine two blocks resting on a table with a significant portion of one block occluding the other. If the initial hypothesis for one of the objects is to the left and up from the correct location, searching only left and right or only up and down will fail to locate the object: a diagonal move must be made. Also, optimizing the parameters for one object without concern for the other will lead to poor performance due to the occlusion interactions.

2. The blocks-world and products-world domains present an interesting contrast in terms of object appearance. While all of the object models in both domains are highly similar in terms of geometry (most are rectangular in shape), the texture maps are not. The blocks-world objects have large regions of uniform face color; while in products-world, the texture is much more volatile.

3. The quality of the initial configurations will greatly influence the behavior of the various search algorithms. Both how close the initial configurations in the population are to the optimum and the population homogeneity will influence the number of iterations required for a given trial and the quality of the resulting solution.

A series of pilot experiments were run on three of the products world images using only Simplex and Evolution Strategies search. For each of the three images, both search algorithms were run using three different error functions based on 1) $\mathcal{E}^{NRGB}_{Correlation}$, 2) $\mathcal{E}^{EDGEM}_{Correlation}$ and 3) their sum. These experiments indicated there was very little interaction between the error function and the search algorithm: certain search algorithms consistently out performed others when compared on the trials where the error function was the same. Therefore, we will compare search performance on the combined error function derived in Chapter 4 and not examine the effects of using different search algorithms with different error functions. We leave such analysis to future work.

Based on the above three factors and the pilot experiments, several predictions about search performance are made:

1. **Given the assumption that the parameters interact, one would expect poor performance from the Hill Climbing algorithm formulated in this dissertation.** The fixed step size method, as defined in Section 5.1.2, optimizes each parameter in series without considering combinations. The other algorithms have the ability to search over multiple dimensions simultaneously and therefore should be better at unraveling the complex interactions between parameters.

2. **The amount of texture on a given object will be related to how closely the true object configuration can be recovered.** The more texture that is observed on an object face, the more local information (constraint) provided by the error function. Therefore, search can be expected to recover the configuration of the scene better in the products-world domain than in blocks-world.

3. **The Mutation-Only and Evolution Strategies search will require numerous error evaluations before convergence.** In practical applications of evolution based algorithms, tens of thousands of error evaluations are required before search converges. Obviously, the exact number required depends on the homogeneity of the configurations in the population. Depending upon both the difficulty of the recognition problem and the diversity of the starting configurations, these two search algorithms may not converge with fewer than the maximum allowed error evaluations.

5.2.1 An Empirical Comparison

The 20 test images shown in Figures 4.1 and 4.2 of Chapter 4 are used to compare the performance of the five search algorithms. The methodology is similar to that used for comparing error functions: 1) form a set of perturbed configurations, 2) run each algorithm on the same set of configurations and 3) compare results.

Each algorithm was allowed 5,000 error evaluations over a given population of scene configurations. The number 5,000 was chosen based on our initial pilot experiments: the search algorithms tended to make good progress and typically converged within that time frame. Once the maximum was reached, the configuration with the lowest error is returned as the best solution for that trial. If the algorithm converged in less than the maximum

number of evaluations, the configuration with the lowest error in the population at the time of convergence is returned. The error function used is the weighted sum of $NRGB$ and $EDGEM$ features using correlation (see Equation 4.62 of Chapter 4). Since each search algorithm begins with the same configurations, the only difference in performance is due to how each algorithm samples the error surface.

The initial configurations are generated by perturbing each of the true parameter settings. These random perturbations were formed in exactly the same manner discussed in Section 4.2.1 of Chapter 4. For each perturbation, the $2D$ overlap between the object faces projected onto the image by first the perturbed and then the true configuration is computed. The $2D$ overlap is used to form populations consisting of configurations all having similar distances from the ground truth. Once again, the structured sampling can be viewed in terms of contours, or rings. A total of 100 trials were run per algorithm using 20 different rings with 5 populations per ring. Each ring represents a collection of configurations successively farther from the true configuration: the first ring has very little difference from the optimum, and the furthest ring should have no portion of any object overlapping the correct visible object portion in the scene.

A substantial amount of computation time was required to collect the data for this experiment. If each algorithm requires the maximum number of evaluations, then the total number of error computations is on the order of $2,500,000$ ($5,000$ evaluations, 100 trials, 5 algorithms). If each evaluation requires one second, then the total computation time is estimated at 35 hours (this disregards the time involved in generating new configurations given the current population) for a single image on a dedicated machine. Since there are 20 images to process, a total of 700 computation hours is required to generate all of the data used in this chapter. Therefore, the amount of computation limited the number of internal variations on a single algorithm which can be examined: parameter settings were fixed before the experiments could be run.

5.2.1.1 Dependent and Independent Variables

For every trial of each search algorithm, the configurations in the population before and after search are examined. The lowest error of the k points in either population is recorded. Figures 5.4 through 5.8 show the before and after 2D face inverse overlap (the $1 - D$ metric) for each trial of each algorithm. The graphs are obviously congested due to the large number of trials run. The figures are shown only to introduce notions of desirable search performance, and not for detailed analysis (that will come in the next three sections). In addition, the figures each show two additional markers: the ten percent inverse image overlap after search marker and the diagonal. These two markers are placed in the figures for reference so that the various figures can be visually compared.

The diagonal marker also provides insight into performance: any point lying below the diagonal represents a configuration where the search algorithm found a solution worse than the best configuration in the starting population. These cases may seem counterintuitive since search should always improve the configurations in the population. However, search is not directly optimizing the inverse overlap, but rather search is minimizing the error function. The second part of each figure (c and d in Figures 5.4 through 5.8) show the before and after error score. From these second set of figures, it is obvious the error is always decreasing. This implies that minimizing the error does not always bring the configurations in the population closer to the ground truth.

Two observations about the figures can be made:

1. The Y axis represents the distribution of initial starting configurations. Notice that the points appear uniformly distributed along the axis. This effect is due to the structured sampling of initial configurations.

2. The X axis represents the quality of the configurations recovered by each search algorithm. Notice that each algorithm tends to cluster the final configurations close to the lower end of the axis range. This effect is due to the algorithm improving upon the initial starting points. A perfect search algorithm, one that correctly recovers the

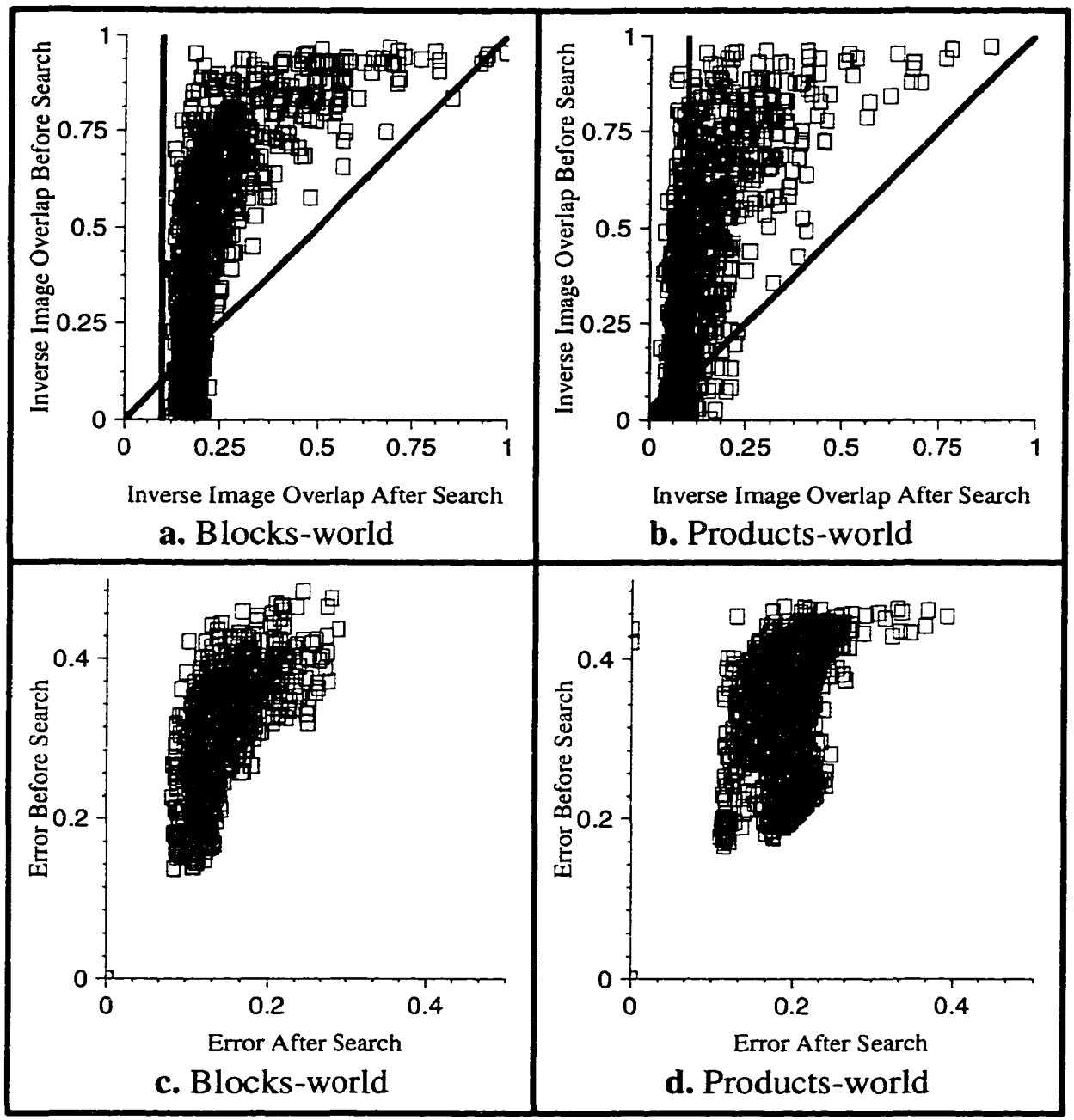

Figure 5.4: Comparing the before and after performance for Evolution Strategies.

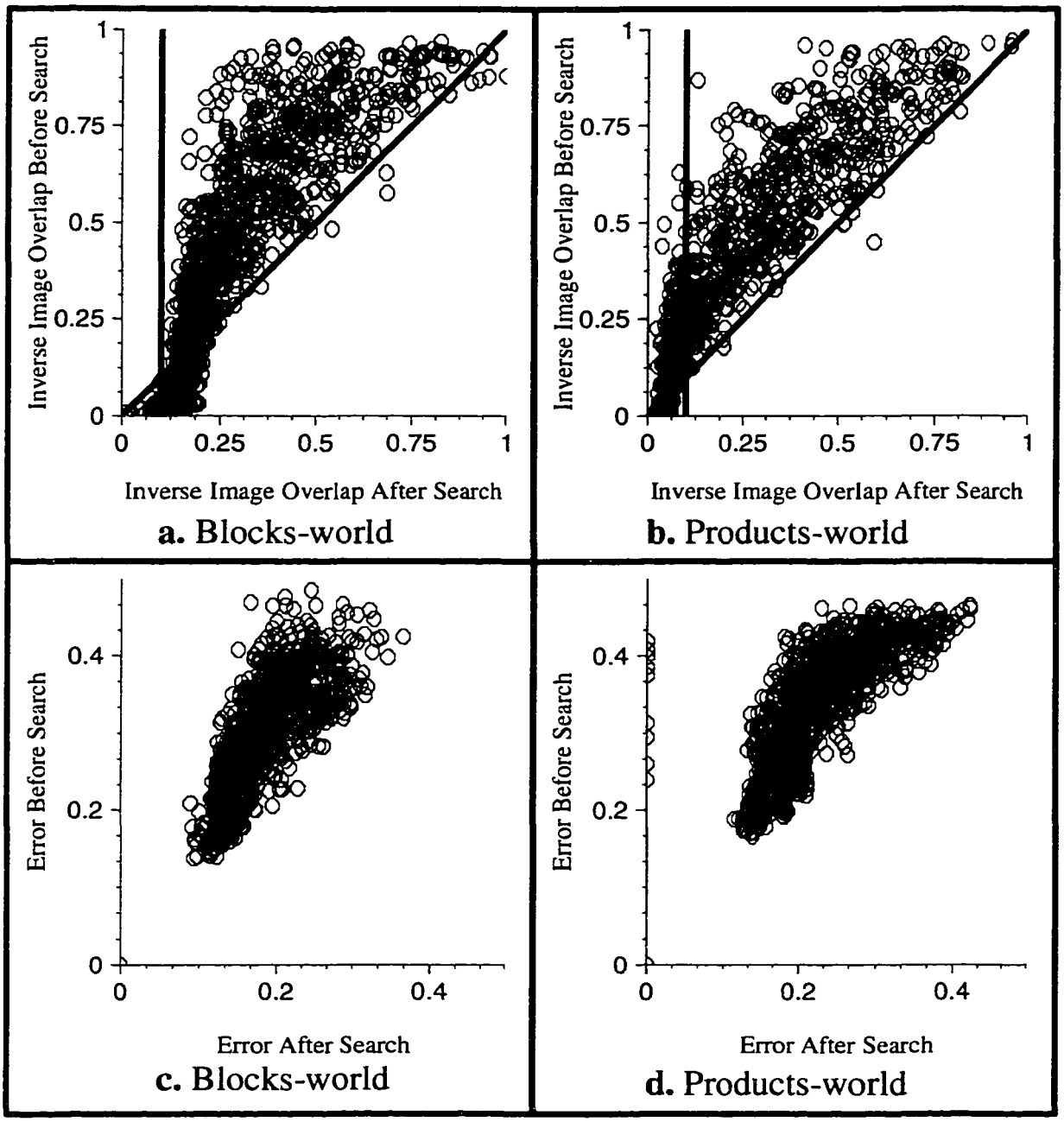

Figure 5.5: Comparing the before and after performance for Simplex.

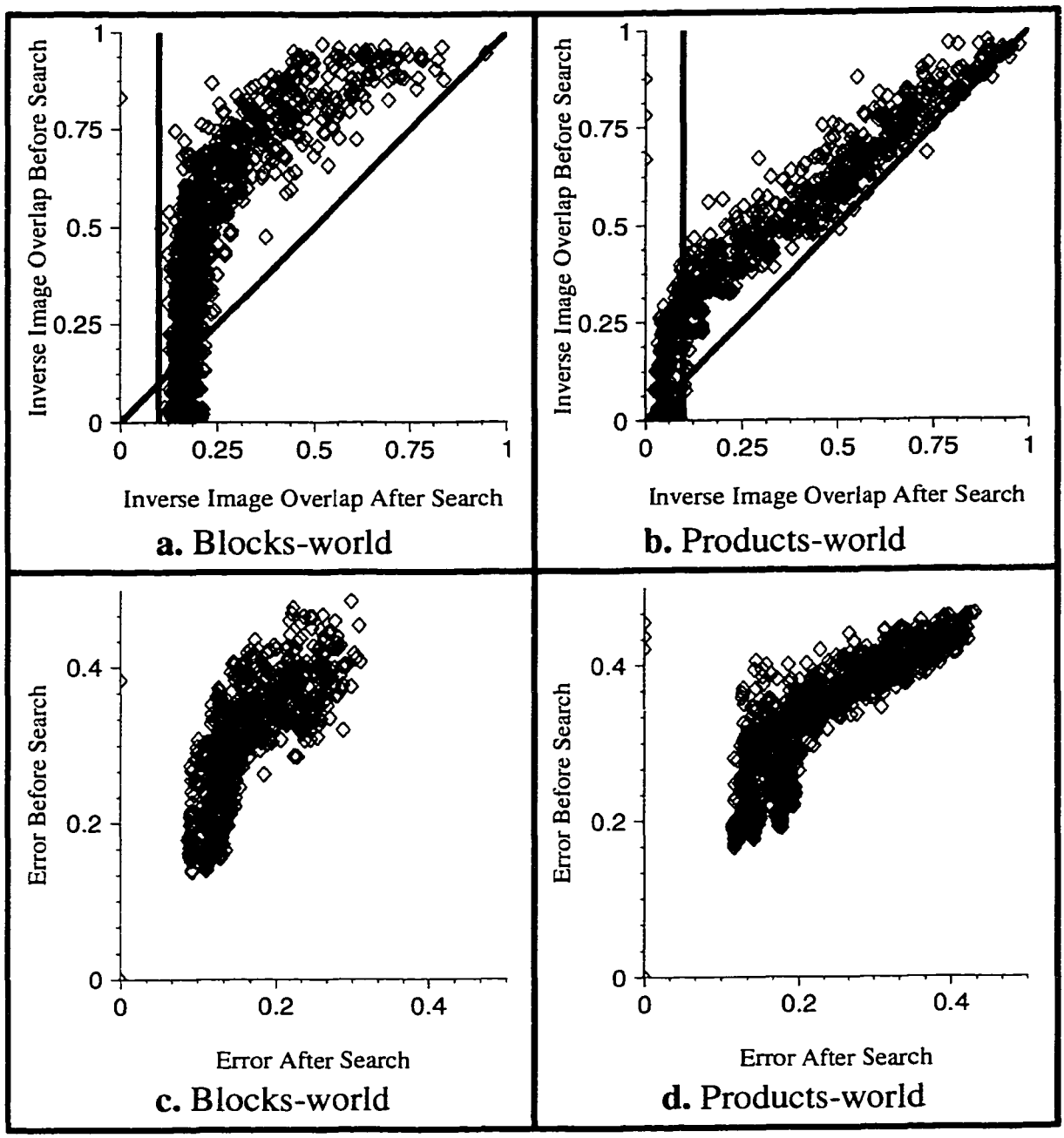

Figure 5.6: Comparing the before and after performance for Hill Climbing.

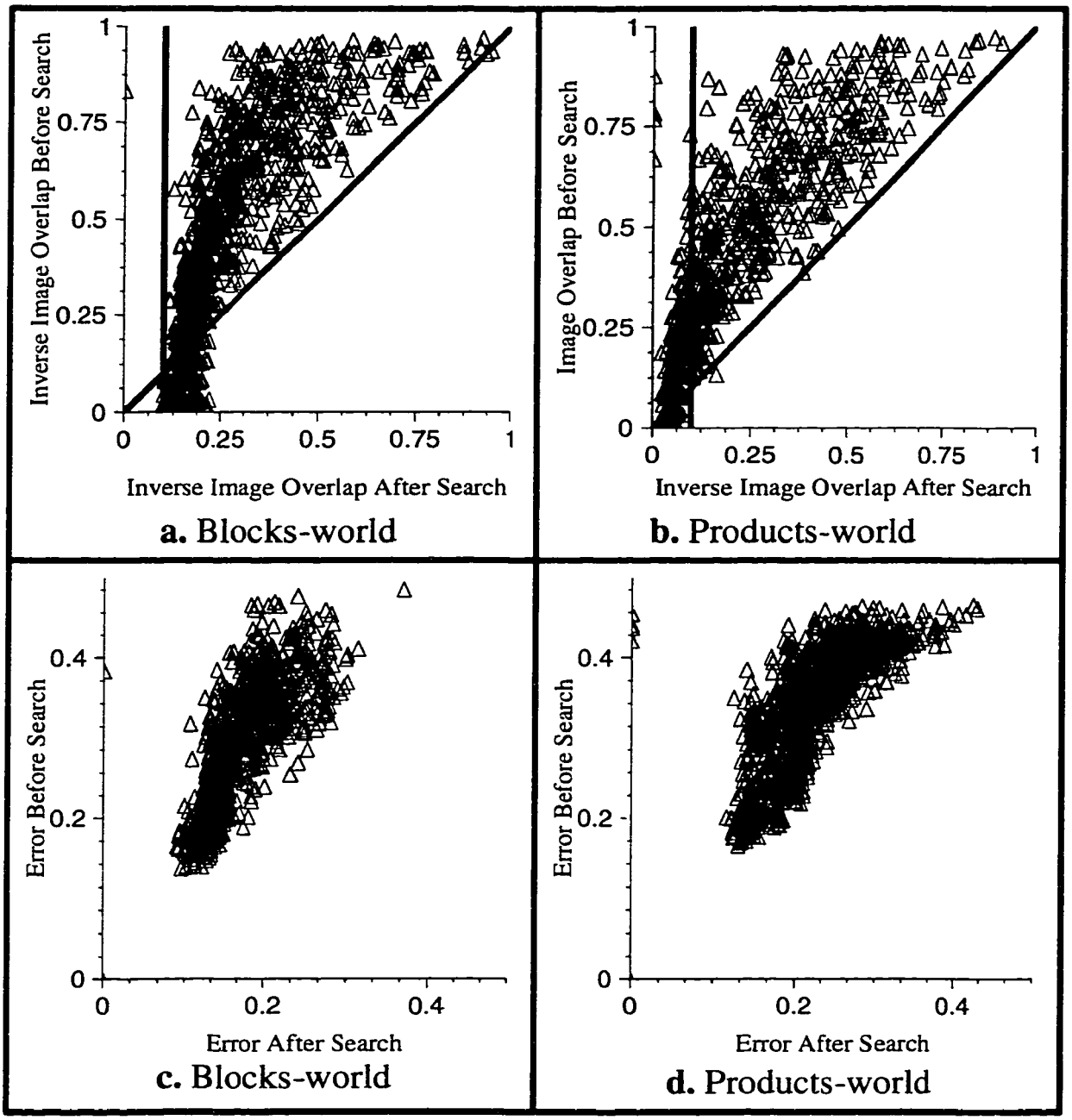

Figure 5.7: Comparing the before and after performance for Combined.

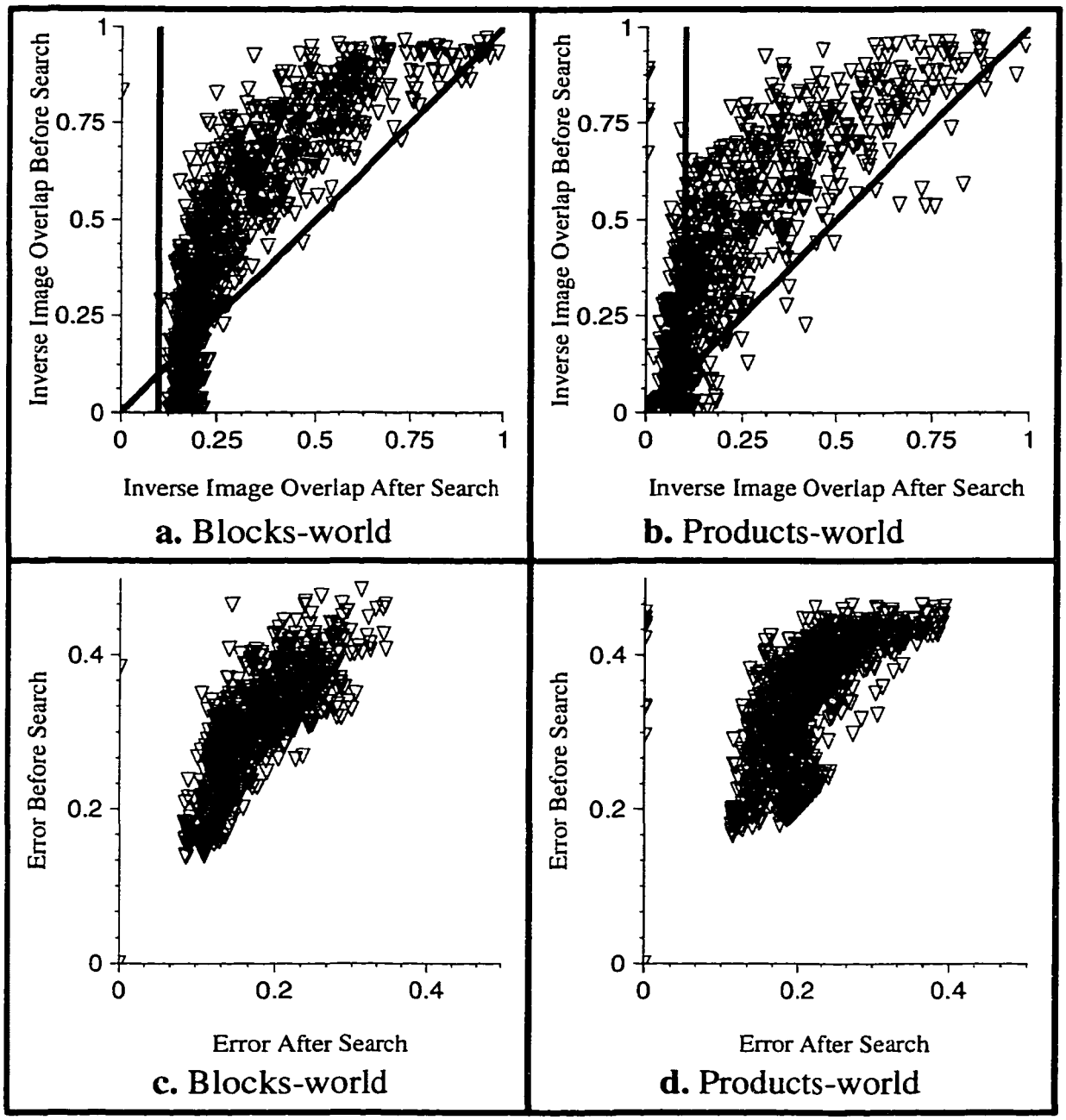

Figure 5.8: Comparing the before and after performance for Mutation.

scene configuration every trial, would generate a vertical line of points at the 0 inverse image overlap after search axis.

At first glance, there seems to be a large discrepancy between the accuracy of the recovered solutions for the two domains. For the products-world images, the various algorithms can obtain solutions with less than ten percent of the image pixels incorrectly aligned with the data. For blocks-world, that number is much higher at around twenty percent. Two explanations are hypothesized to account for the discrepancy. The first explanation is that for blocks-world, the average number of pixels on a given object across all 20 images is $1,632$, whereas for products-world that number is much higher at $3,072$ (the difference is almost a factor of two). Therefore, across both domains the absolute number of pixels incorrectly explained is very similar, but the percentage of correctly aligned face pixels is not. The second explanation is the difference in object texture: blocks-world objects have far less texture that the products-world objects.

To give the inverse image overlap concrete meaning, scene configurations from each ring on two example problems are shown in Figures 5.9 and 5.10. Notice that the configurations seem very close for the rings less than 8. Above 8 the differences become more noticeable; ring 8 represents the case where more that 60 percent of the object features overlap the correct pixels in the image. Remember that the inverse image overlap measures how much of each object face does not overlap the correct visible region it should. Therefore, higher rings (such as ring 20 in Figure 5.10) have the object overlapping the correct region in the image, only the faces are either rotated from where they should be or are covering regions where the object is actually occluded.

5.2.1.2 Search Algorithms Should Maximize p

Estimating the probability of finding the correct solution given a randomly selected starting point is essential for assessing the performance of a search algorithm. Recall that a successful trial can be defined in terms of ϵ:

$$\left|\mathcal{S}^{a} - \mathcal{S}^{true}\right| \leq \epsilon \tag{5.19}$$

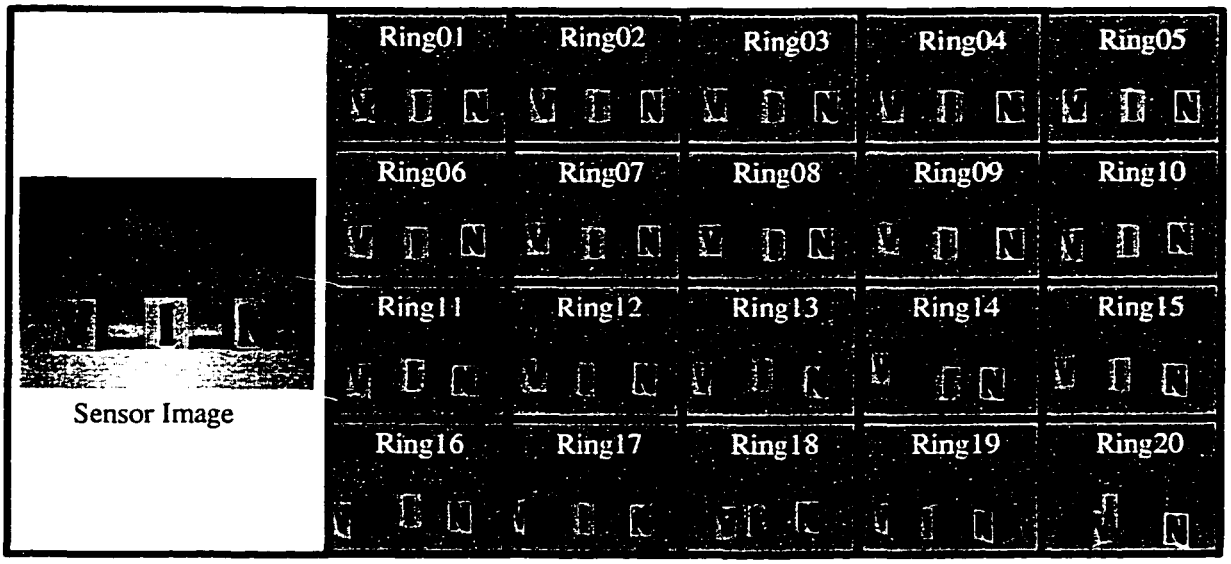

Figure 5.9: Example configurations from various rings for the blocks-world problems.

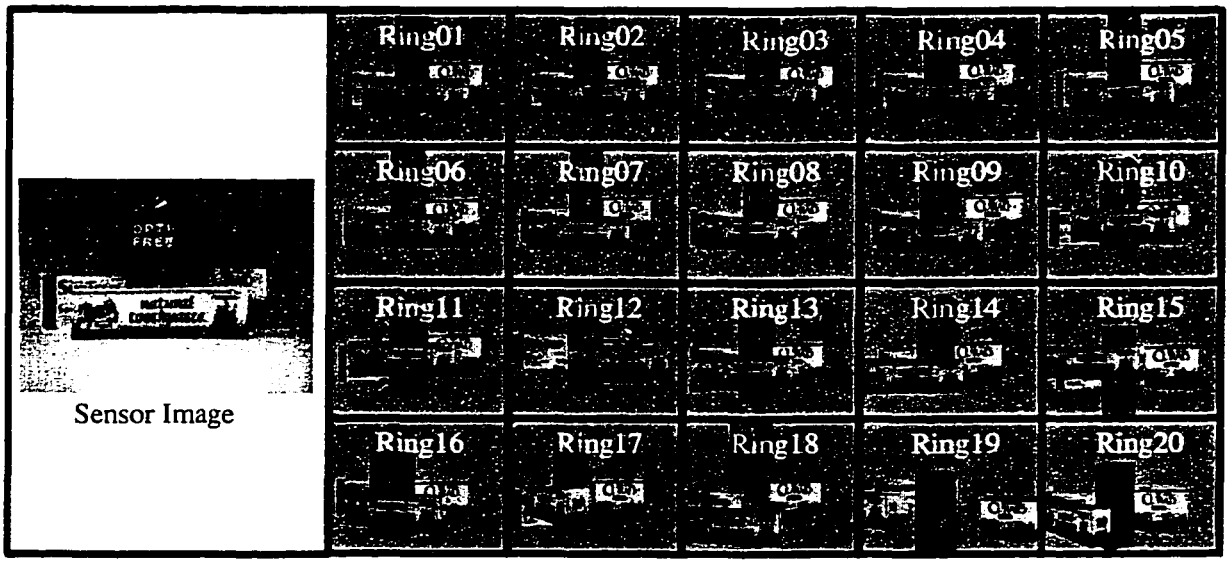

Figure 5.10: Example configurations from various rings for the products-world problems.

If the solution S^a found by the algorithm is within ϵ of the true configuration, then that trial is a success.

Assigning a suitable value to ϵ for an arbitrary scene is problematic. If ϵ is set too high, then the trials labeled as success may not actually be good solutions. If ϵ is set too low, then the search algorithm will only produce failures. In order to determine possible settings for ϵ, ϵ can be viewed as partitioning the set of initial configurations: 1) those starting configurations which are less than ϵ from the true and 2) those starting configurations which are greater than ϵ from the true. These categories are labeled as *inside* and *outside* respectively. Given these labels, the role of search is twofold:

1. Search must not move configurations initially inside the success radius to outside that radius (i.e., should not turn successes into failures). This case is referred to as inside-to-outside.

2. Search should move configurations initially outside the success radius to inside that radius (i.e., should turn existing failures into successes). This case is referred to as outside-to-inside.

Counts of inside-to-outside (IO) trials and outside-to-inside (OI) trials can be turned into percentages:

$$P(IO) = \frac{|inside - to - outside|}{|inside|} \qquad P(OI) = \frac{|outside - to - inside|}{|outside|} \tag{5.20}$$

Obviously the choice of ϵ will effect both $P(IO)$ and $P(OI)$: if ϵ is too high, most trials will start in the inside category; if too low, most will be in the outside category. Figure 5.11 shows $P(IO)$ plotted against $P(OI)$ for the two different datasets. ϵ is lowest in the lower right corner and increases to the left. For clarity, some values of ϵ are laid over the curves. Several patterns are observed in the data:

1. As ϵ increases (from the right of the X axis to the left), the $P(IO)$ values decrease. This phenomena is expected: when ϵ is low, most of the initial (inside) configurations reside very close to the true. Therefore, slight adjustments to the population will most likely produce solutions outside of the radius. As ϵ increases, the diversity of the

initial configurations becomes greater, and it is easier to make improvements: most trials will remain less than ϵ. The desired value of ϵ should produce low $P(IO)$ values.

2. As ϵ increases (from the right of the X axis to the left), the $P(OI)$ values increase. This phenomena is also expected: when ϵ is small, the search algorithm should have difficulty moving inside the ϵ radius since that radius dictates only extremely accurate solutions. As ϵ increases, the definition of success becomes more relaxed and therefore easier to achieve. The desired value of ϵ should produce high $P(OI)$ values.

3. An ϵ of 1.0 will simultaneously maximize $P(OI)$ and minimize $P(IO)$. However, the solutions being labeled as success are of extremely poor quality as they reside anywhere within the sampling rings (see the lower right image in both Figures 5.9 and 5.10) and not necessarily close to the correct solution: everything is a success regardless of how close the produced configuration is to the ground truth.

The choice of ϵ dictates the quality of the successful solutions and should be influenced by three factors: 1) ϵ maximizes $P(OI)$, 2) ϵ minimizes $P(IO)$ and 3) the magnitude of ϵ in relation to some pre-determined bound on solution quality. Relying only on the first two factors can lead to incorrect interpretations of Figure 5.11.

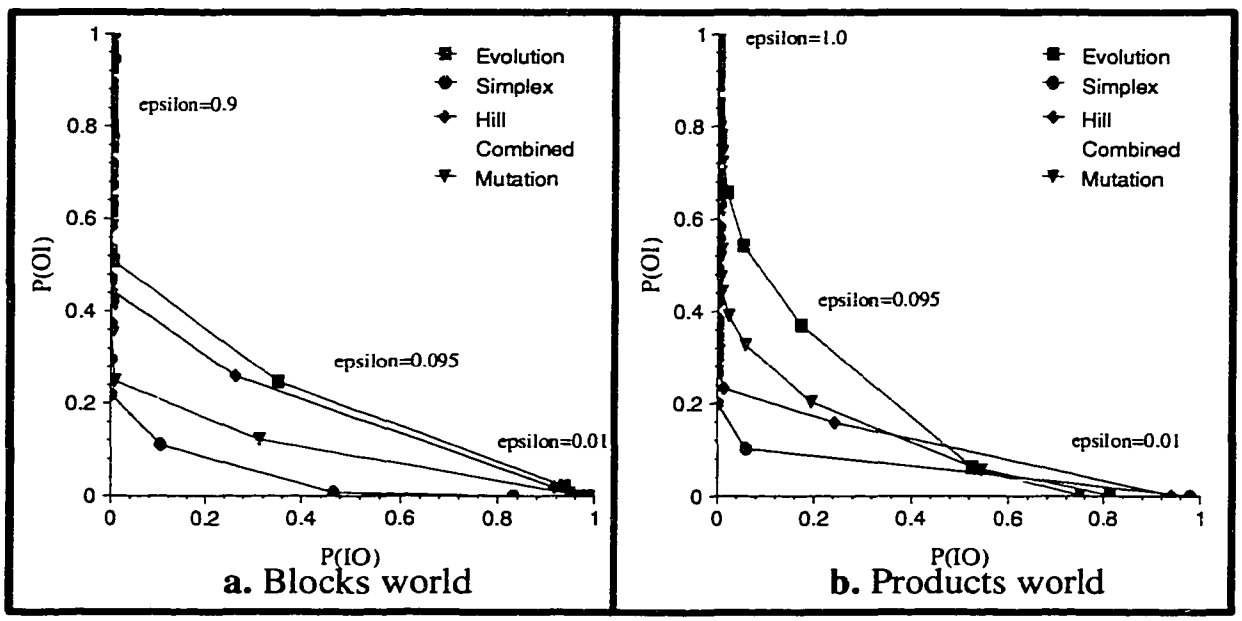

Figure 5.11: Examining possible ϵ values.

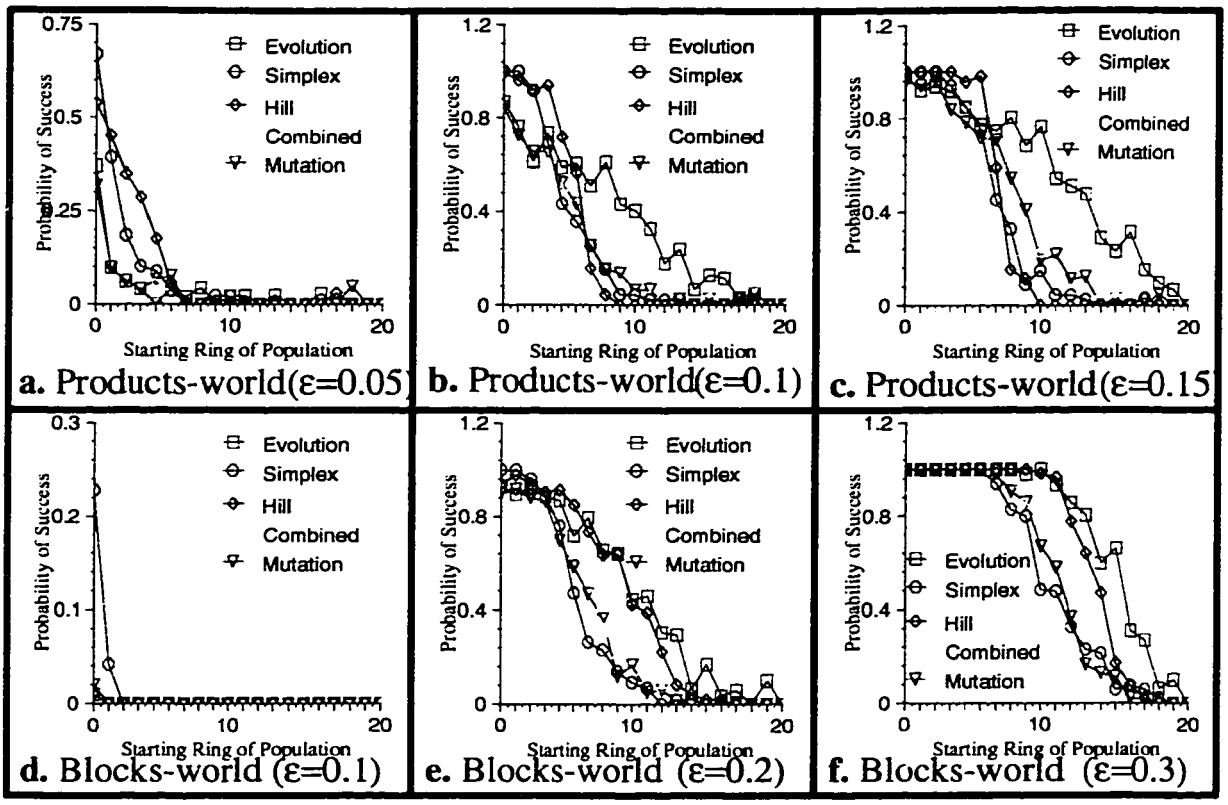

Figure 5.12: The probability of success, p.

The third factor is designed as a point of reference to make the ϵ value chosen correspond to what is perceived as a good solution. Based on these settings and what is shown in Figure 5.11, three different values of ϵ are examined: for products-world values of 0.05, 0.1 and 0.15 and for blocks-world values of 0.1, 0.2 and 0.3. For context, setting $\epsilon = 0.2$ in blocks-world and $\epsilon = 0.1$ in products-world implies that 326.4 pixels on average will be incorrectly explained in the blocks-world domain and 307.2 pixels will be incorrectly explained in products-world.

Figure 5.12 shows estimates of p for each starting ring across both domains for the three different ϵ settings. The estimation of p involves averaging the number of trials in each starting ring converging to a success. These figures indicate how often each search algorithm succeeds given that the algorithm started from a population that was a fixed distance from the ground truth configuration. Based on these six plots, several observations are made on the relationship between ϵ and p:

1. When ϵ is set too low (Figures 5.12a and 5.12d) a floor effect arises where all algorithms have a very low p. In these situations, the algorithms are difficult to compare (as in Figure 5.12d) since very few successful search trials occur.

2. As ϵ is increased (Figures 5.12b and 5.12e), better estimates of p are possible: these estimates have fewer starting rings where $p = 0$. A comparison across the search algorithms for the cases where the starting ring was less than five should be similar to what was observed for the smaller ϵ values (the shapes and relationships of the curves are similar).

3. As ϵ continues to increase (Figures 5.12c and 5.12f), the curves shift to the right and upwards. The relationships between the different search algorithms remains stable, only many more successes have been added so the p values are much higher.

4. The choice of ϵ heavily biases the performance assessment of each search algorithm. Depending upon the ϵ used, some algorithms may seem to outperform others. Increasing ϵ may completely change which algorithm appears best. However, as the settings for ϵ increase, the comparison should stabilize.

For the remaining analysis, $\epsilon = 0.2$ in blocks-world and $\epsilon = 0.1$ in products-world. Based on these settings (Figures 5.12b and 5.12e), several observations about search performance can be made:

1. The value of p is fairly high when the starting configurations are close to the true and drops as more error is added to those configurations. The phenomena fits our intuition that the iterative refinement technique of RMR does better with close initial starting conditions.

2. Initially, the Simplex algorithm has a higher estimate of p than does the Evolution Strategies. At some point, the p values for Evolution Strategies pass those of Simplex. When the initial solutions are fairly homogeneous and close to the optimum, Simplex performs better due to the neighborhood formulation. As the population diversifies, the reproduction operators give an advantage to the Evolution Strategies.

3. In both domains, there is a small region around ring 5 where hill climbing outperforms all of the other algorithms. It seems that hill climbing remains more stable for rings nearby the optimum.

4. Overall, the Evolution Strategies tends to have a high probability of achieving success for most of the given starting rings. In the case of blocks-world, Hill Climbing seems to give comparable performance.

5. There is a sharp and continuous drop in p for each algorithm. Each scene has a region on the error surface where the error function provides reliable information about the correct direction to move to bring the objects into alignment with the imagery. Outside of this region, the information provided by the error function becomes less reliable. The drop off in p is due to the initial configurations being so far from the ground truth configuration the error function is providing erroneous information.

5.2.1.3 Search Algorithms Should Maximize Solution Quality

Besides the binary classification of success and failure, the overall quality of solutions produced by each algorithm can be examined. To assess quality, a histogram of the inverse image face feature overlap is shown in Figure 5.13. The farther to the left the mass of the distribution is, the higher the overall solution quality. For each pair of algorithms, two different tests are used to determine if there is a significant difference between the means of the distributions. These tests are designed to accept or reject one of two possible hypotheses.

The first is known as the null hypothesis (HO) and the second is the alternative hypotheses (H1):

$$H0: \quad \mu_{alg-a} = \mu_{alg-b} \tag{5.21}$$

$$H1: \quad \mu_{alg-a} < \mu_{alg-b} \tag{5.22}$$

The two hypotheses are aimed at determining if the means of two specific algorithms are equivalent. Before that question is addressed, we must determine whether a difference exists between any of the means across all of the algorithms. If all of the distributions are not significantly different, there is no need to apply any paired tests.

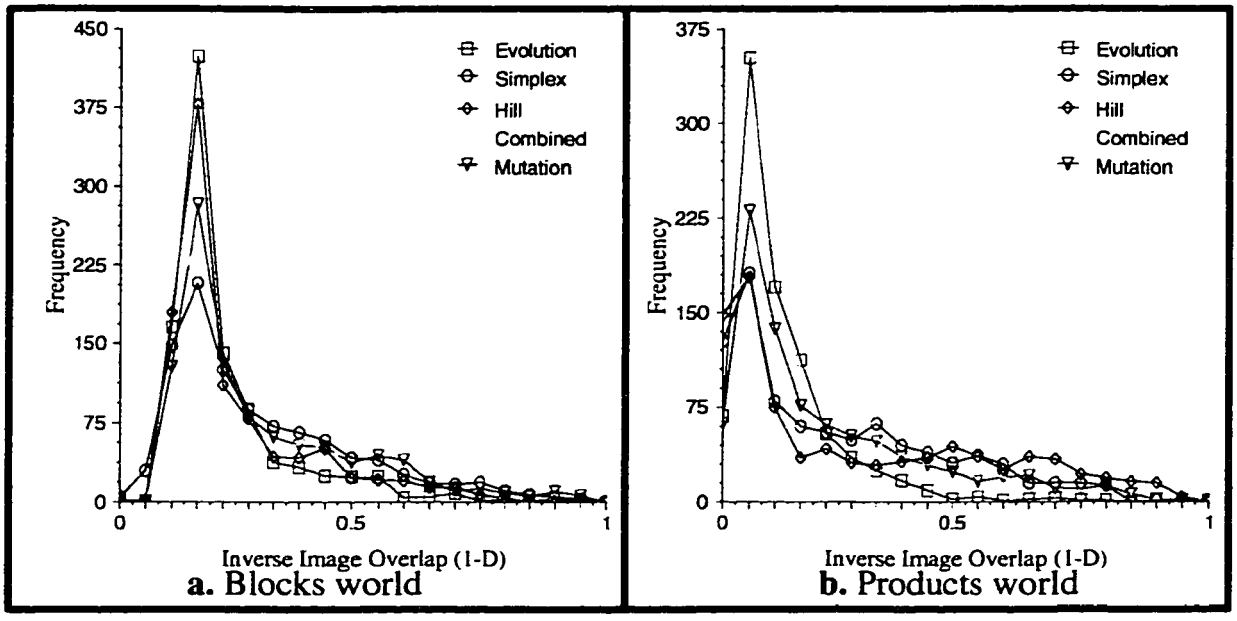

Figure 5.13: The quality of solution produced by each algorithm.

The ANOVA test[3] is used to determine if there exists a difference in performance between the algorithms (Cohen 1995). Table 5.2 shows the ANOVA results on all the algorithms. The table can be interpreted as follows: the *between* category represents the test across search algorithms and has DOF equal to one minus the number of algorithms. The *within* category represents the variation of all of the trials across all of the algorithms. The DOF for the within is equal to the total trials minus the DOF for the between case. The F distribution is used to determine if there is a difference between all of the algorithms. If F is large enough with respect to the degrees of freedom, then the distributions are assumed not to be equivalent.

Based on the table, the ANOVA shows there is a statistically significant difference between the means of the different search algorithms. Paired t-tests were then run to compare the algorithms individually. Table 5.3 shows all of the paired t-tests for the five different search algorithms for both test domains.

[3]The ANOVA test tells whether sample means of various factors are significantly different from one another and whether they interact significantly with each other. The test is usually used to compare differences between three or more means (or groups). The null hypothesis for the ANOVA is that the means of all groups are equal, and the alternative hypothesis is that they are not.

	Degrees of Freedom	Mean Square	F
Between Search Algorithms	4	1.04	39.1 with 4 degrees of freedom ($p \leq 0.001$)
Within	5000	0.03	
Total	5005		

a. blocks-world

	Degrees of Freedom	Mean Square	F
Between Search Algorithms	4	2.87	70.0 with 4, degrees of freedom ($p \leq 0.001$)
Within	5000	0.04	
Total	5005		

b. products-world

Table 5.2: Using the ANOVA test to compare the quality of solutions.

In most cases, there is a significant difference between the tested distributions. Large positive or negative values of t imply the null hypothesis (H0) can be rejected and that the performance across algorithms is not equivalent.

	μ	σ	Evolution	Simplex	Hill	Combined	Mutation
Evolution	0.246	0.128	-	**-9.9**	-3.2	**-5.8**	**-10.4**
Simplex	0.318	0.189	*9.9*	-	*6.8*	*4.5*	-0.4
Hill	0.266	0.148	3.2	**-6.8**	-	-2.5	**-7.2**
Combined	0.283	0.154	*5.8*	**-4.5**	2.5	-	**-4.9**
Mutation	0.321	0.186	*10.4*	0.4	*7.2*	*4.9*	-

a. blocks-world

	μ	σ	Evolution	Simplex	Hill	Combined	Mutation
Evolution	0.145	0.111	-	**-13.7**	**-16.1**	**-10.6**	**-12.0**
Simplex	0.257	0.212	*13.7*	-	**-4.0**	*3.6*	1.7
Hill	0.304	0.266	*16.1*	*4.0*	-	*7.3*	*5.5*
Combined	0.223	0.184	*10.6*	**-3.6**	**-7.3**	-	-1.9
Mutation	0.240	0.205	*12.0*	-1.7	**-5.5**	1.9	-

b. products-world

Table 5.3: Using the t-values from the t-test to compare the quality of solution produced by each algorithm. Significant ($p \leq 0.0001$) negative t-values are shown in bold. Significant ($p \leq 0.0001$) positive t-values are shown in italic.

Table 5.3 can be interpreted in the following manner. A negative value of t implies that the search algorithm in a given row has a lower mean than the search algorithm in the corresponding column (the alternative hypothesis (H1) is accepted). This interpretation is only valid when the test is significant. Negative t-values that are significant are shown in bold. Positive t-values that are significant are shown in italic. Entire rows filled with boldface values imply that search algorithm is outperforming all others. Based on the t-values in the table, the following observations can be made:

1. In both test domains, the Evolution Strategies algorithm recovers very good solutions (the mean and standard deviation are smaller than all of the other algorithms except for when comparing to Hill Climbing in blocks-world). The null hypothesis (H0) that it is equivalent to Simplex, Mutation-Only and Combined search can be rejected.

2. Since the mean and standard deviations for the Evolution algorithm are lower than the means and standard deviations for Simplex, Mutation-Only and Combined search, the alternative paired hypotheses (H1) comparing Evolution Strategies to each of these algorithms is supported.

The table shows that part of our intuition in the theoretical comparison section was correct. Hill Climbing did not perform well in the products-world domain, but did quite well in blocks-world. This discrepancy can be explained in terms of the amount of local object texture. The blocks-world objects tend to have large regions of similar color in the image. Optimizing each parameter independently works well on these homogeneous regions but has trouble for scenes with lots of variation at the pixel level: with uniform texture small search moves increase the overlap of similar color regions but for highly changing texture small movements decrease overlap of color regions.

5.2.1.4 Search Algorithms Should Minimize Work

Besides the probability of finding the correct solution given an arbitrary starting point, the effort expended by the algorithm before convergence is also examined. The reason for such an analysis may not seem obvious at first, but can be demonstrated by a simple example. Imagine recording the following values for two algorithms: 1) Algorithm A has a probability of success at 0.99 and a run time of $10,000,000,000,000$ seconds per trial, 2) Algorithm B has a probability of success at 0.25 and a run time of 0.08 seconds per trial. Notice that while Algorithm A is virtually guaranteed to find the correct solution, it does not do so in a feasible amount of time. Instead, Algorithm B has a much lower chance of finding the solution but runs fairly quickly. By running Algorithm B just 40 times (based upon the Binomial values in Table 5.1) finding the correct solution is almost certain. The cost, or total work, involved in running Algorithm B 40 times is just 3.2 seconds.

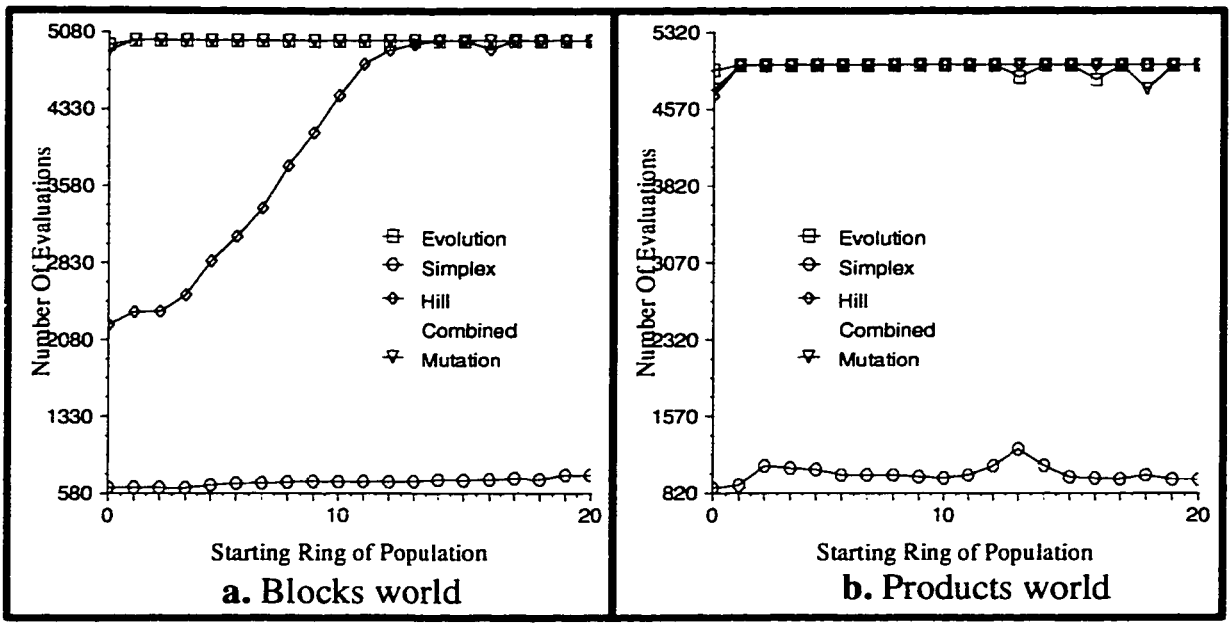

Figure 5.14: The amount of work, or effort, expended by a given search algorithm.

The goal then is to find the total amount of work W required by each algorithm to find the true configuration with a certain level of confidence. Recall Equation 5.14:

$$W = \overline{w} \cdot n \tag{5.23}$$

where $\overline{w}$ is the average amount of work per trial and is computed in terms of the number of error function evaluations. The value of n is determined based upon p and the expectation of finding the correct solution.

The first criterion was used to estimate the value of p for a given ring. Now assume that the desired likelihood of finding the correct solution using a number of trials is set to 0.99. This means that for each image, there is a 99 percent chance that the algorithm will find the correct solution when the correct number of trials are run. Given a value of p, and $P(k \geq 0.99)$, Table 5.1 can be examined to determine n. Once an estimate of $\overline{w}$ is computed, the value of W can be calculated.

In order to determine $\overline{w}$ for the five different algorithms, the number of error function evaluations is averaged across rings. Recall that search can terminate in one of two ways: 1) the error values in the population become nearly equivalent, 2) a maximum of 5,000 error evaluations has been expended. In either case, once search has terminated the best

configuration in the population (in terms of error) is returned as the recovered scene configuration. Figure 5.14 shows $\overline{w}$ for each starting ring. Several observations about this figure can be made:

1. The Mutation-Only, and Evolution Strategies often do not converge to a solution. Instead, each search algorithm terminates by reaching the maximum number of error evaluations (this observation was predicted by the theoretical analysis).

2. The Simplex and Combined algorithms both converge using the similarity of error value constraint. The Simplex clearly requires the fewest average number of evaluations to converge.

Figure 5.15 shows the total work required by each algorithm to locate the desired solution for each given ring, in terms of the p given in Figure 5.12, the $\overline{w}$ given in Figure 5.15, and with a confidence of 0.99. Several observations about this figure can be made:

1. While the Evolution Strategies takes the maximum number of evaluations on any given trial, the value of p is much higher. Therefore, over the course of multiple trials, the Evolution Strategies tends to have a lower total work. Also, for the rings farther from the true, the Evolution Strategies have a non-zero p, which means that given sufficient trials the algorithm will eventually find the correct solution. This is in contrast to the Mutation-Only and Hill Climbing search which tend to have a very small value for p in the larger rings.

2. Subtle differences exist between Hill Climbing in blocks-world and Hill Climbing in products-world domains. In products-world, Hill Climbing quickly becomes computationally infeasible for the larger rings. This compares to blocks-world where Hill Climbing is a very competitive search algorithm. This observation is expected based on the estimate of both p and $\overline{w}$.

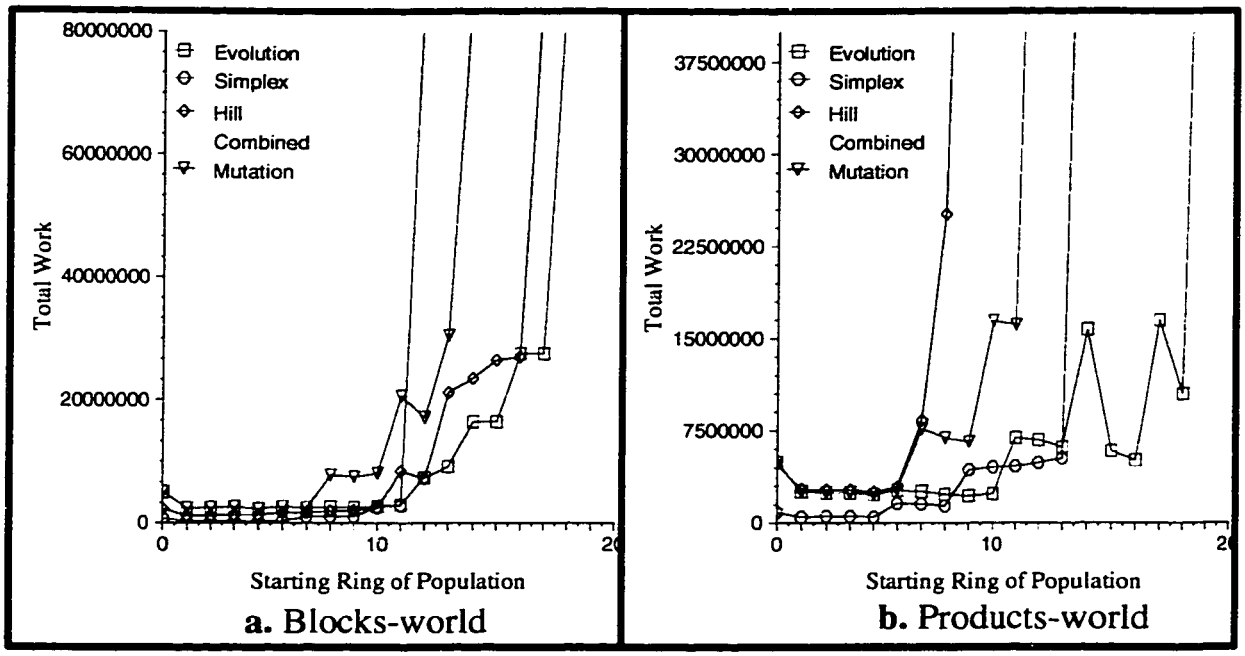

Figure 5.15: The total amount of work, or effort, expended by a given search algorithm to find the correct solution with 0.99 confidence for each ring.

5.2.2 Which Search Algorithm to Use When

The empirical comparison of search provided insight into which algorithms are robust under varying conditions. Based on that analysis, a few *rules of thumb* are proposed to decide which search mechanisms are most appropriate for a given situation:

- Hill Climbing performs well in domains where object surface texture remains stable. Quickly changing surface markings prevent the small, uniform parameter steps from reducing the error. In essence, highly volatile texture produce numerous local optimum within the fixed Hill Climbing neighborhoods.

- Mutation-Only can be viewed as a Hill Climbing algorithm which takes non-uniform steps. As such, Mutation-Only search performs better on objects having lots of surface texture.

- The Simplex search algorithm performs well when the starting scene configurations are close to the optimum. The Simplex algorithm converges quickly to good solutions (solutions close to the optimum) when the starting ring of the population is less than 10 (50 percent of the object faces overlap in the starting configuration).

- On average, the Evolution Strategies recovers high quality solutions. The Evolution Strategies demands a large number of error evaluations on any single trial of search. However, over the course of multiple trials, the total amount of work expended is comparable to the other search algorithms examined.

- Combined Search requires further examination. Simply switching back and forth between Evolution Strategies and Simplex search moves does not achieve performance better than either algorithm in isolation. Future work may further study ways of exploiting the various strengths of the different algorithms.

For the remaining RMR evaluation, the main concern is recovering high quality solutions. Furthermore, no constraints can be placed on how close the starting configurations are to the ground truth configuration. Given these two statements, Evolution Strategies appears best for the current recognition domains.

Chapter 6

Evaluating RMR

> "If your experiment needs statistics, you ought to have done a better experiment."
>
> - *Ernest Rutherford*

The previous chapters described an approach to object recognition based upon top-down predictions and iterative refinement. That discussion centered around the best system attributes for solving recognition problems in two example domains. Now a switch is made from constructing the algorithm to a more thorough evaluation of performance based on a wider range of scenes. Currently, the dataset contains 80 test problems (see Chapter 3). This chapter evaluates the entire Render-Match-Refine (RMR) system on those 60 images not used by the previous two chapters. Within these 60 images, there are 190 instances of 17 different objects.

The goal of this RMR evaluation is to determine which environmental factors influence the accuracy of the scene configurations recovered. One should not confuse the evaluation of our RMR implementation with an evaluation of RMR as a viable object recognition methodology. The evaluation of RMR as an approach would entail either comparing RMR to another recognition system on a series of images, or constructing different systems based on variants of RMR and applying them to a variety of different recognition domains. For this dissertation, we are only concerned with assessing recognition performance on the set of test images presenting in Chapter 3.

Before the evaluation of the current RMR implementation begins, we present a brief review of the conclusions drawn in the previous three chapters about the various RMR

components. The complete RMR system takes as input an initial population of scene configurations. For each single configuration, $\mathcal{S}$, an image is rendered. Using an error function $\mathcal{E}(\mathcal{S})$, based on the combination of $\mathcal{E}^{NRGB}_{Correlation}(\mathcal{S})$ and $\mathcal{E}^{EDGEM}_{Correlation}(\mathcal{S})$, the quality of the scene parameters (match) is assessed. An Evolution Strategies refinement algorithm is then used to search over combinations of configurations in the population until either a maximum number of error evaluations has been reached or the population has converged. Once a stopping condition has been reached, the configuration in the population with the lowest error is returned as the best explanation of the scene.

Up until now, the issue of how the initial population is constructed has been avoided. For the evaluation in this chapter, these populations are constructed using a pose indexing algorithm. The details of this algorithm are presented in Appendix A. For this experiment, the number and type of objects present in the scene are assumed to be known. Therefore, false object hypotheses (objects not really in the scene) are not considered.

Pose indexing generates a list of the 25 most likely pose hypotheses for each object present in the scene. After these lists are constructed, scene configurations are created. Conceivably, the scene configurations could be generated using the power set of all possible permutations of each object's hypothesis list. A quick analysis of the size of that set shows the technique is not feasible: for an image with 4 objects and 25 hypothesis per object there are (25)(25)(25)(25) or 78,125 possible configurations. It is impractical to consider all of these combinations.

To generate a population of starting scene configurations, we randomly sample the power set of possible configurations to create configurations of the required size. The required size is based on having $N + 1$ configurations for N scene parameters (4 objects with 4 pose parameters yields a population of 17 configurations). Evolution Strategies is run on 200 of these populations. The number of trials was set to 200 based on the search analysis in Chapter 5: using the p value (0.02) from the binomial equation for the farthest ring (20) and a desired confidence of finding the correct solution of 0.99.

The best scene configuration, in terms of error, in the population before and after search is recorded. The best configuration found across all 200 trials is considered to be

the best explanation of the scene. One slight modification was made to the Evolution Strategies search algorithm discussed in Chapter 5. Across all search trials for each scene, the best configuration found is retained in the population: when a new trial begins, that best configuration is re-inserted into the population. This allows successive trials of search to start with at least one strong starting point instead of a completely random population. This modification further strengthens what was already the best search strategy. We did not test the effects of this modification in the previous evaluation of search as it violates the independence assumption which is essential for estimating the various p values of the Binomial process.

6.1 An Overview of the Results

Figures 6.1 through 6.6 show the best scene explanation across all 60 images. Next to each sensor image, a rendering of the configuration with the lowest error is shown. All of these results were achieved automatically by our RMR implementation (the pose indexing was used to cue the starting populations and the ground truth parameters were not used during recognition). Notice that in many of the scenes, a high percentage of object occlusion exists. Even so, the RMR implementation is able to successfully recover the object configuration. For instance, in Figures 6.2c and 6.2j, a substantial portion of the *speed-stick* object is being occluded by the *vaseline* object. In both cases, RMR was able to correctly determine the $3D$ relationship of these two objects in the scene. Another impressive example is shown in Figure 6.3j, where almost 90 percent of the *red-stapler* is occluded by the *blue-stapler*. However, RMR is again able to successfully recover the relationship between these two objects. Also notice that the *mac-n-cheese* object in Figures 6.1g and 6.1h contains obvious specular highlights. The match error successfully compensates for these highlights even though there is no explicit method for doing so (enough pixels are matching well enough to compensate for the poor match near the highlight).

Even though some of the scenes contain obvious failures, that does not imply a failure to recognize any of the objects in the scene. In most cases, failures are associated with a particular object type (i.e., the *almond-tea* object) where the system recovered most of the

correct scene configuration, but failed on that single object. The goal of this evaluation is to determine the cause of these failures.

One of the problems associated with recognizing the solid colored blocks (each face is the same color) is ambiguity. In most cases, internal edges (not on the occluding contour) will not be detected by the edge features: there will be a very weak response to the edge mask. Therefore, the edge component of the error function will be based on the object silhouette (Seales and Dyer 1992; Koenderink 1984; Stevens 1995). In this domain, the silhouette tends to be highly ambiguous. For instance, rotating the object around the y axis will give the perceived effect of moving the object towards the camera (the silhouette becomes larger). In these cases, the pose parameters appear to interact, and many different scene configurations can be created to produce the rectangular silhouette shown in several of the blocks-world scenes (see the *blue-block* in Figure 6.5f for an example).

6.2 Evaluating Performance

Evaluating performance begins with the identification of specific architecture attributes (dependent variables) and the tasks and environments which influence observed behavior (independent variables) (Cohen 1995). A set of hypotheses are then generated to explain the expected relationship between the independent and dependent variables. Hypotheses are phrased as a set of questions to be answered: *Does independent variable A influence the performance of dependent variable B?* Experiments to answer these questions are then designed and data is collected. The data, which records the relevant variables, is statistically analyzed to test the validity of the original hypothesis.

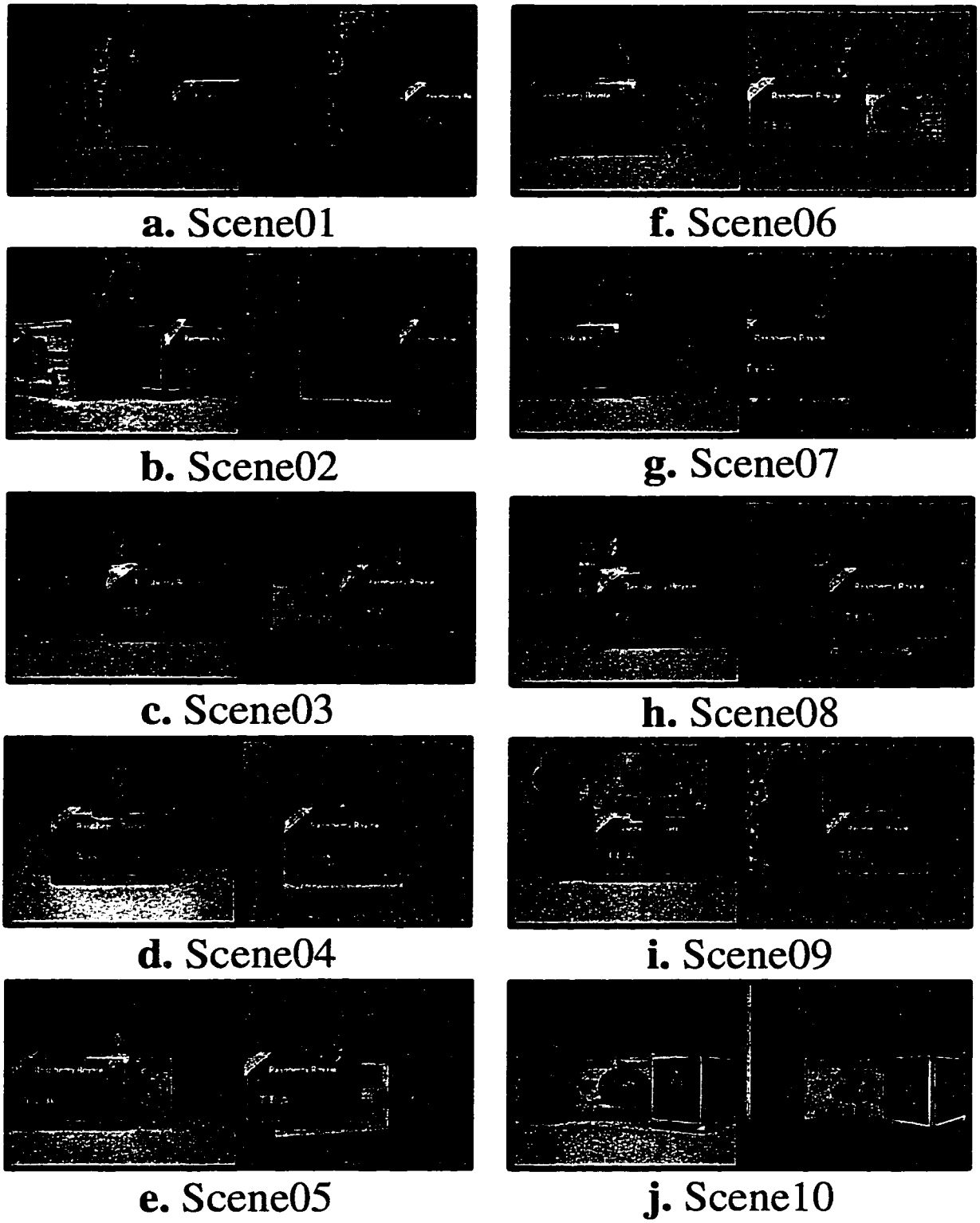

Figure 6.1: Results for the products-world Array01 images. The actual image is shown followed by the prediction for the recovered scene.

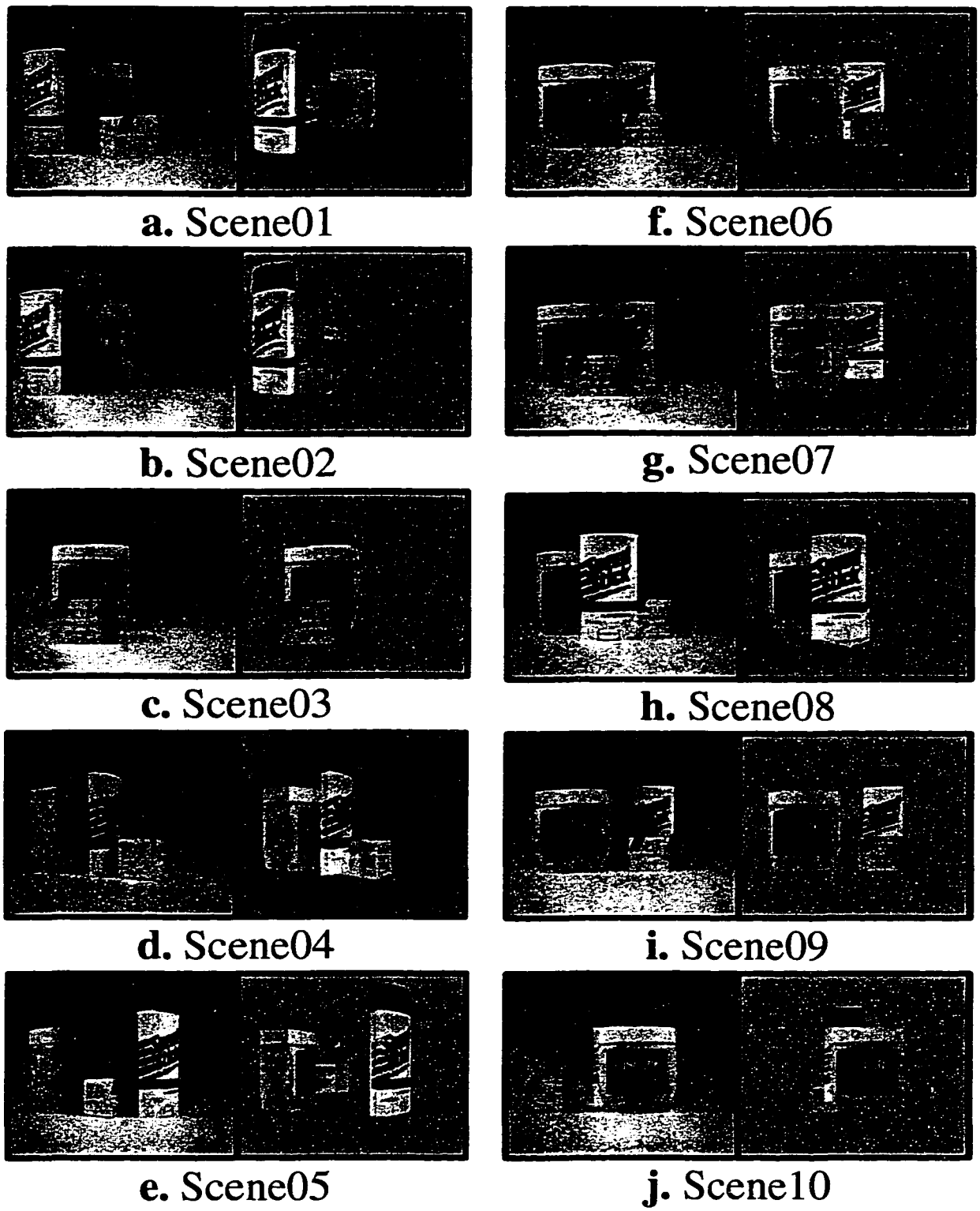

Figure 6.2: Results for the products-world Array02 images. The actual image is shown followed by the prediction for the recovered scene.

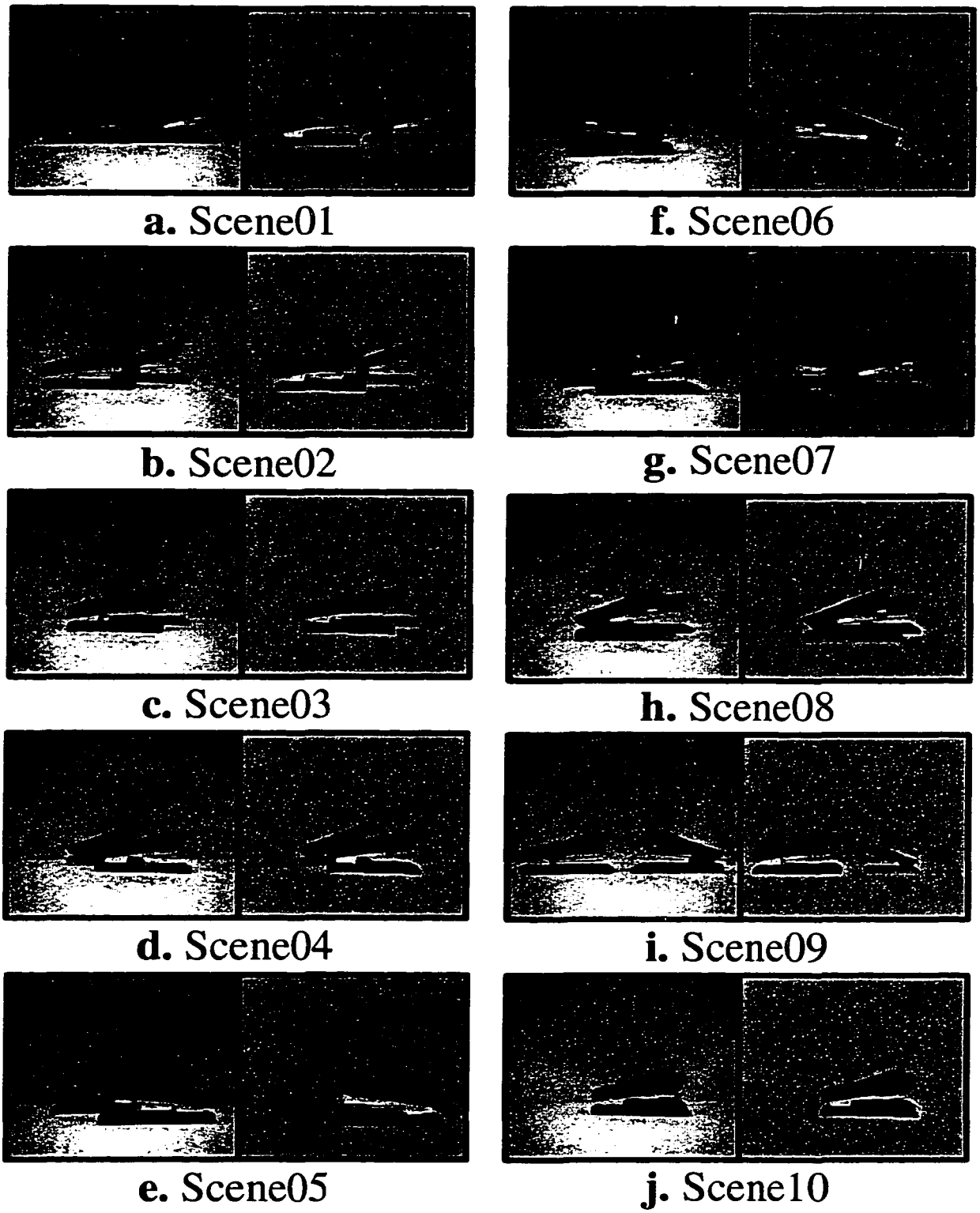

Figure 6.3: Results for the products-world Array03 images. The actual image is shown followed by the prediction for the recovered scene.

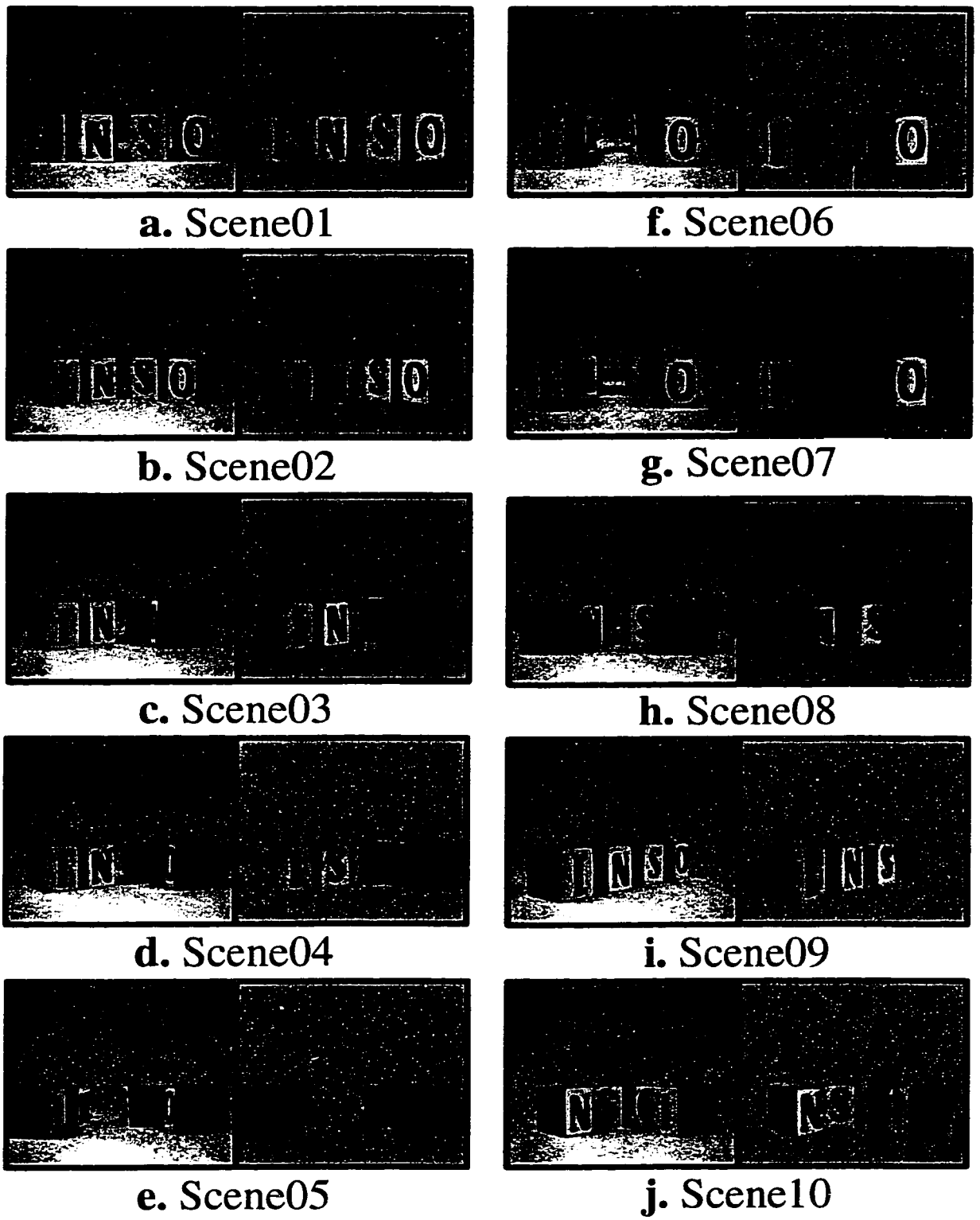

Figure 6.4: Results for the blocks-world Array01 images. The actual image is shown followed by the prediction for the recovered scene.

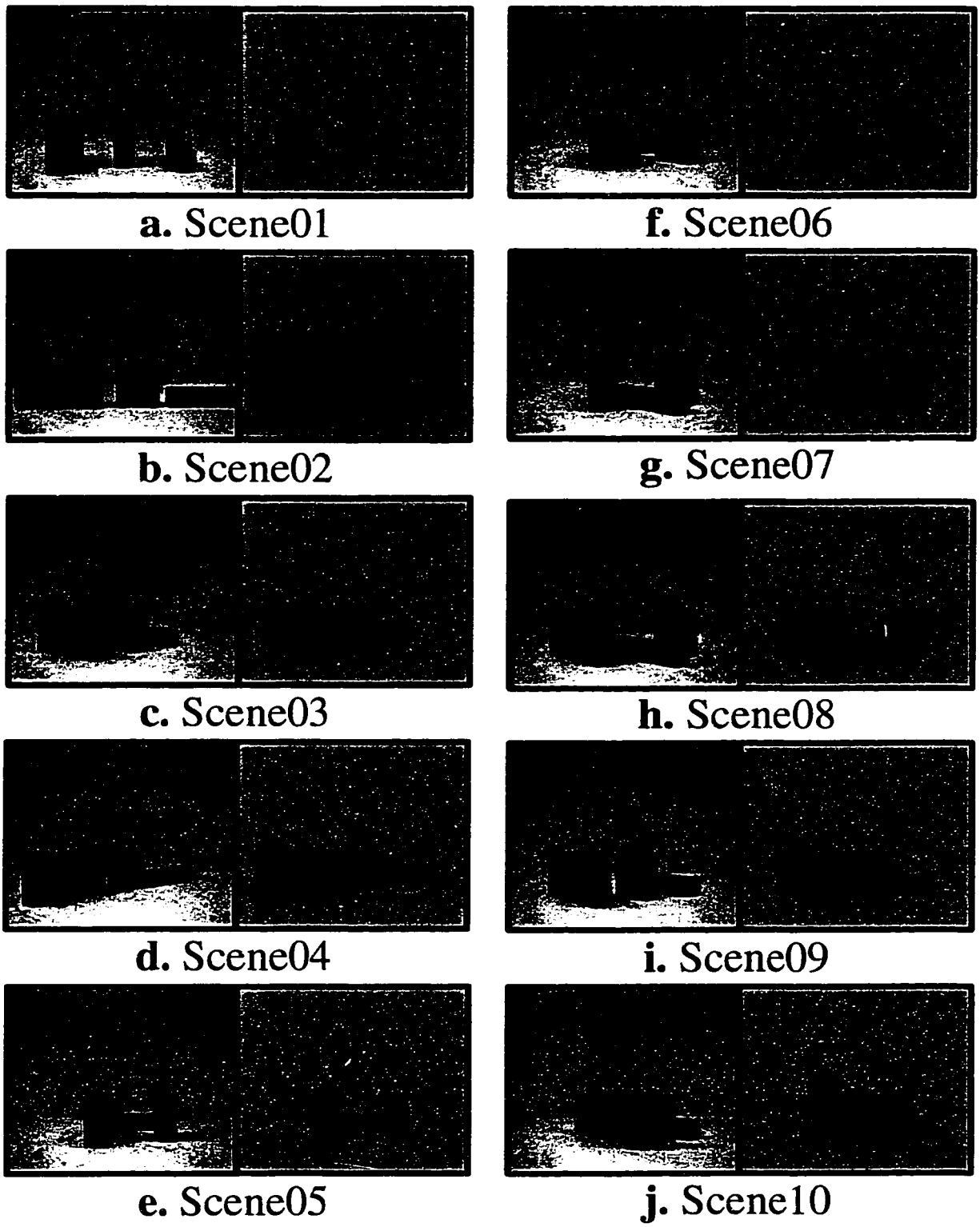

Figure 6.5: Results for the blocks-world Array02 images. The actual image is shown followed by the prediction for the recovered scene.

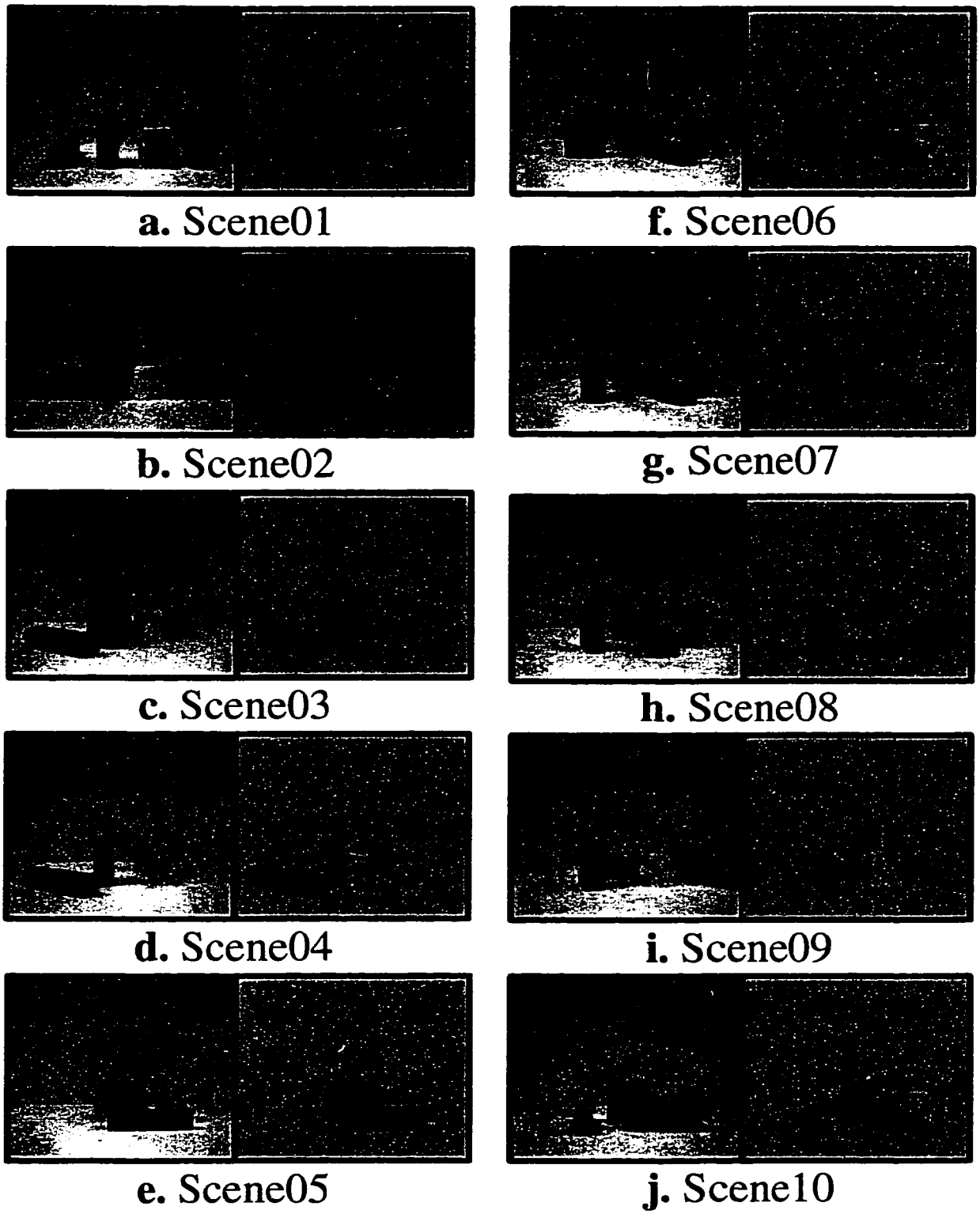

Figure 6.6: Results for the blocks-world Array03 images. The actual image is shown followed by the prediction for the recovered scene.

6.2.1 The Independent Variables

The performance analysis of each RMR component in the previous chapters led to a series of possible reasons our RMR implementation could fail to locate an object. Throughout the dissertation, both object occlusion and object texture were the two main conditions RMR was designed to work under. In addition, the background, or non-object pixels, were listed as influencing the error function. Finally, the starting configurations provided to RMR were shown to influence both the error and search analysis.

Based on these factors, five different scene attributes were recorded: percent object occlusion, a measure of object texture, size of the object in the image, similarity of the object to the background, and quality of initial hypotheses. These attributes, or independent variables, will later be used to characterize the success or failure of our RMR implementation.

6.2.1.1 Percentage of Object Occluded

For each of the 60 different scenes, the ground truth pose parameters of the 190 objects are measured. Using the technique discussed in Section 3.4.1 of Chapter 3, two different sets of pixels are identified based on those ground truth parameters: 1) $\mathcal{Q}_{object}^{true}$ those pixels representing the visible portion of an object, 2) $\mathcal{I}_{object}^{true}$ those pixels which would have belonged to that object if no other objects were present (no occlusion). The percentage of occluded object pixels $\mathcal{O}_{object}^{true}$ in each image is the number of pixels not visible normalized by the total which should be visible:

$$\mathcal{O}_{object} = \frac{\left|\mathcal{I}_{object}^{true}\right| - \left|\mathcal{Q}_{object}^{true}\right|}{\left|\mathcal{I}_{object}^{true}\right|} \tag{6.1}$$

This independent variable is in the range $[0, 1]$ where 0 represents no occlusion and 1 represents complete object occlusion. The upper left histogram in Figure 6.7 shows $\mathcal{O}_{object}^{true}$ for each of the 83 occluded object in all 60 scenes. At first glance, the amount of object occlusion in the entire dataset seems small: most of the objects are not occluded (107 out of 190). Conceptually, at least one object for each scene must have no object occlusion: the object closest to the camera causing the occlusion of the other objects. Obviously, if all of the objects were included in this histogram, the single object case would account for at least 60 of the samples in the 0 bucket of the histogram.

6.2.1.2 Amount of Texture on the Visible Object Surface

To estimate the amount of texture visible on a given object, a measure of entropy is used (Umbaugh 1997). Entropy $\mathcal{T}_{object}$ estimates the number of bits needed to encode all visible object colors. In the worst case, the number of required bits is equal to the dynamic range of the camera. For $24 - bit$ color, that worst case is:

$$\mathcal{T}_{object} = 2^{24} \tag{6.2}$$

However, for an arbitrary object, all possible colors will not be needed to render the object. To measure texture, the number of colors used by the object is computed:

$$\mathcal{T}_{object} = \frac{\sum_{r=0}^{255}\sum_{g=0}^{255}\sum_{b=0}^{255}\begin{cases} 1 & Hist[r][g][b] \geq 1 \\ 0 & \text{otherwise} \end{cases}}{\sum_{r=0}^{255}\sum_{g=0}^{255}\sum_{b=0}^{255} Hist[r][g][b]} \tag{6.3}$$

where $Hist$ is a histogram of the visible object colors in the image. The measure is normalized by the total number of pixels to produce a number in the range $[0, 1]$. A value of 1 represents each visible object pixel being a different color (the object is highly textured). As this number decreases, the observed pixels are less volatile and can be represented with fewer bits (the object is less textured). The upper right histogram of Figure 6.7 shows $\mathcal{T}_{object}$ for all 190 objects in the 60 image dataset.

6.2.1.3 Size of the Object in the Image

Once the ground truth parameters have been used to create a label image of the scene, it is fairly easy to determine the area of the image which each object occupies:

$$\mathcal{A}_{object} = \frac{\left|\mathcal{Q}_{object}^{true}\right|}{(w \cdot h)} \tag{6.4}$$

where $\mathcal{Q}_{object}^{true}$ represents the set of visible pixels for $object$ using the ground truth scene parameters and w, h represents the width and height of the image. The measure will lie within the range $[0, 1]$ where 0 represents the object not having any pixels in the image (not visible or occluded) and 1 represents the object taking up the entire image. This independent variable is used in conjunction with the percentage of the object occluded to

characterize how much information is necessary for robust recognition. Even if the object is completely visible, the visible area in the image may be quite small. The lower left histogram of Figure 6.7 shows $\mathcal{A}_{object}$ for all 190 objects in the 60 image dataset.

6.2.1.4 Similarity to the Background

All of the images in the dataset are taken of scenes containing objects placed against a solid white background. A common concern in many appearance-based techniques is sensitivity to the background color. Obviously, recognition of an object is impossible if the object is highly similar to its surroundings. For robust recognition, contrast between the colors on the object and the background the object is placed against should exist.

In order to characterize the difference between the object colors and the background, $\mathcal{B}_{object}$, a variant on the Kolmogorov-Smirnov (KS) test is used (Press, Flannery, Teukolsky, and Vetterling 1988). The test is designed to compare two distributions and is computed by finding quadrants in color space where the ratio of the number of background colors to the number of object colors is the largest: measure the regions where the two distributions differ the most. To compute the KS statistic, the distribution of pixel colors for each object are compared to the distribution of background pixel colors.

The lower right histogram of Figure 6.7 shows $\mathcal{B}_{object}$ computed by comparing all 190 objects in the 60 scenes to the background. All of the values computed were significant (the p significance level was ≤ 0.0001). The KS-values are viewed as a distance measure between two distributions. If the observed value is close to one, the two distributions are different (object colors contrast with the background). Values less than one indicate the level of similarity (object colors are similar to the background).

6.2.1.5 Quality of Initial Scene Configurations

One of the key factors likely to influence the performance of RMR is the quality of the initial scene configurations. Since RMR is a refinement algorithm, the algorithm is not capable of recovering from extremely poor starting estimates. Therefore, RMR performance is expected to be heavily dependent upon the initial configurations. These initial configurations are determined based upon a pose indexing algorithm fully discussed in Appendix A.

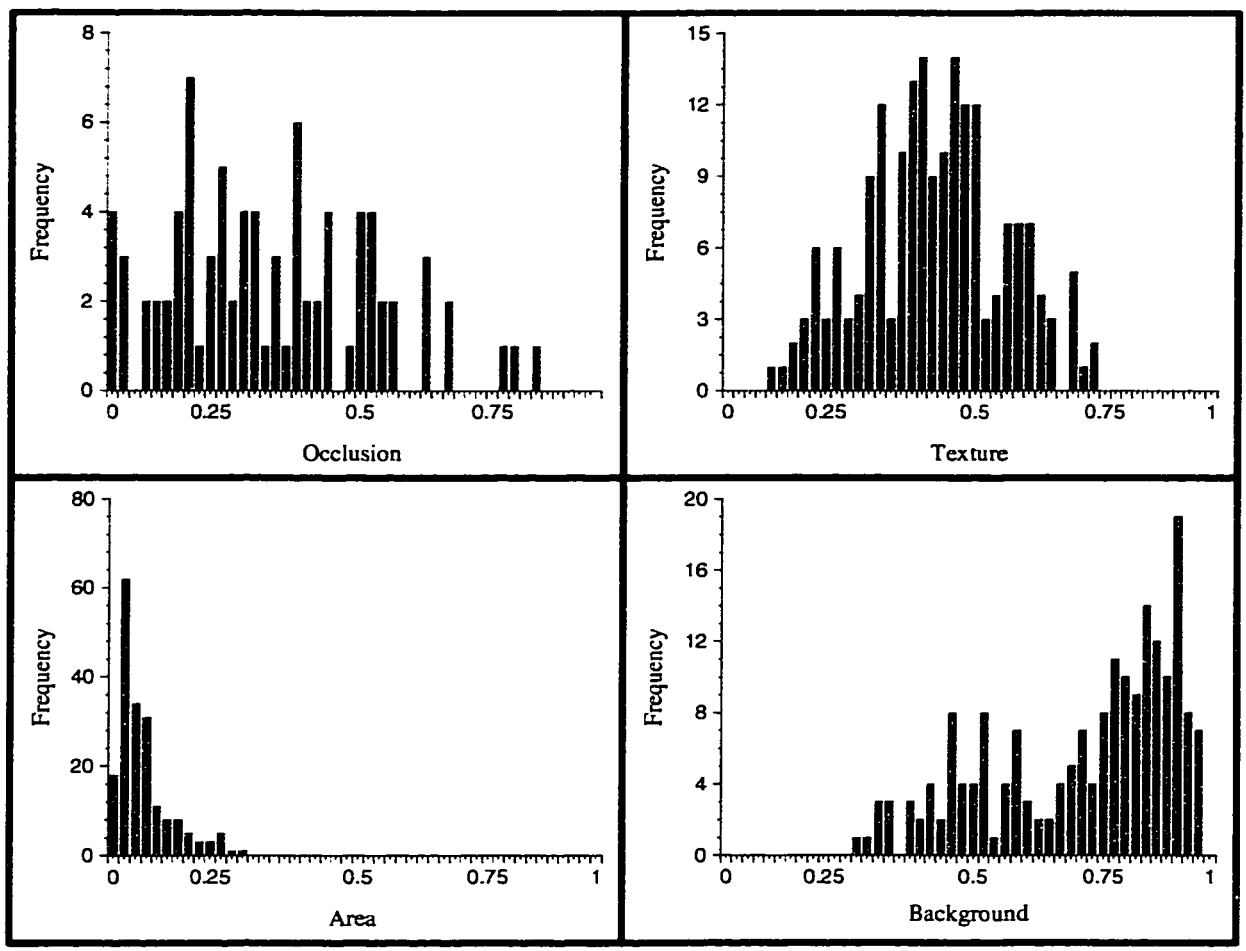

Figure 6.7: The occlusion, texture, area and background independent variables (shown in clockwise order).

Each of the 60 images has been processed by the indexing algorithm, producing a list of the top 25 most likely pose parameters. From this list, scene configurations are created by randomly drawing pose values from each object's hypothesis list. Appendix A demonstrated that the orientation parameters are not accurately predicted by the current hypothesis generation mechanism. To overcome this limitation, a uniform random number is substituted for the orientation parameter provided by the current indexing method. Therefore, the translations are randomly drawn from the hypothesis list and the orientation parameters are drawn from a uniform distribution [0..360]. Figure 6.8 shows signed difference between the best initial pose configurations (lowest RMR error) and the ground truth configurations for each of the 190 objects in the 60 scenes.

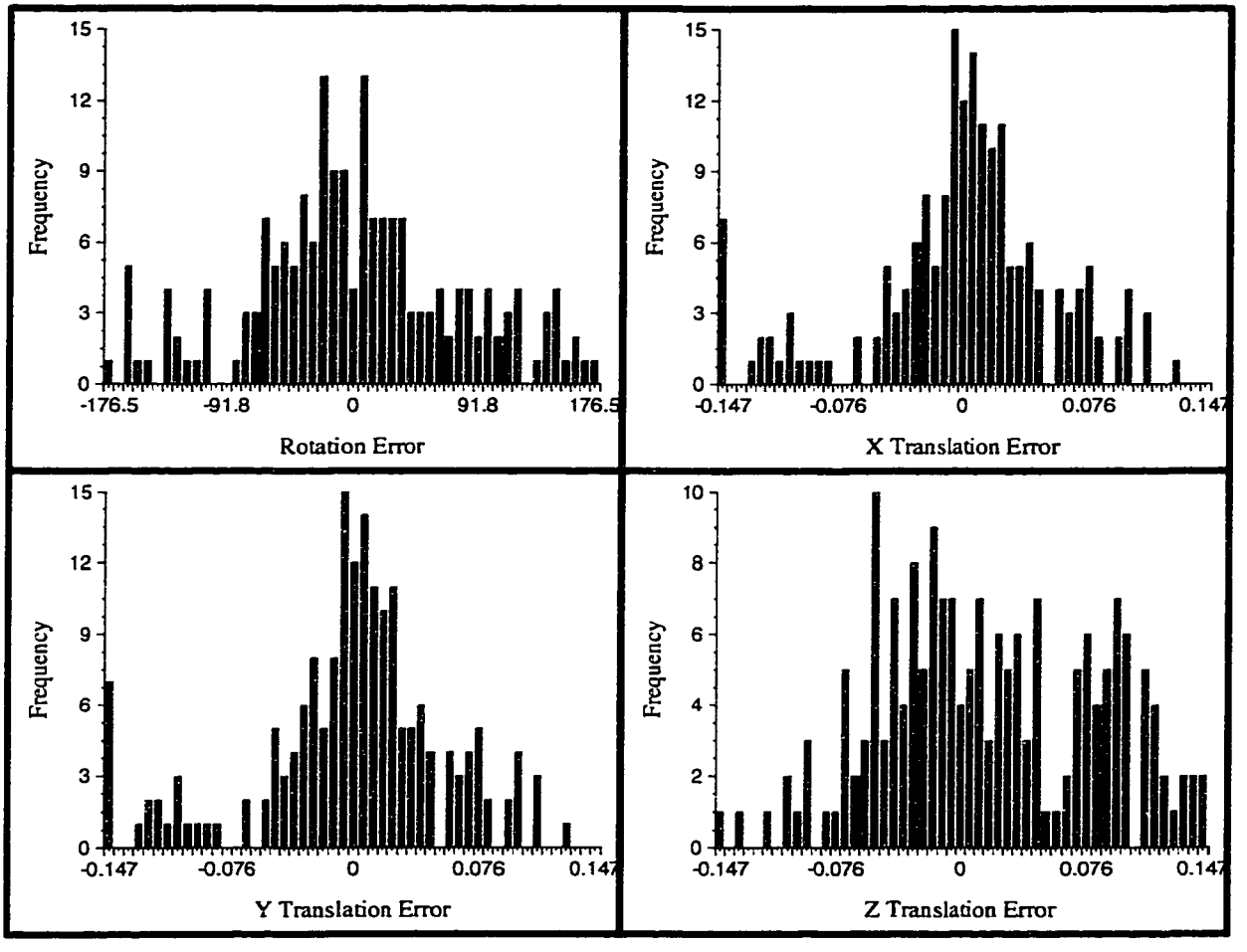

Figure 6.8: The distribution of initial object pose parameters for the 190 objects in the 60 scenes.

Notice that the values in Figure 6.8a do not appear to be drawn from uniform distribution. Remember that the entire population is generated first using the uniform random orientations. The error function is then used to select the best single configuration from that population. This error function introduces a bias which prevents the distribution from remaining uniform.

6.2.2 The Dependent Variables

Dependent variables are measured after 200 trials of RMR have been run. These variables characterize the quality of the recovered scene configuration. The signed difference between the object pose parameters recovered by RMR and the ground truth values are shown in Figure 6.9. These differences represent the subtraction of the recovered parameter from the true settings. These differences show how close RMR placed any given object relative to where it should have been in the scene. The amount of error for each parameter is compared with the initial values provided to the algorithm (shown in Figure 6.8).

Chapter 4 discussed reasons why the difference in $3D$ pose should not be used for comparing scene configurations. However that discussion was within a different context: the goal was to define a single measure describing the difference between two configurations. When working with $3D$ pose values a single measure is difficult to define since the parameters are of different units and types (rotations are in degrees and translations are in meters). For evaluation, we are not concerned with having a single measure and therefore closeness to the ground truth is measured along each of the four parameter dimensions.

6.3 Relating Dependent and Independent Variables

To determine the relationships between the dependent and independent variables, each model in each scene was divided into one of two groups: success or failure. The key question to answer is which of the independent variables most strongly influence the likelihood of failure. For each of the 60 images in Figures 6.1 through 6.6, each model was examined and hand labeled as either a success, ambiguous, or a failure. The success category represents objects that are visually meaningful: the objects are in the correct location in the scene

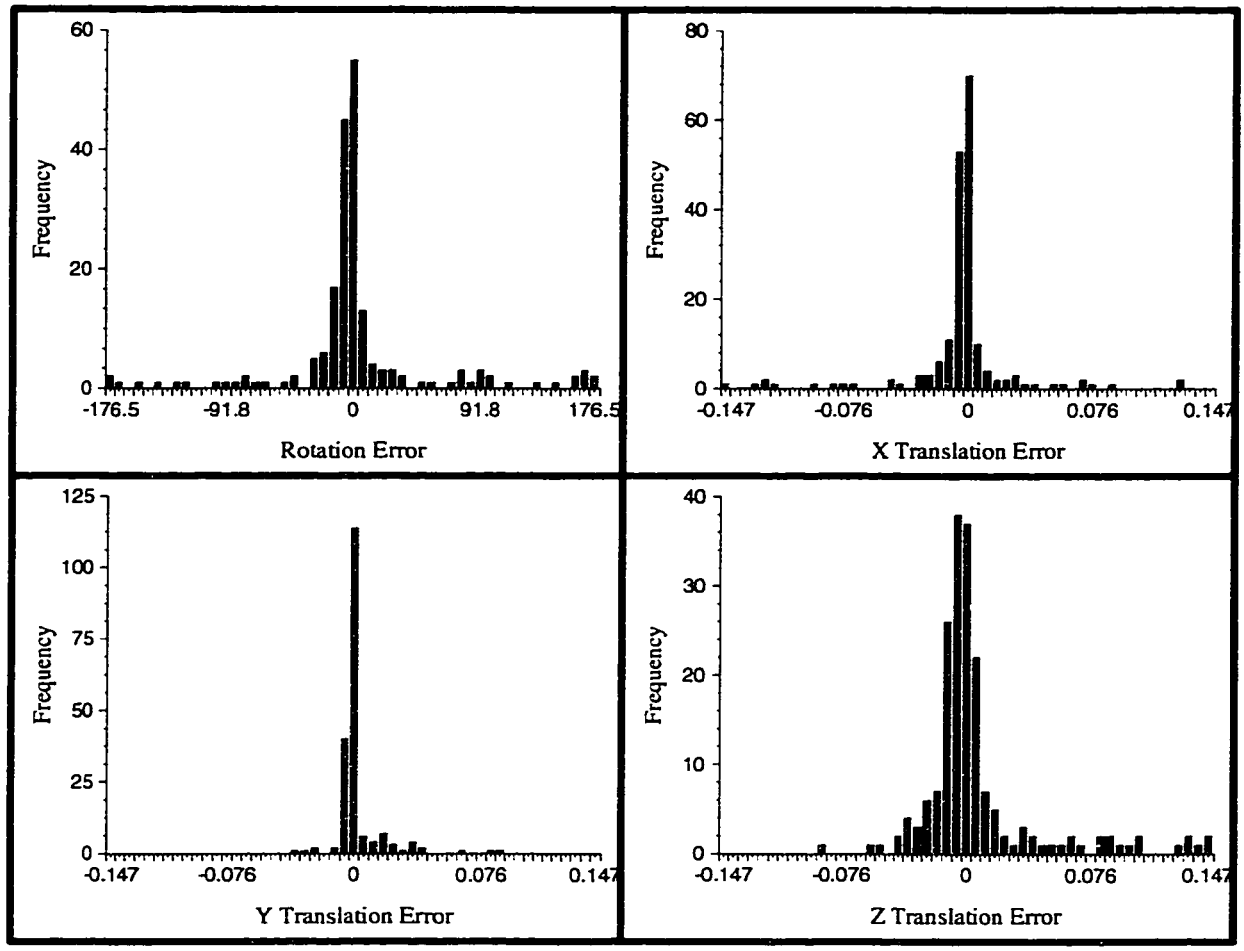

Figure 6.9: The distribution of final object pose parameters for all objects in the 60 scenes.

(within ±0.01 meters) and orientation is nearly correct (within ±15 degrees). The failure cases are quite obvious since the object is usually in the wrong place in the scene.

The ambiguous label is introduced to handle cases where the system appeared to recover the correct scene configuration but object appearance led to the the objects covering a significantly different region in the image from that of the ground truth. For instance, Figure 6.4h shows two cubes both containing red object faces with either one located on opposite sides of the image. The system recovered a viable interpretation of the scene, only the two blocks were swapped (they have the same size and appearance from that viewpoint). Table 6.1 shows the hand labeled object classifications.

6.3.1 Is Failure Related to Object Type?

Table 6.2a shows the contribution of each object to the total number of failures. Notice that the *n-block*, the *s-block*, the *orange-block* the *almond-tea* and the *acuvue* account for 68 percent of the failures. Statistical analysis of failure by object type is accomplished using the ANOVA (same statistic as was used in both Chapters 4 and 5). The results of the test are shown in Table 6.2b. This time, the test is comparing sensitivity to object type and the four recovered (dependent) pose parameters. Both results were significant implying that performance varied both in terms of the object type and the individual pose parameter being recovered. Therefore, the ANOVA test substantiates the hypothesis that RMR performance is sensitive to the type of object being recognized.

6.3.2 Analyzing Success and Failure using the Independent Variables

What is it about the specific object type that causes the failure? Figures 6.10 and 6.11 show the eight different independent variables (occlusion, texture, area, difference from background and initial pose values) separated into those cases where the model was successfully located versus those cases where RMR failed to locate the model correctly. Note that the un-occluded objects are neglected from this histogram but not from the subsequent analysis (there were 87 un-occluded object successes and 16 un-occluded object failures).

To determine whether object attributes are predictive of success, we ran a series of t-tests comparing success and failure distributions for each independent variable. Table 6.3

Object	A02										A03										A04										S	F	A
	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10			
n-block	S	F	S	F	F	F	F	S	S	S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	5	
s-block	S	S	S	F	F	F	F	S	S	F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	5	
o-block	S	S	S	S	A	S	S	A	F	F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	2	2
i-block	S	S	S	S	A	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9		1
blue-block	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	A	A	A	S	S	S	S	S	S	S	S	S	S	F	S	16	1	3
red-block	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	10		
green-block	-	-	-	-	-	-	-	-	-	-	F	S	S	S	S	S	S	S	S	S	F	S	S	S	S	S	S	F	S	S	17	3	
yellow-block	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	S	S	F	S	S	9	1	
orange-block	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	S	S	F	A	S	F	F	S	S	5	4	1
almond-tea	F	F	S	F	F	S	F	F	F	S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	7	
raspberry-tea	S	S	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10		
mac-n-cheese	S	F	F	S	S	S	S	S	S	F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	3	
speed-stick	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	10		
vaseline	-	-	-	-	-	-	-	-	-	-	F	F	S	S	S	S	S	S	S	S	-	-	-	-	-	-	-	-	-	-	8	2	
acuvue	-	-	-	-	-	-	-	-	-	-	F	F	S	F	F	S	F	F	S	F	-	-	-	-	-	-	-	-	-	-	3	7	
blue-stapler	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	S	S	S	S	S	S	S	S	S	10		
red-stapler	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	S	S	S	F	S	S	S	F	S	8	2	
																														Total	141	42	7

Table 6.1: The hand labeled classification of recognition performance. The labels are interpreted as follows: A is ambiguous, F is failure and S is success.

Object	Percent of Failures
n-block	0.12
s-block	0.12
o-block	0.05
i-block	0.00
blue-block	0.02
red-block	0.00
green-block	0.07
yellow-block	0.02
orange-block	0.10
almond-tea	0.17
raspberry-tea	0.00
mac-n-cheese	0.07
speed-stick	0.00
vaseline	0.05
acuvue	0.17
blue-stapler	0.00
red-stapler	0.05

a. Failure Percentages

	Degrees of Freedom	Mean Square	F
Between Objects	16	99046	527 with 16 degrees of freedom ($p \leq 0.001$)
Between Pose Values	3	6984	37 with 3 degrees of freedom ($p \leq 0.001$)
Interaction	48	16511	88 with 48 degrees of freedom ($p \leq 0.001$)
Residual or Random	3944	187	

b. ANOVA

Table 6.2: The contribution to the total number of failures by object type.

			Success		Failure	
Factor	t	p	μ	σ	μ	σ
Occlusion	-1.157	$\leq$0.526	0.276	0.204	0.315	0.198
Texture	-2.187	$\leq$0.030	0.336	0.128	0.501	0.068
Area	0.312	$\leq$0.763	0.067	0.052	0.055	0.045
Background	1.000	$\leq$0.516	0.764	0.168	0.691	0.178
Rotation	-8.588	$\leq$0.001	2.595	72.435	15.724	86.524
X Translation	-0.193	$\leq$0.849	0.005	0.060	0.014	0.082
Y Translation	0.094	$\leq$0.926	0.006	0.025	0.004	0.035
Z Translation	-0.196	$\leq$0.847	0.026	0.070	0.035	0.078

Table 6.3: Using the t-test to compare the independent variables.

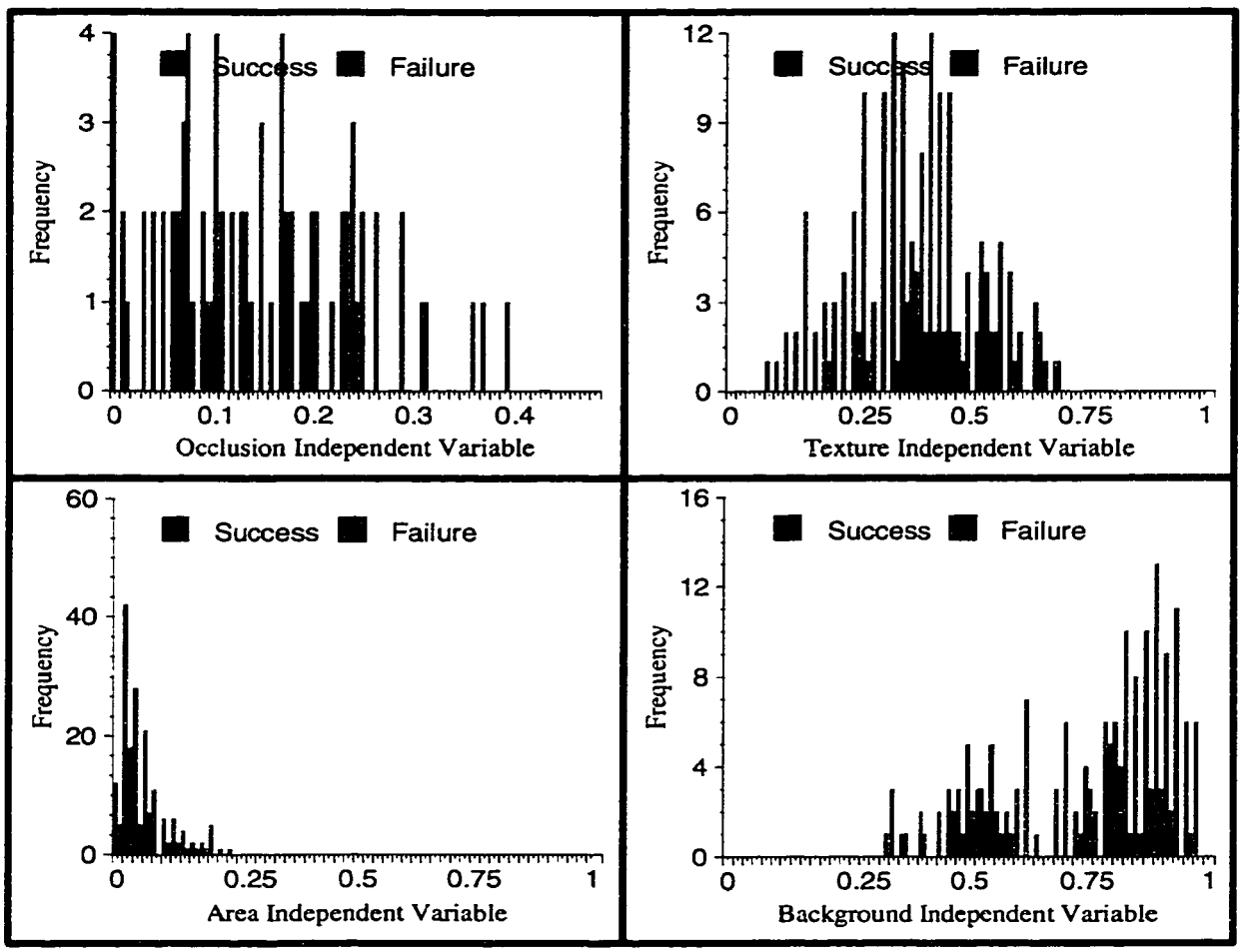

Figure 6.10: Analyzing the failure cases in terms of the independent image variables.

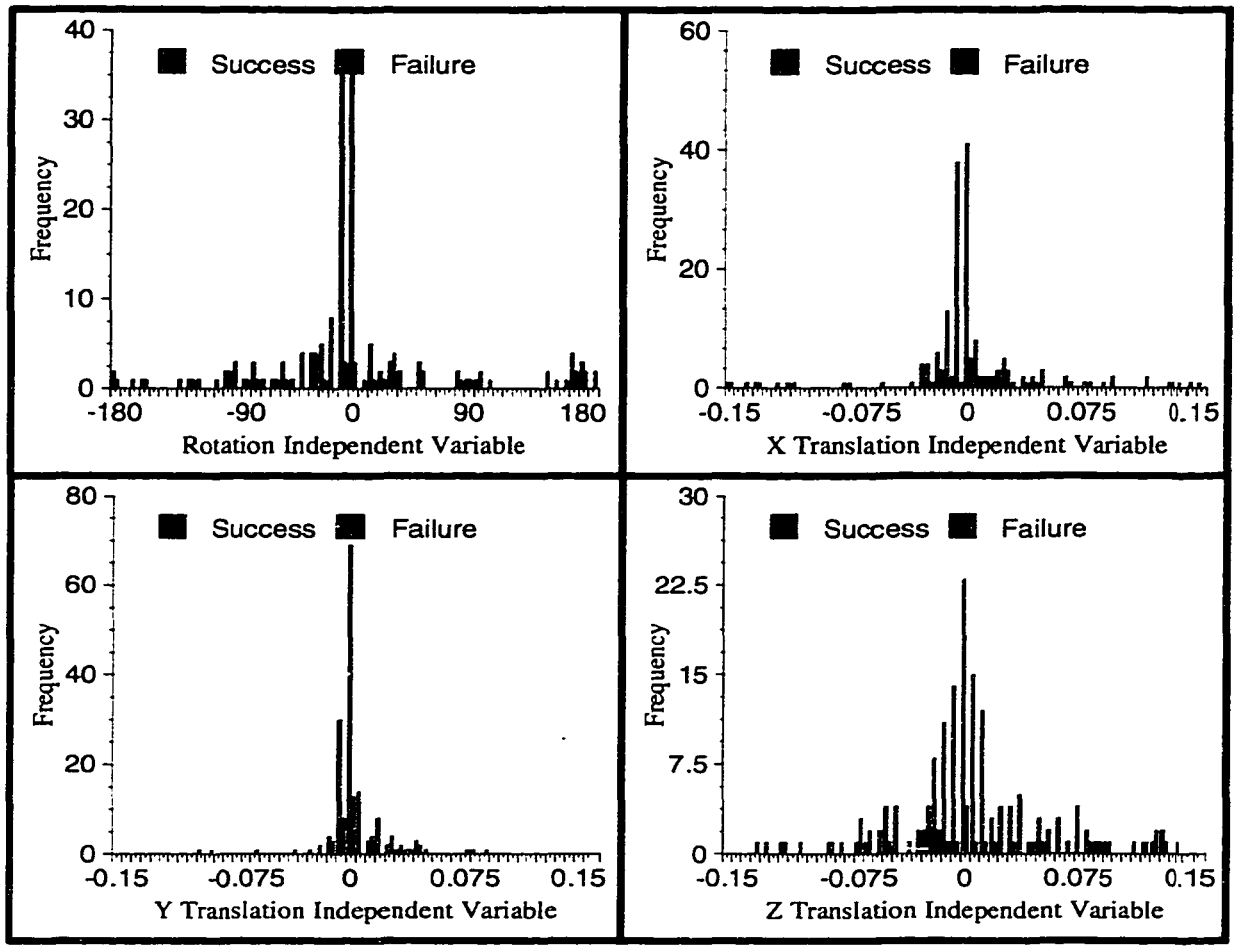

Figure 6.11: Analyzing the failure cases in terms of the independent pose variables. Recall that these pose values are only drawn from the starting configurations.

the results. The test is deemed significant if the corresponding p value is small (≤ 0.001). Based on Table 6.3, several observations are made:

1. The difference between initial rotation values for those objects that succeed and those which failed shows the most significant result. This implies that the RMR algorithm typically fails due to poor initial scene configurations. More specifically, the orientations in the initial populations are not close enough to the ground truth for the local refinement stage (and match error) to recover the correct scene configuration.

2. The t-test for the texture independent variable also indicates a significant difference between the success and failure distributions. This implies that RMR is showing a sensitivity to objects that contain a high amount of texture. The results of this comparison make intuitive sense: if there is too much local variation in the appearance of the object, the error surface will contain many local optima which may be problematic for search. This local optimum hypothesis is supported by both the ranking analysis of the error functions in Chapter 4 and the search analysis in Chapter 5.

3. The ability to recover the configuration of the scene is not highly influenced by either occlusion, the background or the object area in the image. This analysis is promising since scenes with high levels of occlusion, which often occur in real world problems, are successfully solved using our RMR implementation.

6.3.3 Is RMR better than just indexing alone?

The final analysis concerns how well our RMR implementation performed compared to the indexing algorithm. If RMR does not recover better configurations than just using template matching, then there is really no point to running the more expensive search algorithm. Figure 6.12 shows the pose values for the best configuration recovered by the indexing algorithm (before) and the RMR matching algorithm (after). Visually, the RMR histogram has much larger peaks around the 0 mark for each pose parameter. This implies the RMR system is recovering the object pose with a much higher accuracy.

Table 6.4 shows the comparison of the before and after pose distributions using a t-test. The rotation, X, and Z parameters demonstrate significant differences between the distribution of pose values produced by RMR and those produced by using only template matching. Intuitively, part of this comparison makes sense: the indexing algorithm resorted to sampling along the rotation dimension and scaling along the depth dimensions. RMR was able to correct for the inaccuracies along these three parameter dimensions.

The other parameter, the Y translation, was not significantly different. This result also makes intuitive sense: almost all of the 190 objects lie on a common ground plane. Therefore, recovering the height of the object in the scene should be easier since it is the most constrained. Possible avenues of future work could involve examining scenes where the Y dimension is not constrained by the ground plane. Such scenes will provide a better insight into why the Y dimension was better recovered by the indexing algorithm. Overall, RMR is better able to recover the object pose parameters than the simple indexing algorithm.

			Before		After	
Factor	t	p	μ	σ	μ	σ
Rotation	65.410	≤ 0.001	58.732	48.129	12.314	29.676
X Translation	2.214	≤ 0.026	0.046	0.047	0.004	0.006
Y Translation	1.282	≤ 0.549	0.019	0.021	0.002	0.003
Z Translation	2.437	≤ 0.015	0.062	0.047	0.011	0.019

Table 6.4: Comparing the before and after distribution of pose values.

6.4 Summary

The RMR approach has now been demonstrated on 190 objects in 60 images. The analysis was performed in terms of how well the pose parameters for each object were recovered. In addition, the evaluation examined possible causes for the cases where the system failed to recover the correct object configuration.

The RMR system exhibited an unexpected sensitivity to the type of object being located. Furthermore, the system appeared most sensitive to the quality of the initial configurations provided for search and the amount of texture present on the object surface: failures were

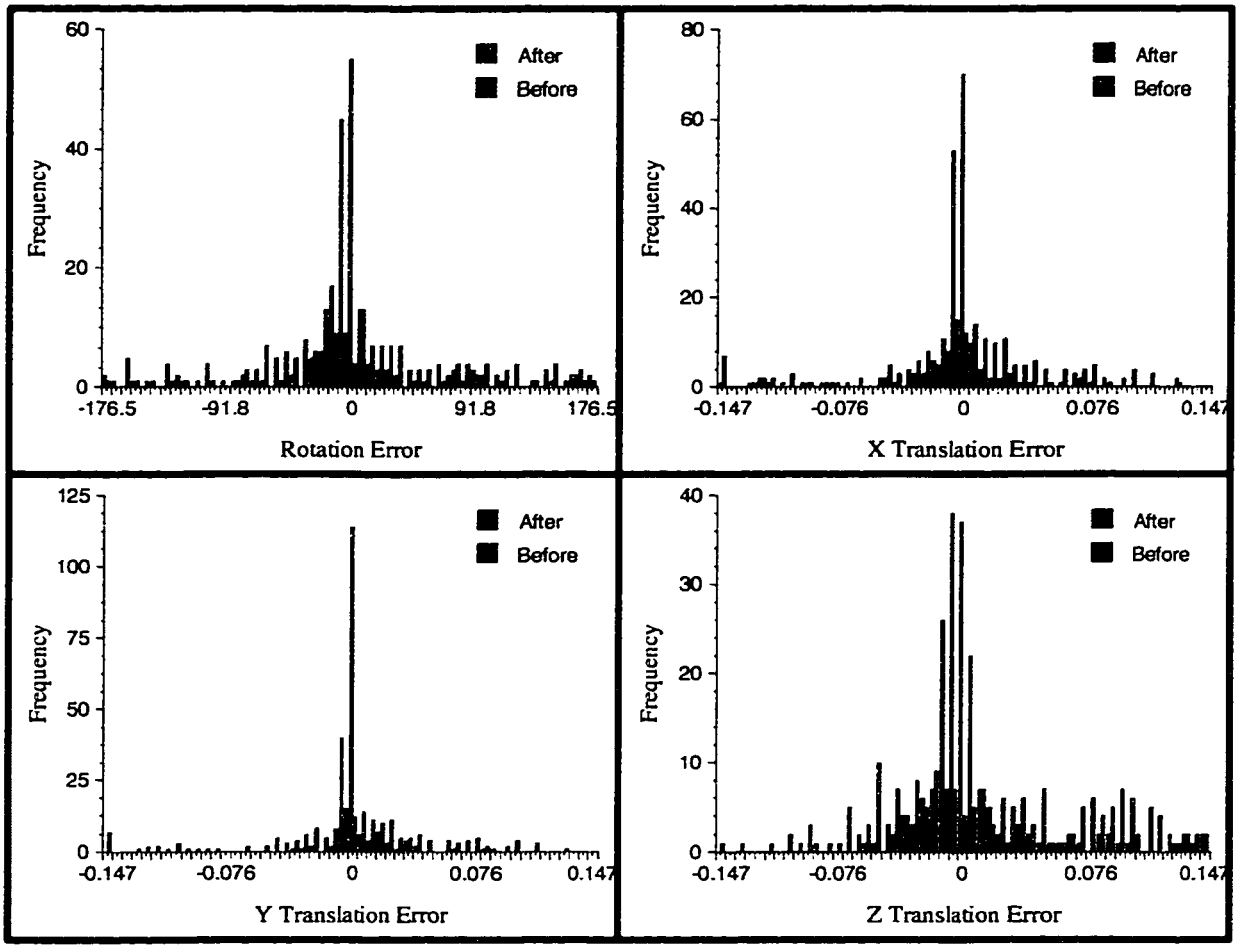

Figure 6.12: The distribution of initial and final object pose parameters for all objects in the 60 scenes.

not caused by object occlusion. Sensitivity to quality of the initial solution could have been predicted from the analysis in Chapter 5: the farther the hypothesized configurations are from the true configuration the lower is the probability of recovering the correct configuration.

Chapter 7

Conclusions

"Now I can't see, I just stare"

- Eddie Vedder

The field of object recognition has been slowly moving towards representations that combine geometry with appearance information. While important work on both techniques is progressing in isolation, the time has come for a merging of ideas[1]. This dissertation represents a vital step forward in the process of combining geometric information with stored object views.

Using computer graphics as the binding tool, both geometry and appearance information are incorporated into the recognition process:

- The appearance information is strengthened because it is based on the underlying $3D$ geometry of the scene.
- The geometry information is strengthened because more complex surface textures are dealt with reliably.

Recognition based on fusing both techniques promises to be more robust than using either method in isolation: appearance techniques cannot extrapolate beyond the stored training views and geometric techniques have difficulties representing objects with complex textures.

The stage has now been set for a much greater reliance upon sophisticated generate and test algorithms which use computer graphics for computer vision. In the purest sense,

[1]These observations are based on the Workshop on Integration of Appearance and Geometry held at the 1999 IEEE conference on Computer Vision and Pattern Recognition.

RMR is an example of a generate and test paradigm. The generate and test approach has the added benefit of being a proven technology pervasive throughout research in the field of Artificial Intelligence (AI). Based on this thesis, the technique has now been shown as a viable method for incorporating rendering (in terms of both geometry and appearance) into an object recognition algorithm.

This dissertation has also provided a solid set of methods for evaluating the performance of both different image comparison algorithms and different scene refinement techniques. While the specific error functions and search algorithms found to work best are limited to the two selected domains, the underlying evaluation methodology extends beyond the limits of this thesis. The following steps for evaluating a computer vision algorithm, which were used throughout this thesis, are invaluable for demonstrating the merits of an implementation:

- List a set of desirable performance criteria.
- Introduce the algorithms to be compared.
- Predict performance of the algorithms in terms of domain and problem specific knowledge (reinforced with pilot studies).
- Evaluating the algorithms on a series of sensor images.

The time has passed when algorithms can only be evaluated theoretically without a solid demonstration on actual sensor imagery. Furthermore, synthetic imagery may be used to both develop the performance criteria and predict performance, but extensive evaluation must eventually be performed on real sensor data. Using this evaluation methodology, a series of domain specific conclusions were drawn. Here we highlight a few of the more interesting discoveries:

1. Success or failure of the RMR algorithm is not a function of object occlusion in the scene. Since RMR exploits information about all of the objects present, it is able to account for missing object pixels by accounting for the cause of the occlusion. Therefore, explanations of an object can be found even when very little of the object is visible: instead of simply not finding a model feature, the location of other objects

in the scene account for feature absences. Thus, the entire scene context constrains the identity and location of each object.

2. The goals of rendering a prediction are different from those for rendering images to be viewed by a person. Early rendering implementations took advantage of special hardware accelerators. These hardware accelerators were designed to quickly display images on a console and not store the information directly into main memory. As such, rendering with this hardware required additional computation to retrieve the images into memory where they could be compared. Moving the images greatly decreased the rate at which images could be generated and compared. Therefore, a software rendering package was used to generate texture mapped predictions directly into main memory. Still, rendering the scene tends to dominate the total computation time of a single error evaluation. Future work should investigate methods for reducing the rendering time and thereby making the entire RMR approach more computationally tractable.

3. A combined error function based on both color and edge features does better than an error function based on either feature alone. Color features measure regions of information while edge based measures are highly local. The combined measure fuses information about both distinct events to make a more robust error function.

4. Evolution Strategies exploits a population of solutions in a manner most appropriate to the pose based indexing discussed in the Appendix: better than both the Simplex and Hill Climbing. Evolution Strategies draws pose parameters from a population of solutions. Exploiting the entire population leads to an increasingly strong set of pose parameters.

7.1 Future Work

The evaluation and empirical analysis presented in this dissertation should provide a baseline for future work: the test suites can now be used to to assess the performance of any subsequent modifications to the error functions, the search algorithms, or the entire RMR

implementation. The current results can be compared against those produced by any future versions of the RMR algorithm to determine the merits of the modifications. Specifically, there are three areas to be addressed:

1. More advanced error functions based on complex feature spaces (such as points, lines and regions) may do much better at ranking configurations than the proposed measures based solely on pixel level measurements. In addition, combination mechanisms based on Bayesian statistics may produce a better error measure by combining information in a more principled manner.

2. Other search algorithms exist which could be utilized by RMR; including simulated annealing and the messy genetic algorithm. Also, very little parameter tuning was done to each of the search algorithms evaluated. Extensive studies are needed to find the parameters which increase performance on the given test problems. For instance, Evolution Strategies has several mutation rate parameters. Changing their settings will change overall recognition performance.

3. The indexing algorithm (discussed in the Appendix) located single objects in the scene without considering occlusion. Based on the results achieved, performance was sensitive to the quality of the rotation parameter provided. Obviously, introducing a stronger indexing algorithm more capable of dealing with occlusion will increase the quality of the RMR results.

Again, any of these modifications must be demonstrated using the methodology discussed throughout this dissertation.

Appendix A

Generating Scene Hypotheses

"When the only tool you have is a hammer, you tend to treat everything as if it were a nail."

- *Abraham Maslow*

The RMR algorithm described in this dissertation provides a method for partial 3D scene interpretation. Using a model library, graphical rendering, sensor imagery, and sets of scene configurations, RMR produces a final explanation of the scene. A common question about the described technique often arises: Where do the sets of initial configurations come from? This appendix presents a technique, known as *indexing*, used to generate the scene configurations used in Chapter 6.

Indexing in the presence of both occluded and textured objects is difficult, and solving this problem is well beyond the scope of this dissertation. However, some form of bootstrapping is required for the evaluation of the entire RMR system. For this reason, an admittedly limited, yet adequate, indexing algorithm was implemented. The algorithms developed do not contain any special occlusion reasoning. Therefore, they should be expected to perform poorly on the examples in the test set which contain object occlusion. Future work will entail examining methods of single object indexing in the face of such occlusions.

A.1 Object Detection and Pose Indexing

The goal of indexing is to determine which sets of models from a stored model library are most likely present in a scene (Grimson 1990b). Formulated in this manner, object indexing is an extremely difficult task: all known objects in the database could be visible at any position and orientation. Extensive work has already been done in the field of object indexing. Three of the more well known techniques are:

Geometric hashing constructs hash tables based on the geometry of the models in the object library (Lamdan and Wolfson 1988; Grimson and Huttenlocher 1988). Given a novel set of features from a sensor image, a hash function indexes into the table to determine the object most likely present. Two of the major problems with the technique are occlusion and clutter. In the case of clutter, non-object points will be used in the hashing function causing erroneous votes for objects which are not present. In the case of occlusion, the number of object points used will decrease (since fewer and fewer object points will be visible). Therefore, as the ratio of object clutter to occlusion increases, the reliability of geometric hashing decreases.

Histogram intersection has been proposed by Swain as a method for indexing based on object color (Swain 1990). Swain's technique requires a color training image of each object. From this training image, a histogram is created. Histogram intersection is used to compare a histogram from a novel image to all of the training histograms. A set of possible model types are returned ranked by the intersection metric. Swain's method tends to not be robust in the face of occlusion, or out-of plane object rotation (Mata, Marik, and Kittler 1995).

Template probing exhaustively compares a set of test points, called probes, to a novel image (Bevington 1992; Li, Ferdousi, Chen, and Nguyen 1986). Each probe is formed from off-line analysis of the object at a specific orientation with respect to the camera. Unfortunately, the template probing technique does not scale well. Each time a new model is added to the object library, an additional set of templates are created. At indexing time, all of the templates must be applied to each image. Since execution time

depends on the number of templates, adding objects increases computation time. This is different from geometric hashing, where indexing time is a function of the number of tuples present in the data, not how many objects are present in the database.

For this dissertation, the larger indexing problem is not being solved. The current hypothesis generation mechanism knows the number and type of each object present in the scene prior to indexing. We refer to this limited form of indexing as pose indexing. For instance, if the scene contains the *curad*, *sudafed*, *opti-free* and *toms-of-maine* objects, only those objects are considered.

The simplified indexing task is broken into two stages. The first stage, referred to as *object detection*, locates regions in the image most likely to contain an object. Object detection is a conceptual form of segmentation where the object pixels are segmented from the background pixels. The second stage, referred to as *pose indexing*, takes the detection result and generates a list of possible object positions and orientations. When pose indexing is complete, each object will have a list of ranked pose values.

Both template probing and geometric hashing were considered for bootstrapping RMR. Since our previous work (Stevens and Beveridge 1997; Stevens and Beveridge 1998) had shown the benefits of using template probing, this method was implemented for providing initial configurations to RMR. Future work will examine other possible techniques to determine if template probing is the best technique available for these domains. Figure A.1 shows a brief overview of the algorithm. The details of each component are presented in the following sections.

A.2 Detection based on Color Decision Trees

Our previous work demonstrated the strengths of using color to identify regions of an image most likely to be object (Beveridge, Draper, Stevens, Siejko, and Hanson 1997). Based solely on color, a focus of attention algorithm was able to substantially reduce the amount of extraneous information in an image. The success of the method was not hindered by the fact that the objects being detected were camouflaged vehicles in natural terrain. The existing technology was modified slightly for the current blocks-world and products-

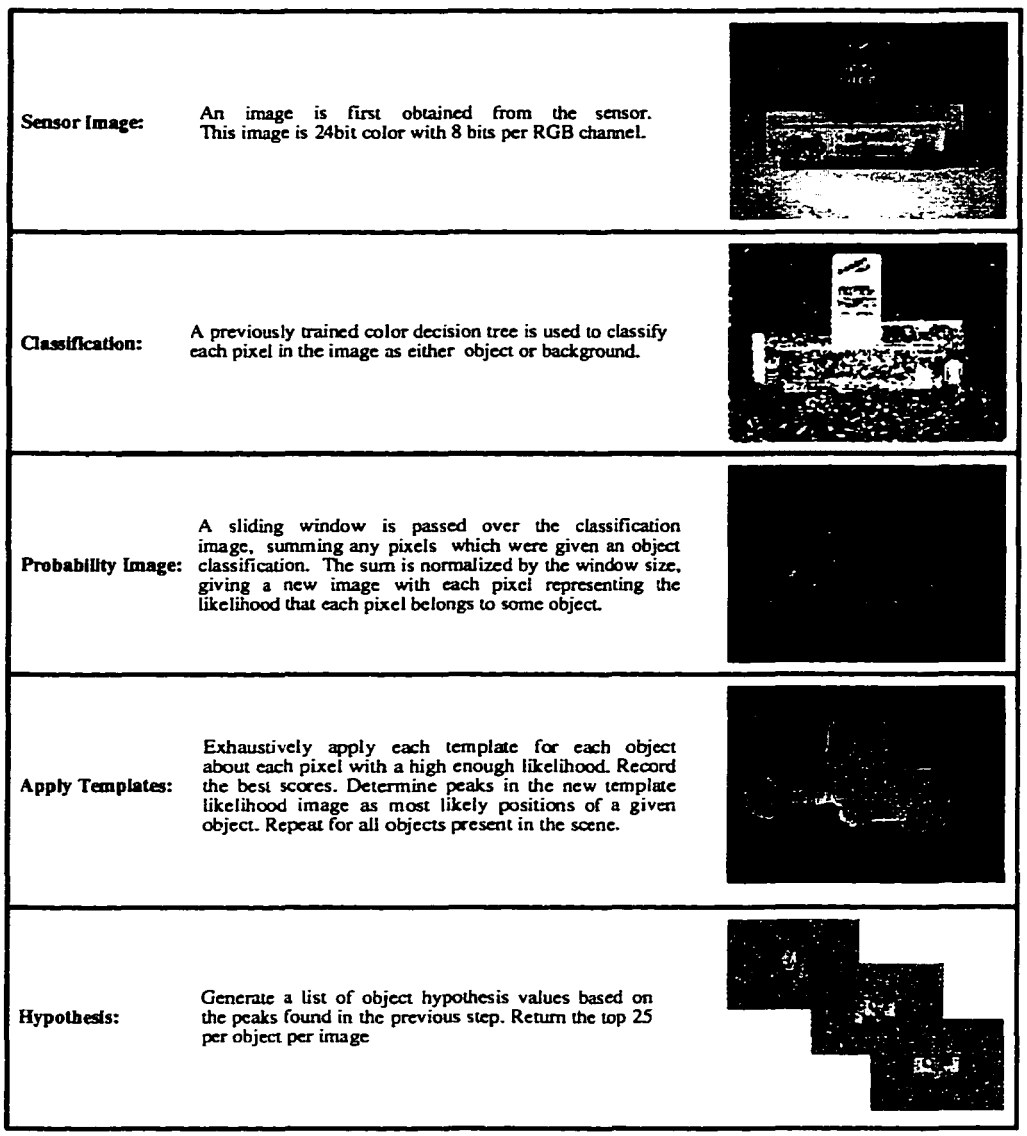

Figure A.1: Overview of the Indexing algorithm

world domains. The modifications were due to the fact that these domains are simple in comparison (all objects are against a white background), and a larger number of training and test images are available. For this thesis, an explicit set of training data already exists: those images used to form the object texture maps.

For color detection, a decision tree is used to classify pixels as either object or background. A decision tree is a binary tree, where each leaf node contains a classification label of either object or not object. Color pixels are fed into the tree at the root node. At each non-leaf node, a decision of whether to take the left or right branch is made. Once the leaf node is reached, the classification label is returned as the appropriate label for that color.

A.2.1 Generating a Color Decision Tree

In order for the decision tree method to be appropriate, a set of hyper-planes must exist which mostly segregate the positive training instances from the negative ones. For our previous work, the algorithm used to construct these hyper-planes was stochastic: given the same set of training data, the algorithm produces different hyper-planes or tree structures (Draper, Brodley, and Utgoff 1994). A deterministic approach is used to train the decision trees used for RMR. As such, the algorithm is simple and easy to both understand and implement. If the simplistic assumptions used to create these decision trees are not acceptable for other domains, more robust non-parametric algorithms are available (Draper, Brodley, and Utgoff 1994; Quinlan 1986; Breiman, Friedman, Olshen, and Stone 1984).

Each non-leaf node in the tree can be viewed as a plane in color space represented by four values (x, y, z, d). Given an RGB value, the distance from the color to that plane can be determined:

$$dist = Rx + Gy + Bz + d \tag{A.1}$$

if $dist$ is positive the left branch of the node is traversed, else the right branch is taken. The set of weights, or plane equation values, are determined based upon the set of training instances provided. Each training instance consists of a RGB color labeled as either positive (object) or negative (background). The positive instances are randomly drawn from the imagery used to create texture maps of the various objects: the pixels under the projection

of the object into the training image. An equal number of negative instances are randomly selected from the remainder of the non-object pixels.

The goal of training is to compute a set of plane weights which best partitions the set of training examples into two groups: each group should contain only instances of the same class (model or background). To find the plane weights, the RGB mean of the positive instances and the mean of the negative instances is computed. The plane normal is the vector from the negative mean to the positive mean. The final value d is set so the origin of the plane is halfway between the two means.

To construct the entire tree, the set of training instances are provided and the plane weights are constructed for the root node. Next the weights are used to divide the set into two groups based upon the sign of the distance between each color and the computed plane (these two groups will contain both object and background instances). The group with positive distances is used to construct the right branch, and the other group is used to construct the left branch. Training continues in a recursive manner until either the set of examples at the current node become homogeneous, or there are less than 10 instances.

A.2.2 Increasing Performance with Multiple Trees

The above training technique will produce a single tree for a specific set of training instances. The hope is that the constructed tree will generalize beyond what information is provided in training: the tree should correctly label similar colors not given in the training set. However, it is not possible to guarantee the optimal tree has been constructed. To improve the classification rate (the number of colors correctly labeled as object or background) of non-training color instances, a technique known as boosting (Breiman 1994) is used.

Boosting reduces the number of incorrectly labeled pixels for a given tree by averaging performance over a collection of trees. To apply boosting here, 25 trees were created, each with a different set of randomly chosen training instances. Voting is then used: the classification of a given color is determined for each tree, and if the color receives more object labels than background it is labeled as object. Otherwise, the color is labeled background.

A.2.3 Converting a Decision Trees into a Lookup Table (LUT)

The decision tree is a poor representation to use for on-line color detection. For each pixel in the image, the tree must be traversed until a leaf node is found. Traversing this tree is very expensive due to the large number of multiplications performed at each non-leaf node. In addition, duplicate colors are bound to exist in the image causing unnecessary computation.

To reduce computation overhead, a lookup table can be formed. The lookup table results from passing every possible color in the discrete color cube through all of the trees and determining the most frequently associated label. Once the LUT is constructed, a novel image is classified by using the color at each pixel as an index into the lookup table. Figure A.2 shows three views of the LUT used for the products-world domain. The first two views are looking down the red and blue axis respectively and the third is along the cyan to red axis. Only object labels are shown with the color representing the actual RGB color in the table. For clarity, every $10th$ color is shown.

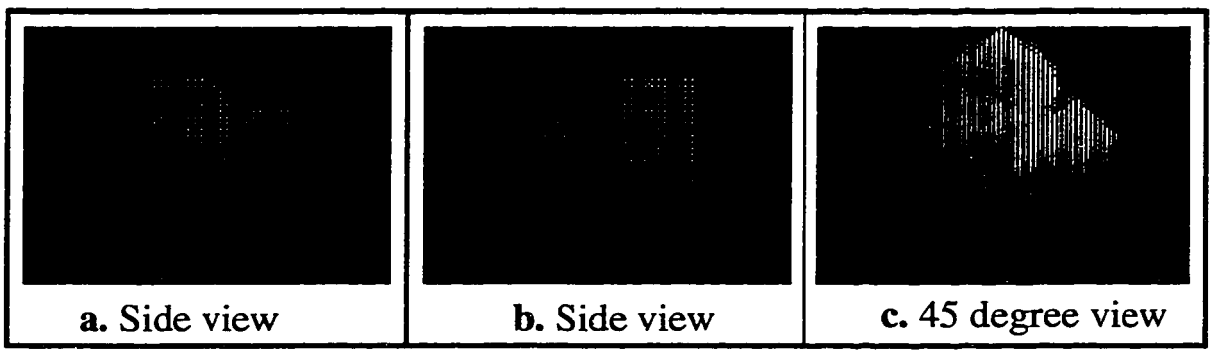

Figure A.2: Three views of the pixels as classified as object in the LUT for the products-world domain.

A.2.4 Results on Example Datasets

Once a lookup table has been created for a dataset, it can be applied to a novel image. The first step is to take every pixel in the image (Figure A.3a) and determine whether the color is object or background (Figure A.3b). Next, a sliding window is passed over the image. The classification labels of all pixels in a window are summed (a 1 represents object, a 0 represents background). The average represents the likelihood that the pixel is object. Figure A.3c shows the likelihoods using a window area of 900 pixels.

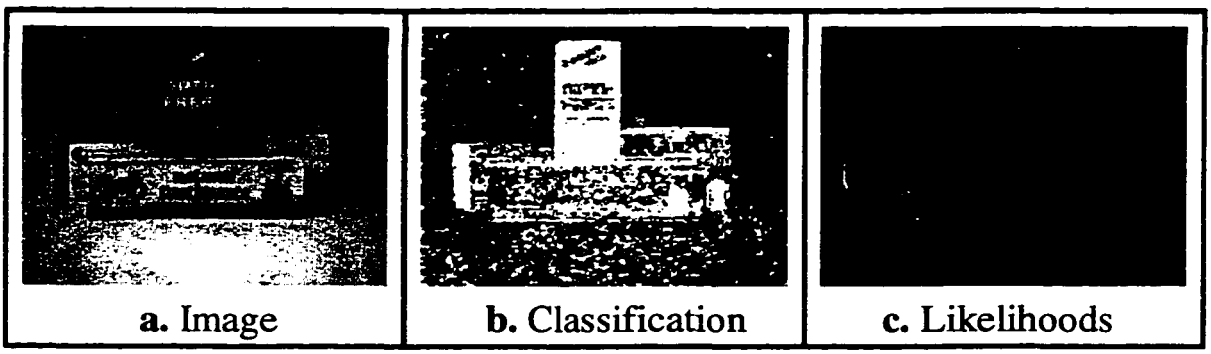

Figure A.3: Applying a lookup table to an example problem.

The lookup tables have been applied to all of the images presented in Chapter 3. Typically, two different dependent variables are recorded to evaluate performance. The first one is a true positive: the percentage of time the true object pixels are classified as object. The second rate is a false positive: the percentage of time the true background pixels are classified object. Figure A.4 shows the true positive rate plotted against the false positive rate for both domains. The goal of any classifier is to have a high true positive and a low false positive rate (the upper left corner of the graph). Notice that the classifier for the blocks-world is doing extremely well (in terms of false positives) compared to the classifier for the products-world domain. We offer one of the many possible intuitive explanations for this behavior: the products-world objects contain a wide range of white colors on the object surfaces. These colors often appear very similar to the actual background colors. In these cases it is almost impossible to build a decision tree which can completely separate the object from the background.

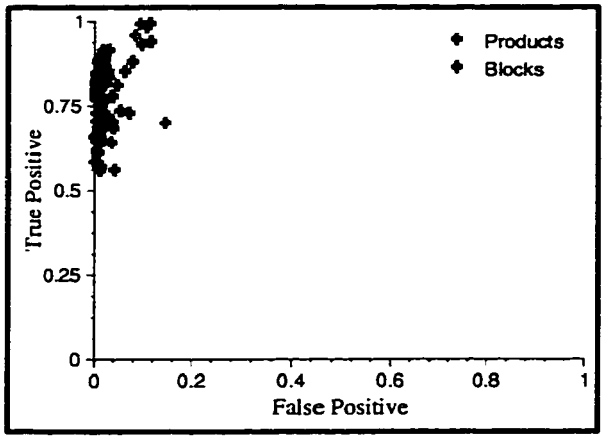

Figure A.4: The false positive and true positive percentages plotted for all of the problems in the blocks-world and products-world domains.

A.3 Pose Indexing

In order to use template probing for pose indexing (forming a set of hypothesis about the position and orientation of an object in the image), templates for each object must first be constructed. Each template represents the given object at a specific orientation with respect to the camera. The templates are composed of a series of probe test points formed from off-line analysis of the underlying object appearance and geometry.

Pose indexing begins after the color detection algorithm has provided a list of the most likely object pixels. About each of these pixels, the templates are compared to the surrounding region in the image. The best matching templates are retained in a hypothesis list. After all of the object pixels have been processed, the best set of pose parameters for each object are used to construct scene configurations for RMR.

A.3.1 Creating a Probe Set

A probe consists of a $3D$ point in camera coordinates plus an expected gradient value. How well the probe matches the underlying image data can be determined by projecting the $3D$ point onto the image plane using the known camera geometry and comparing the expected and observed gradients. If the probe matches well, then it lends support to the belief that the object is present. If the set of probes for a template all match well, there will be a

high likelihood that the object is present in the scene at that location and orientation (each template represents a single object orientation).

A set of probes are generated based on both the $3D$ object geometry as well as the corresponding texture map. Since we have only a 4 DOF problem (using the modified GPC discussed in Section 3.2.1 of Chapter 3), only 2 DOF effect template generation: a rotation about the y axis and the distance of the object from the camera. Variation in rotation parameters is handled by sampling the object viewing angle at every 5°. This produces a set of 72 templates for an object at a fixed depth. To handle changes in depth, the $3D$ probe point is scaled to a variety of depths during on-line indexing. Thus each object is represented by exactly 72 templates.

The first step in creating a template is to produce a rendered image of the object at a fixed distance and orientation with respect to the camera. Figure A.5b shows the an initial rendering of the *curad* object shown in Figure A.5a. This is the first rendering in the series of 72 and represents the object with a 0° rotation around the y axis. The background has been set to black. When selecting probe points, only object pixels are selected (those which have non-zero values for all three RGB bands).

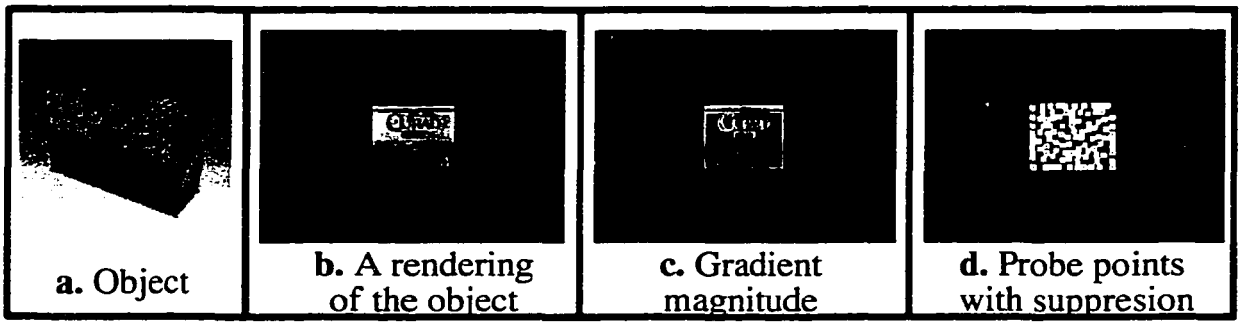

Figure A.5: Sampling the view sphere to produce a set of renderings for each object which are then used to generate probe points.

To select probe points from this image, the gradient magnitude at each pixel is first computed. The same technique as was used to generate these features for an error function (see Section 4.1.1.8 of Chapter 4) is used to estimate the gradient magnitude for each pixel in the rendering (see Figure A.5c). An iterative algorithm is used to select the best probe points from the gradient image. First, the object pixel with the highest gradient magnitude

is selected. A suppression algorithm is used to mask out, or remove, any neighboring pixels from future consideration as probe sites. Non-maximal suppression is necessary to ensure the probes come from diverse regions of the object, and do not cluster around areas with extremely high gradient.

Probe points are continually selected from the image using non-maximal suppression until a total of 75 probe points have been determined. Figure A.5d shows the pixels selected in red along with the suppression region shown in yellow. For each of the selected pixels, the $3D$ point in camera coordinates is recovered. This point plus the gradient magnitude is recorded as a probe in the current template. The entire process is repeated 72 more times for each desired object rotation.

A.3.2 Brute Force Template Application

Given a novel image, the indexing algorithm applies all templates about each pixel having a high enough likelihood (greater than 0.5) of being object. Thus the quality of the color detection algorithm will determine how many pixels in the image will be processed. Since the templates are created at a fixed distance, each template is scaled to 10 different depths and applied. At each depth about each pixel, each $3D$ probe point in the template is projected into the image. The gradient under the each projected probe point is compared against the gradient value stored during training to form the match score for the template:

$$T_j(x,y) = \frac{\vec{EG}_{x,y} \cdot \vec{OG}_{x,y}}{\left|\vec{EG}_{x,y}\right| \left|\vec{OG}_{x,y}\right|} \tag{A.2}$$

where $\vec{EG}_{x,y}$ is a vector of expected gradients for each probe point, and $\vec{OG}_{x,y}$ is a vector of observed gradient values constructed by projecting each $3D$ probe point into the image and recording the computed gradient. $T_j(x,y)$ represents the score of template j at pixel x,y. The likelihood for each pixel is the max template score across all of the templates applied:

$$T(x,y) = \max T_0(x,y) \ldots T_N(x,y) \tag{A.3}$$

A new likelihood image is formed, where now the values represent the belief a certain object at a specific orientation is present at a given pixel. Figure A.6 shows an example image of the likelihoods produced by template matching.

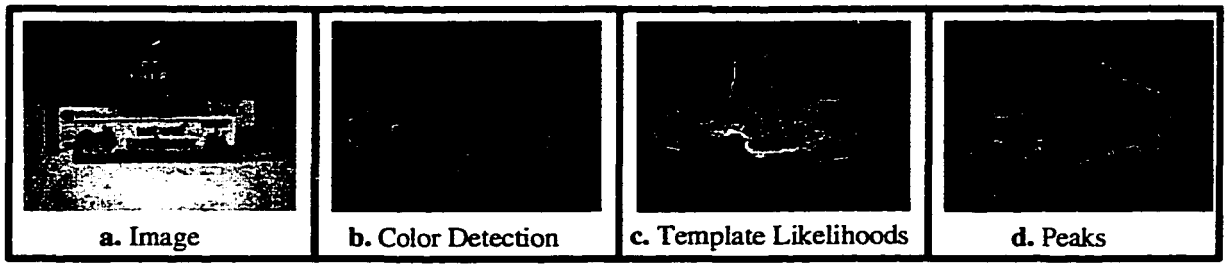

Figure A.6: Exhaustively applying the templates to those pixels deemed worthy by the color detection algorithm.

The next step is to locate peaks in the template likelihood image. A fairly simple peak detector was implemented (Ross and Beveridge 1998). Peak detection begins by first assigning a $2D$ vector to each pixel based on its $8-$*connected* neighborhood. This vector gives the discrete direction towards the largest neighbor. For instance, if the current pixel has the largest value out of the 9 pixels examined, the vector will be $\{0,0\}$. If the pixel to the immediate right is larger, the vector will be $\{0,+1\}$.

A voting image is then formed to measure which pixels are the highest peaks in a local neighborhood. At first, each pixel has a vote of zero. Every pixel in the image is then examined. The vector at a given pixel is added to the current position, producing a new current pixel. The process is repeated until movement along the vector chain does not change the current pixel. The vote at the final destination is incremented. The result is a list of peaks in the template likelihood image. Figure A.6d shows the peaks detected.

The templates for the current object are then re-applied about the top 100 peaks. For each peak, the top 5 templates (based on the score from Equation A.2) are are insertion sorted into a list of the 25 best object locations for the entire scene. After all of the peaks have been examined, the list of the 25 best object configurations are returned as the most likely locations of the object in the scene. The process is repeated for each additional object known to be in the scene. Figure A.7 shows the top 25 configurations determined for the *curad* object in the working example (Figure A.6a).

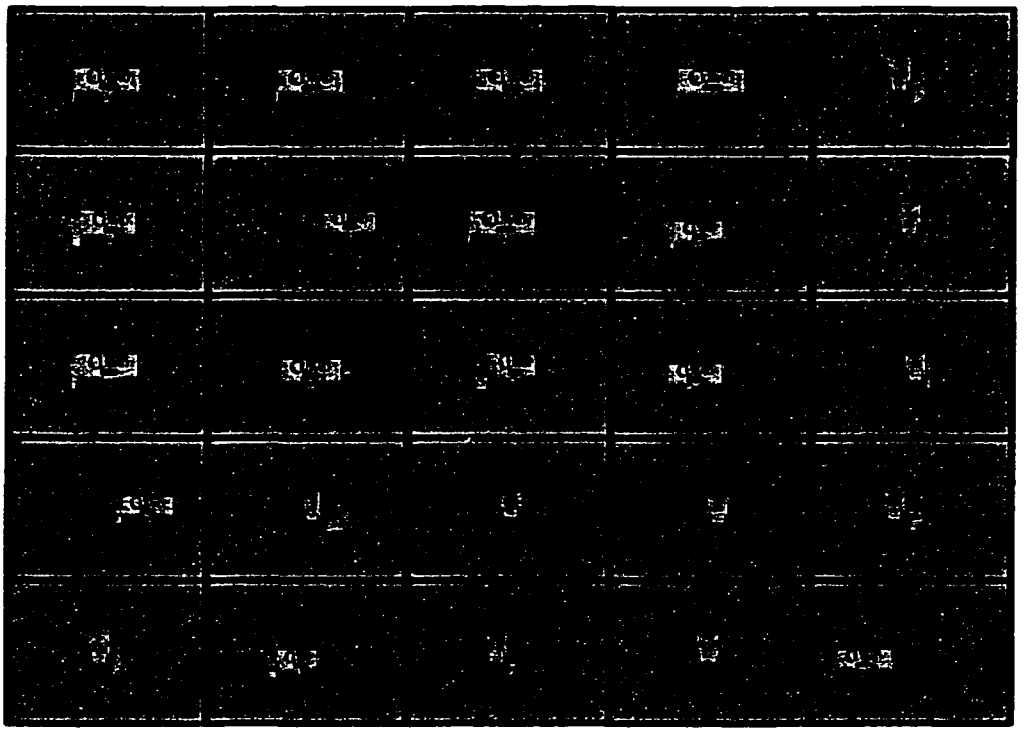

Figure A.7: The top 25 configurations for the *curad* object.

A.3.3 Results on Example Datasets

The hypothesis cuing algorithm has been applied to the 60 images used in Chapter 6. The best measure of indexing performance is in terms of the adequacy of the algorithm for bootstrapping RMR. In order to analyze the performance of the indexing algorithm independent from RMR, four different groups were formed out of the hypothesis lists for each object in all 60 images. Since each image has a total of 25 ranked hypothesis, there are a total of 2,000 hypothesis per object across all of the images. This set of object hypothesis is partitioned into the single best hypothesis per scene (Figure A.8), the top five hypothesis per scene (Figure A.9), the top ten hypothesis per scene (Figure A.10), and the entire collection of hypotheses (Figure A.11).

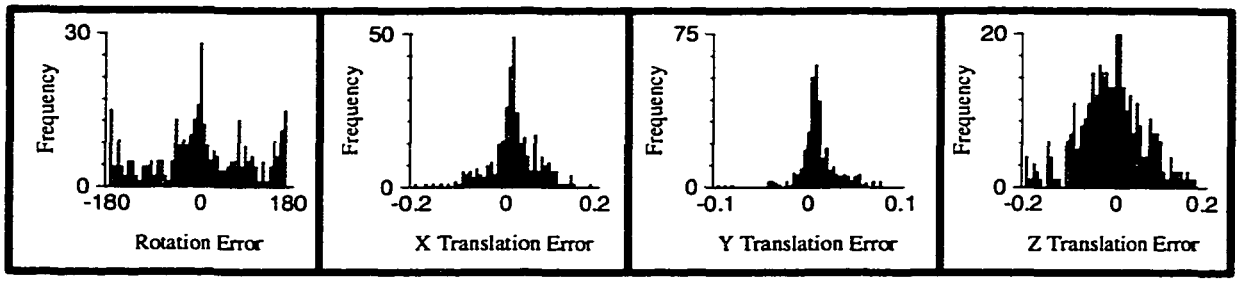

Figure A.8: The signed difference between the ground truth parameters and the pose values of the top hypothesis per object per image.

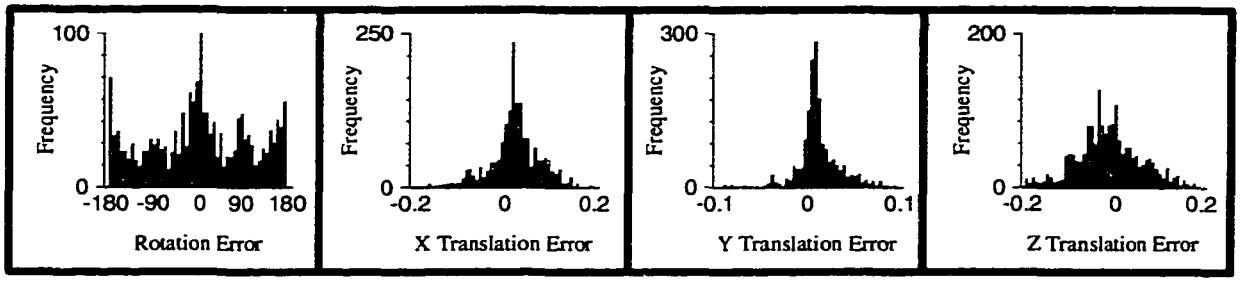

Figure A.9: The signed difference between the ground truth parameters and the pose values of the top five hypothesis per object per image.

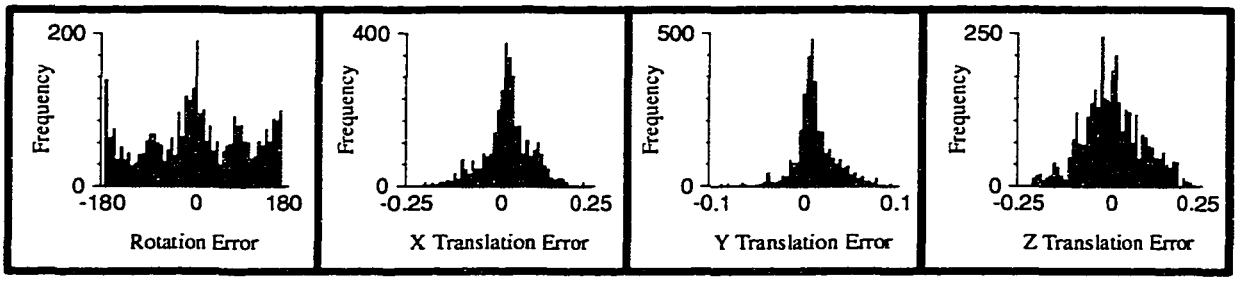

Figure A.10: The signed difference between the ground truth parameters and the pose values of the top ten hypothesis per object per image.

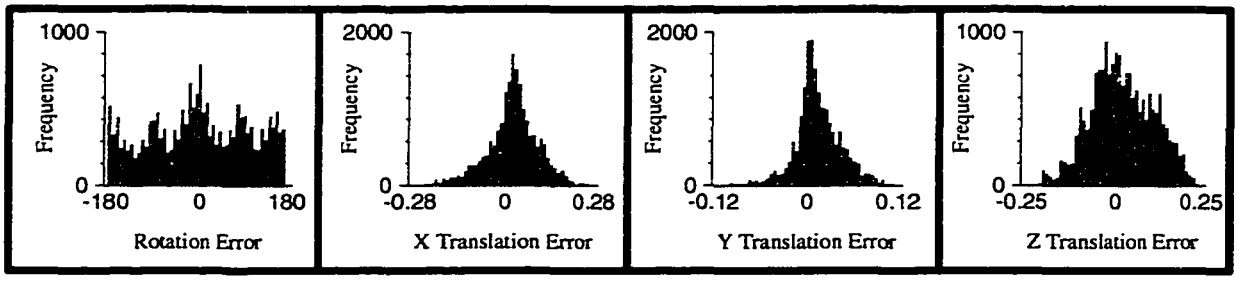

Figure A.11: The signed difference between the ground truth parameters and the pose values of all of the hypothesis per object per image.

Several observations about pose indexing, in terms of these histograms, can be made:

1. The Y translation parameter tends to have the smallest standard deviation (when comparing the distributions of the other parameters, the histogram tails are not as wide). Intuitively, this makes sense it is the most constrained parameter due to the modified GPC (see Section 3.2.1 of Chapter 3).

2. The rotation contains a substantial amount of error. There tends to be numerous peaks in the distribution of rotation values across all of the plots. This implies that rotation is the most difficult parameter to recover.

3. The top ranked hypotheses are typically not any better than the hypotheses in any of the other distributions. This implies that the function used to rank object pose hypotheses is not robust.

These observations lead to an adjustment of the values output by pose indexing. The error in rotation is quite substantial, and will have a large influence over whether the RMR algorithm succeeds or fails: if all of the pose indexing hypothesis are off by 180°, then the iterative refinement of RMR will never locate the correct solution. To compensate, only the translation parameters are used by RMR and a uniform sampling technique (discussed in Chapter 6) is used to generate the rotation parameters.

REFERENCES

A.P., B. Moghaddam, and T. Starner (1994). View-based and modular eigen-spaces for face recognition. *Computer Vision and Pattern Recognition 94*, 84–91.

Bäck, T. and H. Schwefel (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1*, 1 – 23.

Bäck, T. and H. Schwefel (1995). Evolution strategies i: Variants and their computational implementation. In G. Winter, J. Periaux, M. Galan, and P. Cuesta (Eds.), *Genetic Algorithms in Engineering and Computer Science*, pp. 111 – 126. John Wiley.

Balasubrarian, K. (1998). A 3d visualization system for the image understanding environment. Master's thesis, Colorado State University.

Basri, R. and D. Jacobs (1995). Recognition using region correspondences. In *International Conference on Computer Vision*. IEEE.

Besl, P. and R. Jain (1986). Invariant surface characteristics for 3D object recognition from range data. *Computer Vision, Graphics, and Image Processing 33*, 33 – 80.

Beveridge, J. (1993). *Local Search Algorithms for Geometric Object Recognition: Optimal Correspondence and Pose*. Ph. D. thesis, University of Massachusetts at Amherst.

Beveridge, J. (1998). Optimal 2d model matching using a messy genetic algorithm. In *American Association of Artificial Intelligence*, pp. 677 – 683.

Beveridge, J., K. Balasubrarian, and D. Whitley (1999). Matching horizon features using a messy genetic algorithm. To appear in Computer Methods in Applied Mechanics and Engineering.

Beveridge, J., B. Draper, M. Stevens, K. Siejko, and A. Hanson (1997). A coregistration approach to multisensor target recognition with extensions to exploit digital elevation map data. In O. Firschein (Ed.), *Reconnaissance, Surveillance, and Target Acquisition for the Unmanned Ground Vehicle*, pp. 231 – 265. Morgan Kaufmann.

Beveridge, J., J. Griffith, R. Kohler, A. Hanson, and E. Riseman (1989). Segmenting images using localized histograms and region merging. *International Journal of Computer Vision 2*(3), 311 – 347.

Bevington, J. (1992). Laser radar atr algorithms: Phase iii final report. Technical report, Alliant Techsystems.

Bischoff, H. and A. Leonardis (1998). Robust recognition of scaled eigenimages through a hierarchical approach. In *Computer Vision and Pattern Recognition Conference*. IEEE.

Blinn, J. (1996). *A Trip Down the Graphics Pipeline.* Morgan Kaufmann.

Boldt, M., R. Weiss, and E. Riseman (1989). Token-based extraction of straight lines. *Transactions on Systems, Man, and Cybernetics 19*(6), 1581 – 1594.

Boult, T. and G. Wolberg (1992). Correcting chromatic aberrations using image warping. In *Computer Vision and Pattern Recognition*, Volume 92, pp. 684 – 687. IEEE.

Bowyer, K. and P. Phillips (1998). Overview of work in empirical evaluation of computer vision algorithms. In *Empirical Evaluation Techniques in Computer Vision.*

Breiman, L. (1994). Bagging predictors. Technical Report 421, University of California, Berkeley.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). Classification and regression. Technical report, Wadsworth International Group.

Burns, J., A. Hanson, and E. Riseman (1986). Extracting straight lines. *Transactions on Pattern Analysis and Machine Intelligence 8*(4), 425 – 456.

Camps, O. (1993). *PREMIO: The use of Prediction in a CAD-Model-Based Vision System.* Ph. D. thesis, University of Washington.

Canny, J. (1986). A computational approach to edge detection. *Transactions on Pattern Analysis and Machine Intelligence 8*(6), 679 – 698.

Chaudhuri, B. and B. Chanda (1984). The equivalence of best plane fit gradient with roberts', prewitt's, and sobel's gradient for edge detection. *Signal Processing 6*, 143 – 151.

Cohen, P. (1995). *Empirical Methods for AI.* MIT Press.

Cohen, P. and E. Feigenbaum (1982). *The Handbook of Artificial Intelligence*, Volume 3, Chapter XIII Vision, pp. 126 – 321. William Kaufmann, Inc.

Crowley, J. and J. Martin (1995). Experimental comparison of correlation techniques. In *International Conference on Intelligent Autonomous Systems.*

Davies, E. (1987). The effect of noise on edge orientation computations. *Pattern Recognition Letters 6*, 315 – 322.

D.P. Huttenlocher, G. K. and W. Rucklidge (1993). Comparing images using the hausdorff distance. *Transactions on Pattern Analysis and Machine Intelligence 15*(9), 850 – 862.

Draper, B., C. Brodley, and P. Utgoff (1994). Goal-directed classification using linear machine decision trees. *Transactions on Pattern Analysis and Machine Intelligence 16*(9), 888–893.

Duda, R. and P. Hart (1973). *Pattern Recognition and Scene Analysis.* Wiley.

Edwards, J. and H. Murase (1997). Appearance matching of occluded objects using coarse-to-fine adaptive masks. In *Computer Vision and Pattern Recognition.* IEEE.

Flynn, P. and R. Campbell (1998). A www-accessible database for 3d vision research. In *Empirical Evaluation Techniques in Computer Vision.*

Fogel, D. (1993). On the philosophical differences between evolutionary algorithms and genetic algorithms. In D. B. Fogel and W. Atmar (Eds.), *Second Annual Conference on Evolutionary Programming*, pp. 23 – 29. Evolutionary Programming Society.

Foley, J. and A. vanDam (1982). *Fundamentals of Interactive Computer Graphics.* The Systems Programming Series. Addison-Wesley.

Fua, P. and Y. Leclerc (1994). Registration without correspondences. In *Conference on Computer Vision and Pattern Recognition*, pp. 121 - 128. IEEE.

Gaston, P. and T. Lozano-Pérez (1984). Tactile recognition and localization using object models: The case of polyhedra on a plane. *Transactions on Pattern Analysis and Machine Intelligence 6*, 721 - 741.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley.

Grimson, W. (1990a). The combinatorics of object recognition in cluttered environments using constrained search. *AI Journal 44*(1), 121 - 165.

Grimson, W. (1990b). The effect of indexing on the complexity of object recognition. In *Third International Conference on Computer Vision*, pp. 644 - 651. IEEE.

Grimson, W. (1990c). *Object Recognition by Computer: The Role of Geometric Constraints.* MIT Press.

Grimson, W. and D. Huttenlocher (1988). On the sensitivity of the Hough transform for object recognition. In *International Conference on Computer Vision*, pp. 700 - 706.

Grimson, W. and D. Huttenlocher (1991). On the verification of hypothesized matches in model-based recognition. *Transactions on Pattern Analysis and Machine Intelligence 13*(12), 1201 - 1213.

Hadjidemetriou, E. and S. Nayar (1998). Appearance matching with partial data. In *Image Understanding Workshop.* DARPA.

Hanks, S., M. Pollack, and P. Cohen (1994). Benchmarks, test beds, controlled experimentation and the design of agent architectures. *AI Magazine Winter*, 17 - 42.

Haralick, R. and L. Shapiro (1985). Image segmentation techniques. *Transactions on Computer Vision, Graphics, and Image Processing 29*, 100 - 132.

Healey, G. and R. Kondepudy (1992). Ccd camera calibration and noise estimation. In *Computer Vision and Pattern Recognition*, Volume 92, pp. 90 - 95. IEEE.

Heckbert, P. (1989). Fundamentals of texture mapping and image warping. Master's thesis, University of California at Berkeley. CSD-89-516.

Hildreth, E. (1983). The detection of intensity changes by computer and biological vision systems. *Transactions on Computer Vision, Graphics, and Image Processing 22*, 1 - 27.

Hoff, W., K. Nguyen, and T. Lyon (1996). Computer vision-based registration techniques for augmented reality. Technical report, Colorado School of Mines, Division of Engineering.

Hoff, W. and M. Sarojak (1998). Measurement of implant component position and orientation from x-ray images. Technical report, Colorado School of Mines, Division of Engineering.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Hoogs, A. (1996). Pose refinement using a parameter hierarchy. In *Image Understanding Workshop*, pp. 857 - 864. DARPA: Morgan Kaufmann.

Hoogs, A. and R. Bajcsy (1995). Using scene context to model segmentations. In *Workshop on Context-Based Vision*, pp. 50 - 60.

Hoogs, A. and R. Bajcsy (1996). Model-based learning of segmentations. In *International Conference on Pattern Recognition*, Volume 4, pp. 494–499. AIPR: IEEE.

Horn, B. (1986). *Robot Vision.* The MIT Press.

Huang, C., O. Camps, and T. Kanungo (1998). Hierarchical organization of appearance-based parts and relations for object recognition. In *Computer Vision and Pattern Recognition Conference.* IEEE.

Huttenlocher, D. (1998). *Three-Dimensional Recognition of Solid Objects from a Two-Dimensional Image.* Ph. D. thesis, Massachusetts Institute of Technology.

Jones, T. and S. Forrest (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman (Ed.), *Sixth International Conference on Genetic Algorithms*, pp. 184–192.

Kernighan, B. and S. Lin (1972). An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal 49*, 291 - 307.

Kirby, M. and L. Sirovich (1990). Application of the karhunen-loeve procedure for the characterization of human faces. *Transactions on Pattern Analysis and Machine Intelligence 12*(1), 103 - 107.

Kodak, E. (1999). Kodak digital camera dc25. http://www.kodak.com/US/en/digital/cameras/dc25/.

Koenderink, J. (1984). What does occluding contour tell us about solid shape? *Perception 13*, 321 - 330.

Koenderink, J. and A. vanDoorn (1979). The internal representation of shape with respect to vision. *Biological Cybernetics 32*, 211 - 216.

Kumar, R. (1992). *Model Dependent Inference of 3D Information From a Sequence of 2D Images.* Ph. D. thesis, University of Massachusetts at Amherst.

Lamdan, Y. and H. J. Wolfson (1988). Geometric hashing: A general and efficient model-based recognition scheme. In *Second International Conference on Computer Vision*, pp. 238 - 249. IEEE.

Lengyel, J. (1998). The convergence of graphics and vision. *Computer 31*(7), 1 - 11.

Li, X., M. Ferdousi, M. Chen, and T. Nguyen (1986). Image matching with multiple templates. In *Computer Vision and Pattern Recognition*, Volume 86, pp. 610 - 613. IEEE.

Lichten, S. (1988). High accuracy global positioning system orbit determination: Progress and prospects. In Y. Bock and N. Leppard (Eds.), *Global Positioning System: An Overview.* Springer-Verlag.

Lin, S. and B. Kernighan (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*, 498 - 516.

Linneanmaa, S., D. Harwood, and L. Davis (1988). Pose determination of a three-dimensional object using triangle pairs. *Transactions on Pattern Analysis and Machine Intelligence 10*(5), 634–647.

Lowe, D. (1985). *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers.

Lowe, D. (1987a). Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence 31*, 355 - 395.

Lowe, D. (1987b). The viewpoint consistency constraint. *International Journal of Computer Vision 1*(1), 58 - 72.

Mantyla, M. (1990). *An Introduction to Solid Modeling.* Computer Science Press.

Marr, D. and E. Hildreth (1980). Theory of edge detection. *Royal Society of London B207*, 187 - 217.

Mata, J., R. Marik, and J. Kittler (1995). On representation and matching of multi-colored objects. In *International Conference on Computer Vision*, pp. 726–732. IEEE.

Nalwa, V. (1993). *A Guided Tour of Computer Vision.* Addison-Wesley Publishing.

Nayar, S., S. Nene, and H. Murase (1996). Real-time 100 object recognition system. In *Image Understanding Workshop.* DARPA: Morgan Kaufmann.

Nelder, J. and R. Mead (1965). A simplex method for function minimization. *Computer Journal 7*, 308 - 313.

Nene, S., S. Nayar, and H. Murase (1996). Columbia object image library (coil-100). Technical report, Columbia University. CUCS-006-96.

Nevatia, R. and R. Babu (1980). Linear feature extraction and description. *Transactions on Computer Vision, Graphics and Image Processing 13*, 257 - 269.

Ohba, K. and K. Ikeuchi (1996). Recognition of the multi-specularity objects using the eigen-window. Technical Report CMU-CS-96-105, School of Computer Science, Carnegie Mellon University.

Paul, B. (1998). The mesa 3-d graphics library. http://www.mesa3d.org.

Peli, T. and D. Malah (1982). A study of edge detection algorithms. *Transactions on Computer Vision, Graphics, and Image Processing 20*, 1 - 21.

Plantinga, W. (1988). *The ASP: A Continuous, Viewer-Centered Object Representation for Computer Vision.* Ph. D. thesis, University of Wisconsin at Madison.

Pope, A. (1995). *Learning to Recognize Objects in Images: Acquiring and Using Probabilistic Models of Appearance.* Ph. D. thesis, University of British Columbia.

Press, W., B. Flannery, S. Teukolsky, and W. Vetterling (1988). *Numerical Recipes in C.* Cambridge University Press.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning 1*, 81 - 106.

Rana, S. (1999). *Examining the Role of Local Optima in Genetic Search.* Ph. D. thesis, Colorado State University.

Roberts, L. (1965). Machine perception of three–dimensional solids. In J. T. Tippett (Ed.), *Optical and Electro-Optical Information Processing*, Chapter 9, pp. 159 - 197. MIT Press.

Ross, C. and J. Beveridge (1998). First break velocity picking using local search. Colorado Advanced Software Institute report.

Sarojak, M. (1998). Interactive pose estimation of total joint arthroplasty via x-ray fluoroscopy. Master's thesis, Colorado School of Mines.

Seales, W. and C. Dyer (1992). Modeling the rim appearance. In *Third International Conference on Computer Vision*, pp. 698 - 701.

Shustorovich, A. (1994). Scale specific and robust edge/line encoding with linear combinations of gabor wavelets. *Pattern Recognition 27*(5), 713 - 725.

Sokal, R. and F. Rohlf (1981). *Biometry: The Principles and Practice of Statistics in Biological Research.* W.H. Freeman and Company.

Spiegel, M. (1996). *Probability and Statistics.* McGraw Hill.

Stevens, M. (1995). Obtaining 3d silhouettes and sampled surfaces from solid models for use in computer vision. Master's thesis, Colorado State University.

Stevens, M. and J. Beveridge (1997). Precise matching of 3-d target models to multisensor data. *Transactions on Image Processing 6*(1), 126 - 142.

Stevens, M. and J. Beveridge (1998). Localized scene interpretation from 3d models, range, and optical data. Submitted to Computer Vision and Image Processing Journal.

Sullivan, G. (1992). Visual interpretation of known objects in constrained scenes. In *Royal Society Discussion Meeting On Known Natural and Artificial Low Level Seeing Systems.*

Swain, M. (1990). *Color Indexing.* Ph. D. thesis, University of Rochester.

Swets, D., B. Punch, and J. Weng (1995). Genetic algorithms for object recognition in a complex scene. In *International Conference on Image Processing*, pp. 595 - 598.

Umbaugh, S. (1997). *Computer Vision and Image Processing.* Prentice Hall.

Waltz, D. (1972). Generating semantic descriptions from drawings of scenes with shadows. In *The Psychology of Computer Vision*, pp. 19 - 92. McGraw Hill.

Wells, W. (1993). *Statistical Object Recognition.* Ph. D. thesis, Massachusetts Institute of Technology.

Wells, W., M. Halle, R. Kikinis, and P. Viola (1996). Alignment and tracking using graphics hardware. In *Image Understanding Workshop*, pp. 837 - 842. DARPA.

Wheeler, M. and K. Ikeuchi (1995). Sensor modeling, probabilistic hypothesis generation, and robust localization for object recognition. *Transactions on Pattern Analysis and Machine Intelligence 17*(3), 252–265.

Whitley, D., J. Beveridge, C. Graves, and K. Mathias (1996). Test driving three 1995 genetic algorithms: New test functions and geometric matching. *Journal of Heuristics 1*, 77 - 104.

Whitley, D., J. R. Beveridge, C. Guerra-Salcedo, and C. Graves (1997). Messy genetic algorithms for subset feature selection. In *International Conference on Genetic Algorithms*, pp. 568 - 575.

Witkin, A. (1983). Scale space filtering. In *International Joint Conference on Artificial Intelligence*, pp. 1019 - 1022.