Real-Time Large Object Category Recognition Using Robust RGB-D Segmentation Features

by

Ross Edward Finman

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feburary 2013

Author ....[illegible].......
Department of
Electrical Engineering and Computer Science
October 5, 2012

Certified by ....
Seth Teller
Professor of Electrical Engineer and Computer Science
Thesis Supervisor

Accepted by ....[illegible].......
Leslie Kolodziejski
Chairman, Department Committee on Graduate Thesis

Real-Time Large Object Category Recognition Using Robust RGB-D Segmentation Features

by

Ross Edward Finman

Submitted to the Department of
Electrical Engineering and Computer Science
on October 5, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis looks at the problem of large object category recognition for use in robotic systems. While many algorithms exist for object recognition, category recognition remains a challenge within robotics, particularly with the robustness and real-time constraints within robotics. Our system addresses category recognition by treating it as a segmentation problem and using the resulting segments to learn and detect large objects based on their 3D characteristics. The first part of this thesis examines how to efficiently do unsupervised segmentation of an RGB-D image in a way that is consistent across wide viewpoint and scale variance, and creating features from the resulting segments. The second part of this thesis explores how to do robust data association to keep temporally consistent segments between frames. Our higher-level module filters and matches relevant segments to a learned database of categories and outputs a pixel-accurate, labeled object mask. Our system has a run time that is nearly linear with the number of RGB-D samples and we evaluate it in a real-time robotic application.

Thesis Supervisor: Seth Teller
Title: Professor of Electrical Engineer and Computer Science

Acknowledgments

This work required the support of many people. First, to my advisor Prof. Seth Teller, thank you for your invaluable guidance, advice, and, most importantly, patience during these past two years. To Sachithra Hemachandra, thank you for your thoughtful input into my work and invaluable friendship. To Matthew Walter, thank you for your research insights and advice on my work and bigger picture.

To all my friends, thank you for being there for the long research discussions, the in-depth debates, and making sure my life was never boring. To my family, thank you for your unwaivering moral support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic systems are pervasive in labs, factories, and other well-structured environments, but are limited by their ability to interpret their surroundings. For robots to function within new and dynamic settings, they must be able to effectively sense their environment. Robots must be able to distinguish objects around them so that they can interact with, avoid, or learn from objects. Humans effectively understand their environments using only sight in many cases; the human eye gathers light through a lens and the human brain combines these optic nerve firings to sense its surroundings with stereo vision. While cameras also gather light through a lens, the abstraction from individual camera pixels to meaningful object representations on a computer is vastly inferior to the brain. How can we use pixels to detect categories of objects so they may be utilized by a robot?

The problem is that by themselves, pixels carry relatively little information about the world. A blue pixel from the ocean or the sky is impossible to distinguish between if the RGB values are the same. A single depth point is just as ambiguous. As such, it is intuitive to look not just at individual pixels, but to look at many of them at once - ideally all the pixels that make up the object of interest. However, identifying those pixels is a difficult problem. Take a chair for example. Chairs come in many different colors, shapes, and sizes so trying to match a pixel based model is difficult (as seen in the Pascal VOC Challenge [2], [3]), and trying to do an exhaustive search across all combinations of RGB-D pixels is intractable. Higher level representations are needed to decrease the search space. One simple approach is to use a bounding box (a rectangular box of pixels) to give a region of an image that an object may be in. However, not all objects can be represented by a box; even varying the size of the bounding box won't necessarily lead to correct results. Our chair, for example, does not fit well in a rectangular box for many viewpoints. Also, for our end goal of utilizing category detection on a robot, a bounding box is less useful than the pixel-accurate object. Rough knowledge of where a doorknob is may not be useful when a robot needs to grasp the doorknob and not merely move its arm into the vicinity of the knob. This motivates our goal of recognizing categories of objects and accurately determining their pixel masks.

1.1 Problem Description and Approach

The research task undertaken in this project aims to address some of the technological challenges in object category recognition such as scale invariance, computation time, and consistency of features (each described below). An object recognition system is scale invariant if it can recognize an object from many scales. If an object is 1m away or 3m away, the object recognition system should be invariant to the distance. A recognition system running on a robot should be computational fast, i.e. run in real-time for the robot to be timely in its actions relating to objects. Lastly, feature consistency is having features remain persistent and repeatable between frames with minimal movement and noise. The end result of this research is a **real-time large**[1] **object category recognition system** that produces a pixel-accurate mask of objects based off of RGB-D segment features. We validate our method by measuring error rates on real-world datasets and populating a map with objects seen on a tour of a building containing several categories of objects previously learned.

Our solution addresses the constraints autonomous robots put on computer vision algorithms so that robots can perceive their environments. Robotic systems rarely

[1]Large is defined as either a collection of multiple feature vectors, or a single feature vector with features distinguishable from noise.

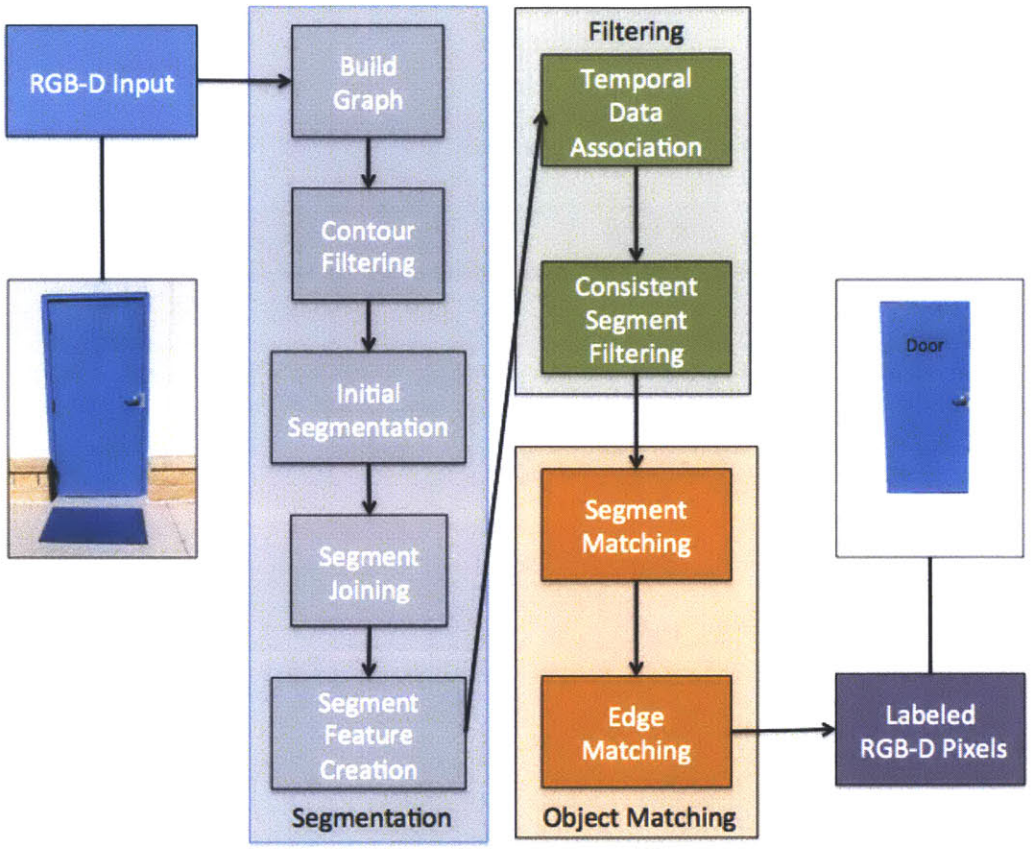

Figure 1-1: Outline of algorithm.

run object category recognition algorithms online due to their computational time and error rates. Exceptions being tabletop object recognition since it is a constrained environment, and large outdoor vehicles with abundant computation power. We demonstrate progress toward real-time object category recognition for robots using the algorithm in Figure 1-1.

On a high level, our method in Figure 1-1 takes a live RGB-D video stream and a set of previously learned object templates as input. For every frame, the algorithm builds a pixel-based graph, then filters out large depth discontinuities. The algorithm then over-segments the RGB-D image, and selectively joins segments together based on their similarity. Then the algorithm creates a feature vector from the segment properties. Next, the segment features are associated between frames and unstable segments are filtered out. Finally, the algorithm matches groups of segments against object templates. The result is a pixel-accurate labeled object mask.

1.2 Thesis Overview

The organization of this thesis is as follows.:

Chapter 2 gives the background of the current problem domain and current ways others have approached the issue. It also describes relevant research that has influenced our work.

Chapter 3 covers the RGB-D camera options currently available, along with the relevant software packages and robotics systems utilized.

Chapter 4 describes how the system segments the RGB-D frames robustly through an initial contour filtering, then segmentation, and finally creation of segment feature descriptors.

Chapter 5 describes data association between frames, longer-term matching of objects, and learning of new objects.

Chapter 6 shows results from a series of trials of image sequences.

Chapter 7 summarizes our approach and discusses possible future work.

Chapter 2

Related Work

Object category recognition is a longstanding and ongoing field in computer vision [4]. As discussed in Chapter 1, classifying an object within an image based on individual pixels is intractable. As a result, many current object recognition algorithms represent objects with an object model, segmentation, or collection of features[1] to match against an object database [4]. In this chapter we will first describe different feature representations used in object recognition. Then we will discuss popular object recognition algorithms. Finally, we will discuss segmentation algorithms relevant to our feature representation.

2.1 Feature Descriptors

Feature detectors have been used successfully for object recognition [5] - in computer vision, features are based on the RGB values of pixels. Generally, features can be sub-divided into two classes: local, and global [4]. Global features, such as GIST [6], describe an entire image and, thus, do not offer the desirable pixel-level granularity so they will not be looked at in this thesis.

A local feature is a representation of a simple, but ideally distinctive property of a part of an image. While a local feature can be as simple as the average color of a predetermined region, such simple representations are not invariant to subtle changes,

[1]A feature is part or all of a set of data

such as scaling or illumination. Lowe [7] developed the more advanced Scale Invariant Feature Transform (SIFT) descriptor, which is mostly invariant to uniform scaling, orientation, and partially invariant to affine distortion and illumination changes. Bay et al. [8] developed the Speeded Up Robust Feature (SURF) descriptor; a faster, though less accurate, version of the SIFT feature. Features are not specific to RGB space, but to other datasets as well, such as 3D point clouds. Johnson [9] created spin images to describe a point on a surface in 3D by looking at the surface normal and the radially surrounding 3D points.

Local features can have larger neighborhoods that provide more information about regions of an image. A common example is a sliding window, which Viola & Jones [10] used in their real-time object detector. Dalal et al. [11] developed Histogram of Oriented Gradients (HOG) descriptors which, when used for object category detection, rank high in the Pascal challenge [3]. Forssén et al. [12], [13] created features by modeling similarly colored regions in an image as ellipses. These methods, however, are limited in that they provide only regions and not pixel-level granularity.

2.2 Object Recognition

Many object recognition methods are built upon local features [4]. Many of the features described in Section 2.1 were validated, at least in part, on object recognition datasets [7], [8], [10], [11], [12]. Lowe [14] used a SIFT matching framework to do robust object instance recognition. Walter et al.[15] also used SIFT features to build multiple feature-based descriptions for varying viewpoint object instance recognition. Lai et al. [16], using an RGB-D camera [17], combined RGB and depth features in a recognition framework to increase performance over just RGB feature methods. Felzenszwalb et al. [18] useed HOG features in their deformable part models to match parts of an object, thus being more robust to occlusions and variance in the object model as evidenced in the Pascal Challenge [3]. Since these methods are based off of underlying region features, they too only provide regions and not pixel-level granularity of the objects.

2.3 Segmentation

Segmentation is the process of partitioning a set of data into a condensed representation that is more *meaningful* and easier to analyze than the raw data - in computer vision, segmentation groups individual pixels that *go together*. Deciding what is *meaningful* or what pixels *go together* is task dependent and difficult to define, therefore generic segmentation is an under constrained problem [19]. In this section we look at unsupervised segmentation methods (no training or human in the loop) that run in near real-time for segmenting RGB images, 3D point clouds, and RGB videos.

2.3.1 RGB

Segmentation algorithms for RGB images has been a research area within computer vision for over four decades [20], and remains an active research area today. There are a multitude of algorithms for the segmentation problem [19], [21]. We will discuss two of the most widely used. Comaniciu et al. [22] introduced the mean-shift segmentation technique that first does a mean shift filtering of the original data and the clusters the filtered data points. Felzenszwalb et al. [23] developed an efficient graph-based segmentation algorithm based on grouping pixels based on dynamic thresholds. We use the latter in our feature implementation detailed in Chapter 4.

2.3.2 Point Cloud

In addition to RGB images, segmentation is used in 3D point clouds as well. Holz et al. [24] developed a real-time plane segmentation method based on using a point's surface normal and Felzenszwalb's segmentation algorithm [23]. Holz and Behnke [25] furthered Holz's point cloud segmentation method [24] by taking the plane segmentation and running RANSAC [26] to find geometric primitives (planes, spheres, or cylinders) in real-time for down-sampled images. Both segmentation methods specifically look for simplestic shapes, but do not account for other shapes besides planes [24] or geometric primitives [25]. Shotton et al. [27] introduced a robust, real-time human body part segmentation method on dense 3D point clouds using a decision tree that, while not general enough to use for generic objects, is effective. Strom et al. [28] introduced a combined RGB and 3D segmentation approach that uses a local surface normal, depth, and color features to segment an RGB-D image, but due to the individual threshold for each feature, the algorithm is sensitive to the chosen parameter values. Lai et al. [29] developed a detection-based object labeling algorithm using sliding window detectors to determine the probability of a pixel belonging to a previously learned object class, and then using the pixel probabilities to label previously learned objects in a point cloud.

2.3.3 Video

Video segmentation methods differ from image segmentation by using segmentations from multiple frames to create a temporally consistent segmentation result. Grundmann et al. [30] built upon the single image segmentation work done in [23] by creating multiple segmentations from different parameters, and then, using a hierarchical method, combining the segmentations temporally. While the method gives robust results, it takes on order of twenty minutes to process forty seconds of 25 frames/second video. Abramov et al. [31] combined depth information with RGB to create a more robust and faster segmentation than just RGB alone. However, this work requires full RGB-D videos and, as such, does not take in each frame as it is received.

Chapter 3

Infrastructure

The work in this thesis would not be possible without the recent advances in RGB-D cameras [17]. These cameras can and have been incorporated into robotic platforms in order to greatly increase their capabilities. The cameras can be used by robots to recognize their surroundings, but due to the generality of cameras, the systems can be applied to many situations and/or datasets. In this chapter we will describe RGB-D cameras, the platform on which our system was tested, and the packages utilized in this work.

3.1 RGB-D Cameras

Historically, stereo cameras have been the primary source of RGB-D images, however active lighting RGB-D cameras have become commonplace in robotics due to their relative inexpensiveness and ease of use. While the most popular RGB-D camera in use is the Microsoft Kinect [17], there are other options available with different pros and cons for this application and others as discussed below.

3.1.1 Primesense

The most popular RGB-D camera is the Microsoft Kinect (see Figure 3-1) [17] developed by Primesense which brought to market an affordable depth sensor. The camera

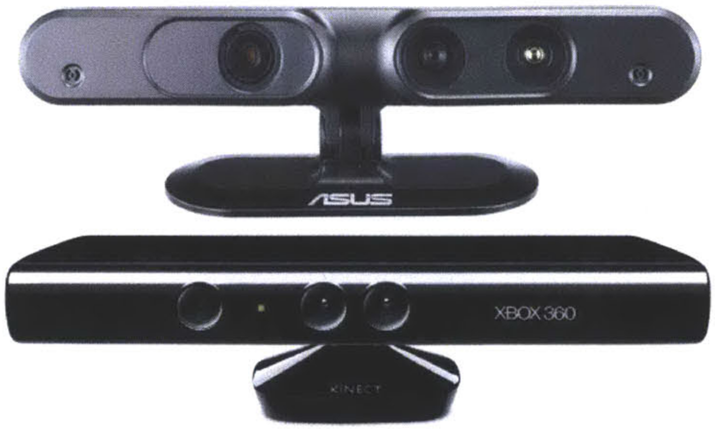

Figure 3-1: The ASUS Xtion (top), and Microsoft Kinect (bottom).

uses an RGB camera, along with an infrared (IR) projector and IR camera that together determine depth. The depth system works by projecting a pseudo random set of IR laser points into a scene, which the IR camera sees. Then using image based 3D reconstruction methods, such as Zhang et. al. [32], the system can estimate the depth accurately between 0.5m and 4m away from the camera. The benefit of active lighting cameras is that they offer a dense point cloud that works even on featureless surfaces such as walls. Two major drawbacks are, first, the IR points are easily washed out due to IR interference such as sunlight and, second, the rolling shutter on these cameras causes scene warping when the camera is in motion [33].
Primesense makes several types of cameras that have distinguishing features pertinent for robotic applications. The Microsoft Kinect sensor, while the most widely used, has two other limitations when compared to other active lighting RGB-D cameras such as the Primsesense ASUS Xtion. First, the Kinect RGB and depth cameras are not synchronized. This causes errors when matching pixels from the IR to RGB camera coordinate frames when the sensor is in motion, thus limiting the maximum velocity at which reasonable data can be expected. The second limitation is that the Kinect requires more power than a USB port can provide and requires an external battery for mobile movement. The ASUS does not have these two problems, thus providing better data and easier use. The work in this paper is based on the ASUS

and Kinect RGB-D cameras.

3.1.2 SoftKinetic DepthSense

While Primesense cameras are the primary structured-light cameras on the market, there are other technologies such as Time-Of-Flight (TOF) cameras, for example, the SoftKinetic DepthSense [34]. This technology is promising in that it can work in direct sunlight with similar frame rates to structured-light cameras. The current downside of these cameras is that the resolution is 1/4th that of the Primesense cameras discussed in Section 3.1.1. In testing, this camera was able to get depth data in sunlight, though only on some materials (plastic and unreflective metal), and not others (concrete, bark, grass). The camera also has the same limitations as the Kinect in that the color and depth cameras are not synchronized, and it needs external power.

3.1.3 Stereo

As stated above, stereo cameras have historically been the primary means of collecting RGB-D data. Stereo camera systems, such as the PointGrey Bumblebee, while offering dense colored point clouds, have some limitations when compared to structured lighting cameras. First, stereo pairs are generally quite expensive, and while single cameras can be used together with additional calibration, the individualistic design makes generalizing more difficult. Second, stereo cameras work only in scenes that contain some texture; providing incorrect results when looking at large textureless objects. Lastly, due in part to the previous points, there is not as large of a community supporting stereo systems as there are with structured light cameras. One primary draw of stereo cameras is that they work outside, which is what future work will build upon.

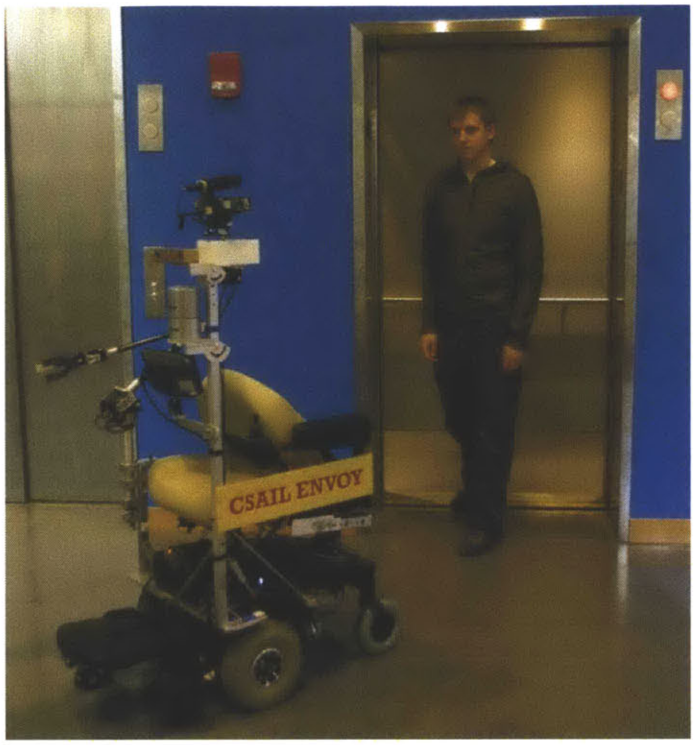

Figure 3-2: Envoy robot guiding a person.

3.2 Platform

The work for this thesis was developed for use on the CSAIL Envoy platform, a converted robotic wheelchair [35]. The Envoy robot has numerous sensors for robotic navigation and human interaction. As seen in Figure 3-2, a Kinect is mounted roughly at human height on a pan-tilt mount near the top. This mount allows the camera to view around the robot, thus offering more dynamic and varying viewpoint datasets (used in the results later).

3.3 OpenNI

The most widely used software used for interfacing with the Kinect and ASUS Xtion is OpenNI (Open Natural Interaction), an open source standard interface for 3D sensor data. The stated goal of OpenNI is to provide an interface for applications that use natural interface (humans gestures or poses) as their input [36]. Because OpenNI guarantees backwards compatibility, any application can use the interface, and then run regardless of the version of the natural interface input.

3.4 PCL

With the advent of cheap and robust depth sensors like the Microsoft Kinect, 3D perception has become a fast growing area. The Point Cloud Library (PCL) [37] is an advanced and extensive open source library with users and developers from around the world. The library utilizes OpenNI, and contains advanced algorithms for filtering, feature estimation, surface reconstruction, and additional applications. In this work, we use the PCL point cloud framework as the base of our system.

Chapter 4

Segmentation

In order to have a pixel-accurate object segmenter, each pixel must be labeled as belonging to an object or segment of an object. Segmentation is a well-researched area, but remains a notoriously under constrained problem. A human segmenting a scene uses higher-level semantics that traditional segmentation methods do not capture. As discussed in Chapter 1, going from raw pixels to semantic objects is a challenging problem. We define the following desirable features of a segmentation algorithm that further constrain the problem for use in future modules.

- **Robust**

A segmentation is considered to be robust if it is invariant to rotation, translation, as well as changes in scale and illumination. If a scene is viewed from a vastly different viewpoint in either rotation, translation, scale, or any combination, but still has most of the original scene unobscured, then the segmentation output should be similar.

- **Consistent**

A segmentation is considered to be consistent if there is minimal or no camera movement, then the segments should remain persistent between frames. The RGB-D frame should be segmented the same despite camera noise, or any other minor change in the scene.

While the above definitions are similar, the important distinction is that robustness provides a constraint for how the features should be defined (rotation, translation, and scale invariant), and consistency provides a constraint for what is a correct segmentation (one that is persistently segmented the same).

One limitation to the above segmentation constraints is if the camera does not have the actual data to distinguish the information. If, due to aliasing, motion blur, or any other camera-based shortcoming, there is no data to influence the segmentation method; then a segmentation algorithm cannot reasonably be expected to correctly segment an image. We use higher-level modules to handle such problems, and other issues that will be addressed in the next chapter.
The above definitions are useful in that they refer only to the sensor and the resulting image and not to some higher-level semantic information. This avoids many of the difficulties that go along with the semantic segmentation problem in the process. Now we describe a segmentation algorithm that efficiently segments a scene while being qualitatively robust. The outline of this algorithm is in Figure 4-1. Consistency is handled by a higher-level module and is described in the next chapter.

4.1 Graph Building

As shown in Figure 4-2, the graph building module takes in an RGB-D frame and outputs a graph with each pixel as a node and each edge is the Hue Saturation Value (HSV) distance [38] between a pixel and a neighboring pixel. The pixels are connected in a standard four-connected manner (Figure 4-3). Each pixel has eight directly neighboring pixels in a standard camera, though since the edges in our graph are undirected, a full eight-connected graph is redundant.

4.2 Contour Filtering

The critical problem in image segmentation is determining which pixels are part of the same segment. To help simplify the problem, we assume an object is smooth in depth for at least one part of the object. As shown in Figure 4-4, if there is a discontinuity of more than 5 cm in the graph from Section 4.1, we remove the edge between the two pixels/nodes in the graph. The 5 cm was determined as just larger than the specified

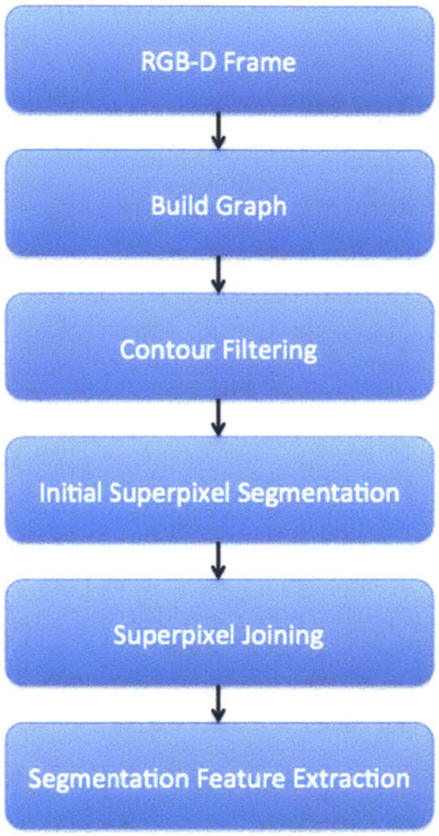
RGB-D Frame
Build Graph
Contour Filtering
Initial Superpixel Segmentation
Superpixel Joining
Segmentation Feature Extraction

Figure 4-1: Segmentation Outline.

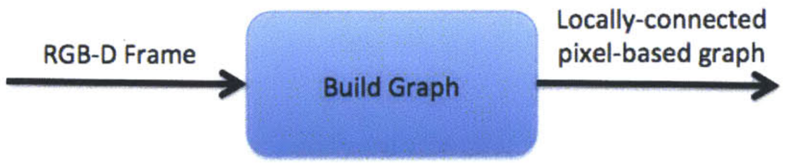

Figure 4-2: Input and output diagram for the build graph module.

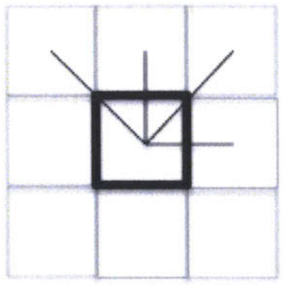

Figure 4-3: Pixel layout showing a the eight neighboring pixels to the center pixel, with the four-connected edges shown as black lines.

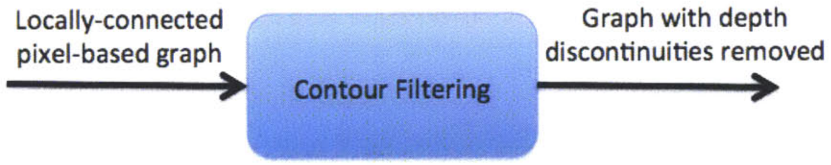
Locally-connected pixel-based graph
Contour Filtering
Graph with depth discontinuities removed

Figure 4-4: Input and output diagram for the contour filtering module.

depth camera noise of 1-2 cm at 4 m (so 2-4 cm between two surfaces).

4.3 Initial Segmentation

We use the Felzenszwalb segmentation algorithm [23] to take the resulting graph from the contour filtering method to create superpixels (Figure 4-5). The benefit of this algorithm is that it runs in O(N log(N)) time with N pixels. The algorithm provides a principled guarantee that the segmentation is neither too fine nor too coarse (as defined within the paper) based on two parameters, T and k, where T is the initialized threshold of every node, and k is the growth rate threshold (described in detail below). The segmentation algorithm uses the graphical model described above, with contour edges having already filtered out, and the edges having the HSV distance value between the two nodes/pixels in the graph. Using the simple raw RGB sum of absolute differences is not as robust as other popular methods such as HSV distance [38], and as such, the latter was used in this work. Depth is ignored in this step as segmenting based on depth and color simultaneously is an ambiguous

Graph with depth discontinuities removed
Initial Superpixel Segmentation
Labeled Superpixels

Figure 4-5: Input and output diagram for the Initial Segmentation module. The labels of the superpixels are arbitrary, but unique identifying integers.

comparison.

Once the graph is set up, the Felzenszwalb algorithm sorts the edges by the magnitude of the edge HSV distance value. A counting sort method was used for computation speed to bring the O(N log(N)) down to O(αN) for a constant α. Each node i is initialized to threshold parameter T, and the edges are then considered from lowest to highest HSV distances, with their corresponding differences checked against the node threshold T_i. If the HSV distance between two nodes i and j is less than T_i or T_j, then the nodes are joined together in a UnionFind data structure. The new root node's threshold, say T_i, is then updated and used as the threshold for all nodes unioned together.

$$T_i = T_i + k/N_i \tag{4.1}$$

Where k is the growth rate parameter described above and N_i is the number of nodes within segment i (initially 1). Updating the threshold increases the value of the threshold less and less, so with an increasing edge magnitude, the segment is less likely to join another node. The algorithm is shown in Algorithm 1.

4.4 Superpixel Joining

The Felzenszwalb algorithm outputs segments made up of pixels with minimal HSV distance between them, given the parameters k and T. However, the parameters of the algorithm bias the super pixels to be similarly sized, thus making edges where the

Algorithm 1 Initial Segmentation algorithm

```
Initialize UnionFind nodes for nodes in filtered graph G
Initialize all node thresholds $T_i$ to $T$ $\forall$ $i$
Sort edges E $\in$ filtered graph G

for Every $e_k \in E$ do
   /**Find root nodes of the edge nodes*/
   i = UnionFind.find($e_k$.node1)
   j = UnionFind.find($e_k$.node2)

   /** If the HSV Distance between nodes $i$ and $j$ of edge $e_k$ is less than the dynamic
   thresholds $T_i$ and $T_j$ then join*/
   if $e_k$.dist < $T_i$ && $e_k$.dist < $T_j$ then
      UnionFind.join(i,j)
      /**Update the root node's threshold*/
      i = UnionFind.find(i)
      $T_i = T_i$ + k / UnionFind.size(i)
   end if
end for
```

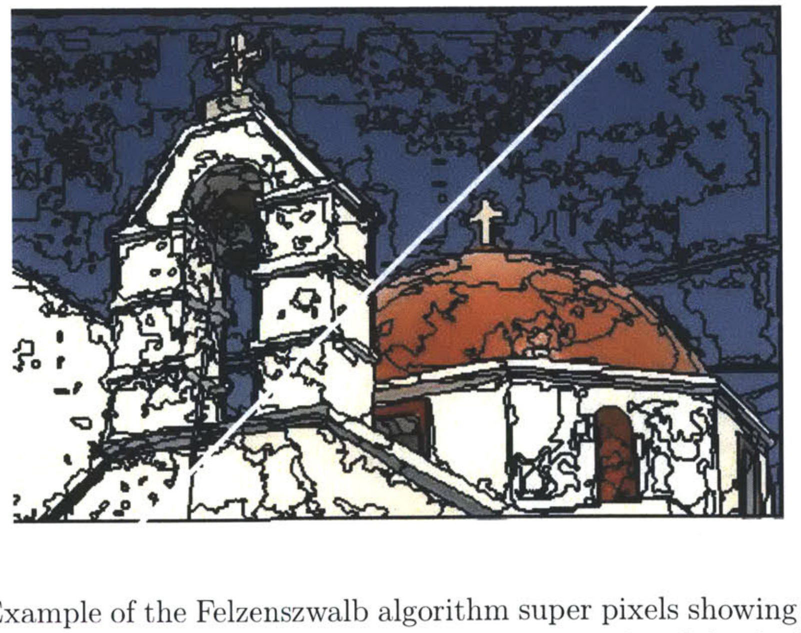

Figure 4-6: Example of the Felzenszwalb algorithm super pixels showing two different parameter settings. The top left favors smaller super pixels while the bottom right favors larger super pixels. Source: Radhakrishna Achanta et al. [1].

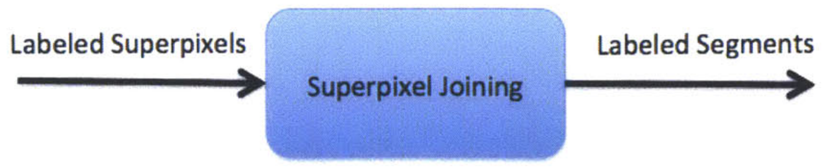

Figure 4-7: Input and output diagram for the Superpixel Joining module. The labels of the segments are arbitrary, but unique identifying integers.

image is locally smooth in color (as seen in Figure 4-6). This property is useful for superpixels if the parameters are set to favor small segments, but limits the algorithm from being the final segmentation for our problem. Since we want the result to be robust and consistent, as defined above, a table or any object broken up in somewhat arbitrary segments won't work (as seen in Figure 4-6), nor will ignoring small objects (in terms of the number of pixels), as this would not be scale invariant. We want the super pixels to be joined to create our final segments (Figure 4-7).

Once the image is segmented into superpixels, the joining algorithm takes in the set of edges where the edge magnitude was below T_i and T_j when the edge was considered in the segmentation loop. This set is the edges between separate segments minus the edges removed in the contour filtering method. Again using the contour filtering described above, if the edge magnitude is below a global joining threshold A, then the superpixels are joined together. The intuition is that superpixel algorithms find some boundaries of objects, but do not find others, determined qualitatively. Joining along the edges combines superpixels that have similar HSV values along the border (suggesting the Felzenszwalb algorithm stopped joining nodes due to the growth rate parameter), but ignores large HSV difference values (suggesting the algorithm stopped due to the threshold parameter).

The last part of the joining method is to handle sensor noise. Due to lack of depth returns and pixel aliasing, there are some stray pixels that are not joined into any segment even after the joining method above. To handle these minimal errors, every segment that has less than D nodes is joined with an arbitrary neighboring segment. It is important to note that these errors are minimal and that this step is

Labeled Segments
Segment Feature Extraction
Set of Segment Feature Vectors

Figure 4-8: Input and output diagram for the Segment Feature Extraction module.

not required for this work to function properly. The process only cleans up images for human viewing and is used in all images in this thesis. In practice, D is set to 30 pixels, or 0.01% of the whole image.

4.5 Segment Feature Extraction

Once the segmentation is complete, a segment feature vector (SFV) is created for each segment and will be utilized by methods discussed in the next chapter. In order for the SFV to be both scale and rotational invariant, the features selected are based on coordinate independent properties of the XYZ points. The features are defined as follows with descriptions of how each feature is computed. Principle Component Analysis (PCA) [39], [40] on the XYZ point cloud is used for several of the features and is described below.

Feature	**Description**
Length	The max distance of points projected along the first normalized eigenvector of the segment from PCA. Each point is turned into a vector starting from the centroid and projected onto the first eigenvector (the major axis of the segment). The difference of the maximum and minimum values is the length.
Width	The width along the second principle component vector, computed in the same way as the length above, but projected onto the second normalized eigenvector.
Standard deviation along length	The standard deviation of all the points along the first eigenvector from PCA. The projection of each point on the first eigenvector is recorded and the total standard deviation is computed. This feature is used because the length and width do not provide sufficient information to uniquely determine a segment. For example, a plus sign and a box with the same length and width have different standard deviations.
Standard deviation along width	The standard deviation of all the points along the second eigenvector, computed similarly to the previous feature.
Curvature along length and width	The curvature along the first and second principle axises. This is computed by taking the first and second eigenvalues (eigenvalues of the length and width), and dividing the first by the second. The value is strictly greater than or equal to 1 since the eigenvalues are ordered.
Area	The pseudo area of the segment, computed as the number of pixels multiplied by the squared distance from the camera. This does not give the true area, but is scale invariant as the number of pixels decreases by the squared distance away from the camera. This feature is susceptible to foreshortening, but this is accounted for in the next chapter.
Average RGB value	The average RGB value of the entire segment.

Feature cont.	Description cont.
Surface normal:	The third eigenvector from PCA, which gives the approximate surface normal of the whole segment. This vector is defined in the camera coordinate frame. The feature is only used with other segments within the same frame so the local coordinate frame is irrelevant as long as it is consistent between all segments within a frame.
Centroid	The XYZ centroid of the object in the camera coordinate frame. As with the previous feature, this camera centric coordinate frame is used only with other segments within the same frame.

Table 4.1: Table of features in a Segment Feature Vector (SFV)

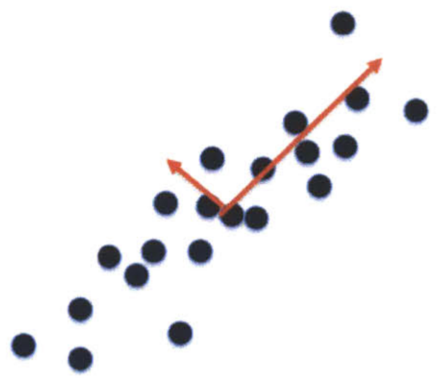

Figure 4-9: Example of PCA. The black dots are data points and the red axes show the major and minor axis representing the data.

Each one of these features is scale and rotationally independent (except the surface normal and centroid, used later for object matching), thus fulfilling the robustness requirement as described at the beginning of this chapter.

4.5.1 PCA

The system uses an XYZ point cloud to determine the segment features. It is useful to transform the XYZ space into a coordinate system specific to the segment based on 3D points. In the general case, PCA takes a set of d-dimensional vectors and assigns a weight to each dimension based on the variance of the component [40]. We use PCA in the system to determine a coordinate frame based on the variance of the input data. Relating that to 3D, the first eigenvector from PCA corresponds to the vector with the most variance (the major axis) in XYZ space, the second eigenvector corresponds to the vector with the second most variance orthogonal to the first, etc... In this work, this is the axis along the length, width, and surface normal of a segment. A 2D example is shown in Figure 4-9. The corresponding eigenvalues of the eigenvectors measure the variance along each axis and are used as a relative measure of curvature. For example, a perfect circular disk will have arbitrary first and second eigenvectors in XYZ space since the variance is equal, and their respective eigenvalues would be approximately equal.

Chapter 5

Matching Objects

In order for an SFV to be useful, it must be consistent between frames. The segmentation algorithm in Chapter 4 creates robust SFVs, though the segments may not be consistent between frames. Consistency is achieved by doing data association and tracking over time - filtering out any segments that are not consistent. Once the segmentation is complete, the SFVs must be matched against learned objects, made up of one or more segments.

5.1 Data Association

Data association is required between sequential frames in order to determine whether a segment is consistent over time. Typically, data association refers to the task of assigning a label c_t to an unlabeled example x given past example(s) x_i and respective labels c_i. In this work, the examples paired with their respective labels are the the previous M temporally consistent frames, where M is set to 4.

Data association on SFVs has its own challenges in that each segment may not be unique to an object. There may be multiple SFVs that map to the same segment. For example, a piece of paper is a common object and shape (solving this problem for object matching is discussed below). Doing a simple nearest neighbors (NN) approach in SFV space leads to incorrect associations. A NN search around the centroid in XYZ space between frames is more robust, but fails if the segmentation is not strictly

Feature	Description
(U,V)	Particle coordinates in image space.
PREV	Ordered vector of previous M SFVs.
ID	Ordered vector of previous M segment IDs.
L	Lifetime counter

Table 5.1: Table of features in the particle data structure

consistent. Below we discuss an approach that works well in practice.

5.1.1 Temporal Filtering

We implement data association in this work by using a weighted random particle voting scheme, where a particle is defined in Table 5.1. For every frame, a set of particles, P, is instantiated in image space at random locations within a grid cell, with a single particle per grid location. The location within the grid is random and changes frame to frame to account for thin segments, thus allowing for a sparser particle distribution. For computational efficiency with minimal degradation in performance, $||P||$ is set to be 1/64th of the image, so for a 640x480 RGB-D image, that is 4800 8x8 grids. Each grid stores the following particle features.

The particles are initialized to the first frame's SFV and given unique integer IDs γ. With each new frame, every particle votes for a different ID with one vote per previous SFV weighted by the age of that SFV (at most M) and the similarity between that SFV and the current unlabeled SFV. To determine the similarity of two SFVs, we use a comparison-scoring function which returns 1 if the same SFVs are compared and the function is strictly monotonically decreasing for larger differences between any individual feature. To obtain these properties, the value of the i-th feature within the j-th SFV of PREV (higher j values are older) is modeled as a normal distribution with μ_i and $\sigma = \beta * \mu_i$. The score of a feature is chosen to be $(1 - A_i)$, where A_i is the area under the normal distribution from the $\mu_i - \delta_i$ to $\mu_i + \delta_i$ where $\delta = |\mu_i - \kappa_i|$ and κ_i is the i-th feature of the unlabeled SFV (see Figure 5-1). The total score for a particle p and age j, $W_{p,j}$, is all feature scores multiplied together and the result is weighted by α^{j-1}, with α determining how influential older votes are. This is done

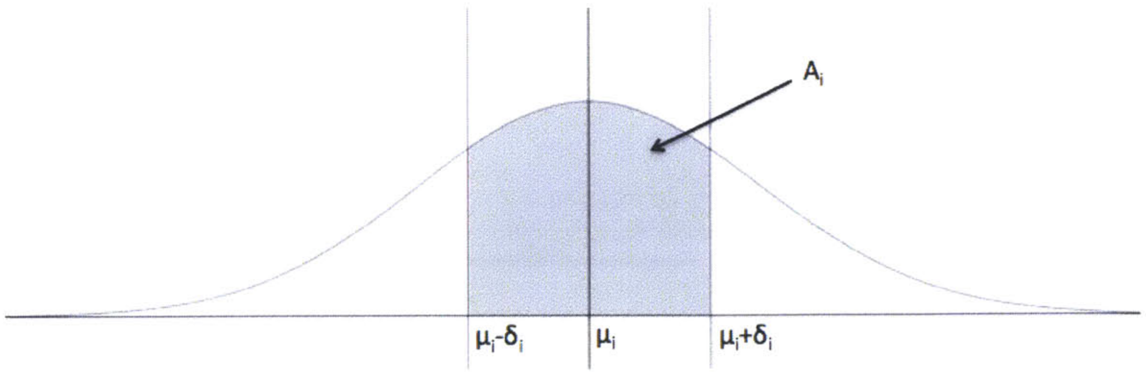

Figure 5-1: Individual feature scoring

for all M SFVs in all particles P_s within a segment s and the final label is the ID with the maximum vote. Values for α and β of 0.5 and 0.05 respectively work well in practice. Equation 5.1 shows this in detail.

$$\begin{array}{lcl} \forall s \in S & & \\ \gamma_s & = & \arg\max_\gamma(\sum_{p \in P_s} \sum_{j=1}^{M} (W_{p,j} * I(ID_p[j] == \gamma)) \\ W_{p,j} & = & \alpha^{j-1} \prod_i (1 - A_i) \\ A_i & = & \int_{\mu_i - \delta}^{\mu_i + \delta} \frac{1}{\sigma_i * \sqrt{2\pi}} * e^{\frac{(x-\mu_i)^2}{2*\sigma_i^2}} dx \end{array} \tag{5.1}$$

Where $I(...)$ is an indicator variable expressing whether the weight is included or not for a specific γ and S is the set of all segments. Intuitively, the individual feature weights in equation 5.1 can be thought of as measures of how similar two features are. The weighted vote $W_{p,j}$, which is the product of the feature weights, means that if any one feature is significantly different, then the vote is small. The new ID of segment s, γ_s, is determined to be the ID γ with the largest total vote across all weighted votes, i.e. the current segment that overlaps the previous segment best as determined by the scoring function. While Equation 5.1 uses the area of a normal distribution which suggests A_i is a probability, it is not a true probability since β is a parameter. As an optimization, each SFV pair is stored in a table and checked before every new computation since many of these calculations are duplicated.

Once every particle has voted, it may be the case that due to motion or a flawed segmentation, that segment ID bleeding occurs (i.e. many segments are given the same ID). The segments are assigned IDs in order from largest to smallest, with each segment taking the maximum vote count, and if that ID has not been taken, choosing the corresponding ID and updating every particle within the segment. If the ID has been taken, a new ID is set, and the particles are updated. This handles the case where, due to camera or dynamic object motion, new segments come into frame.

5.2 Consistent Segment Filtering

In order to produce a consistent segmentation, non-consistent segments need to be filtered out, i.e. all SFVs that are not stable over short periods of time. The lifetime counter in each particle is initialized to 1 at the start, and is incremented every time the same ID is assigned to a particle. Whenever a particle is reassigned, the counter is set to 1. Also, if the maximum value of the weighted voting normalized by the number of votes is below a threshold τ, the lifetime counter is set to 1. τ is set to 10^{-6} in practice and accounts for segments that are scored quite differently from the previous segments.

Consistency is now determined simply as having an average lifetime counter greater than L over all particles within the segment. This implies that quickly moving to a new scene will take L frames for the consistent segments to be declared salient. When the scene is not moving, the algorithm filters out unstable segmentations since either the lifetime of the segments will be less than L, or the segment changes greatly between frames and is reset because of τ (Figure 5-2). L is set to 3 in practice.

5.3 Objects

To fully utilize the segmentation feature method, it must be used not just for consistency between frames, but also for recognizing objects. For that, objects must first be defined in a general model or template to learn. We use a segment template (ST) and edge template (ET) to describe part or the entire object. STs store learned statistics

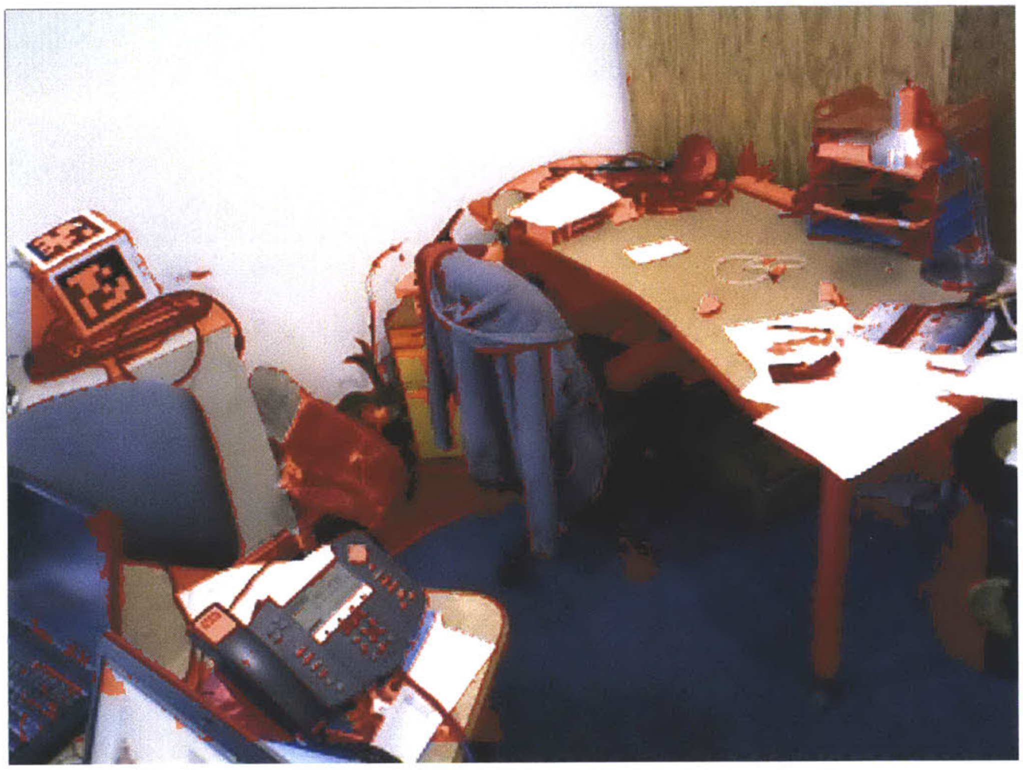

Figure 5-2: An RGB-D image with the unstable (or cluttered) regions highlighted in red. The areas not highlighted in red are deemed stable.

of an object so that SFVs can be matched against the STs in real time. STs are feature vectors including all the features in a SFV except the coordinate-based features (surface normal and centroid), but with also including the standard deviation of the feature. So each feature in the ST has a mean and standard deviation. ETs store the L_2 distance between two SFV centroids and the magnitude of the dot product between the two SFV surface normal vectors along with their respective standard deviations. The mean and standard deviation are both calculated during the learning stage described in Section 5.3.1. The intuition is that an object can be made up of one ST and no ETs (ex. a single piece of paper), or can be made up of multiple STs and ETs (ex. a file cabinet with multiple drawers). An object model is as follows:

- Set of STs consisting of the entire object.
- Set of corresponding ETs between every ST.

Since the centroid positions and normal vectors require the same coordinate frame between SFVs, only comparisons that have the segments in the same reference frame are allowed (i.e. both segments have to be detected in the same frame). As with the STs, both the average and standard deviation of the edge features are recorded.

5.3.1 Learning Objects

An object is defined by a set of connected segment(s). This definition can include other segmentations that semantically are not paired together depending on whether segments are grouped together while they are being learned. We use supervised learning over labeled video sequences to fill in the object template described in Section 5.3. Through a video sequence (Figure 5-3), the segmentation from the image is shown to a user (Figure 5-4), who labels one or more segments that represent an object by selecting it in a frame (Figure 5-5). The segments are tracked through the video by their labels and statistics are gathered on each segment until one of the segments is outside the frame or is inconsistent. The average value of the feature and the standard deviation are recorded into the template. For every frame with all labeled segments in

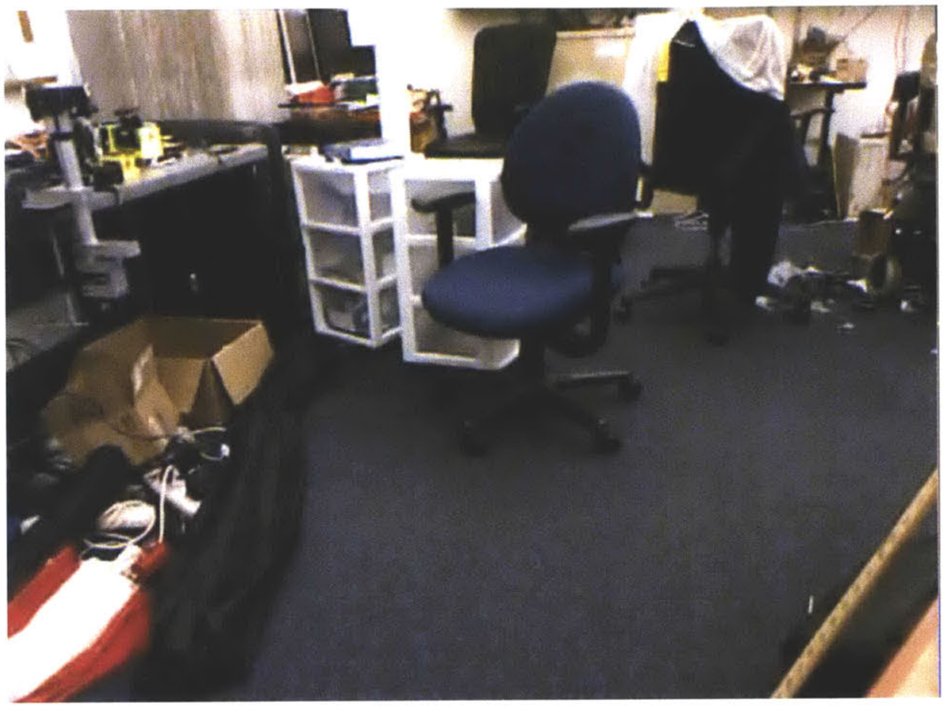

Figure 5-3: Initial RGB-D image for supervised learning on a chair.

view, the statistics on the fully connected edges are recorded by also fitting Gaussian distributions. Overfitting is a problem if only a few frames are available from the same viewpoint, thus having tight matching requirements, but the input training data has been collected to provide multiple angles as seen in Figures 5-3 and 5-6.

The result of this module, run off-line, is a set of object templates. Each object template is made up of N STs and $0.5 * N * (N-1)$ ETs for N > 0 (an edge between every ST).

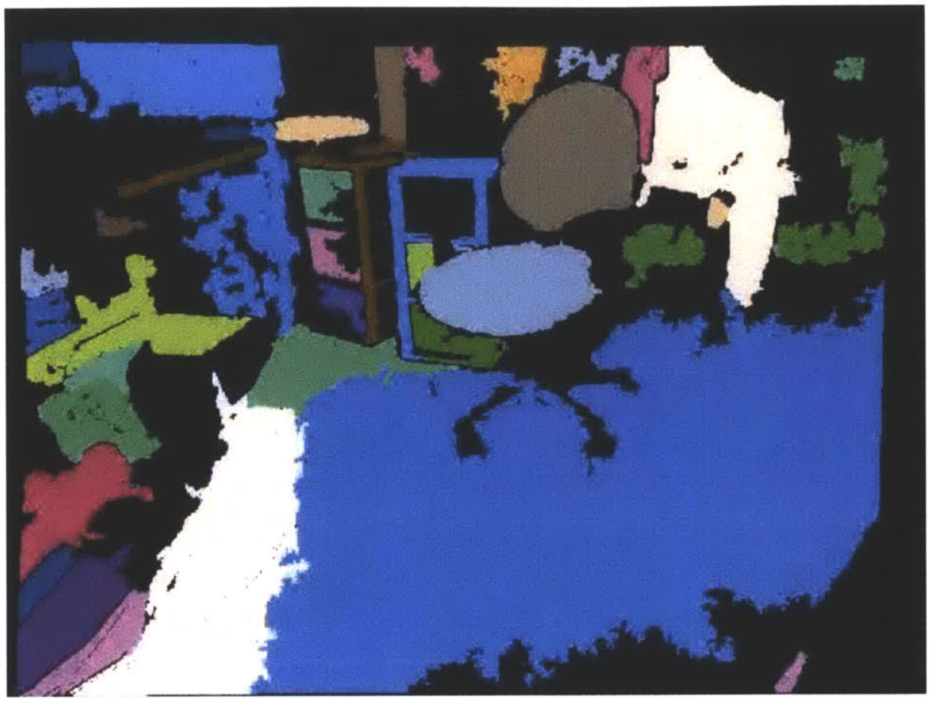
Figure 5-4: Segmentation of RGB-D image from Figure 5-3 colored by unique segment ID.

5.4 Matching

Once the segmentation is complete, the matching module has to check it against the learned models from Section 5.3.1. The outputs of this module are the matched SFVs with their corresponding object labels. The naive way of matching against the object model is by comparing every pair of segments in a frame, every pair of STs in the model, and every edge for every object ($O(O_j^2 * S_k^2 * |E|)$, with O_j being the j-th segment in the object, S_k being the k-th segment in the frame, and $|E|$ being the magnitude of all the edges). One constraint is that S_k is limited to only segments not along the border of the frame since border segments are not fully visible, and thus not useful for matching full objects. For efficiency, we used the following algorithm to match objects ($O(O_s * S_k + O_s^2 * |set_j| * |set_l|)$ where set_j and set_l are the sets of segments from a frame that match the j-th and l-th object model STs respectively). The matching algorithm for each object is detailed in Algorithm 2 and an example is shown in Figure 5-8. O is the set of STs in the object model, and

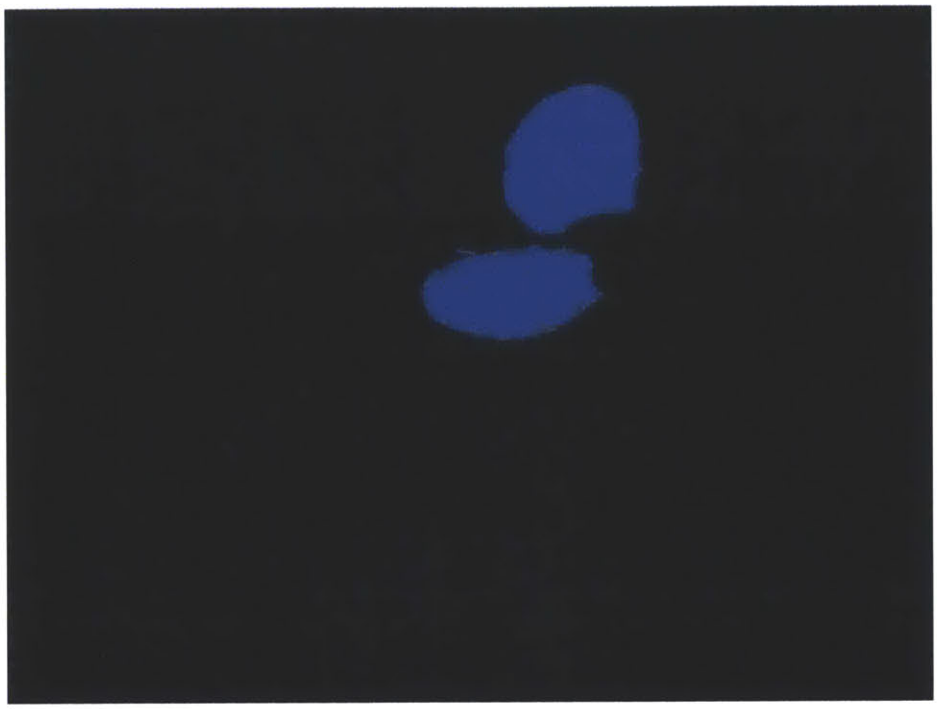

Figure 5-5: Seeded object selected by user from segmentation in Figure 5-4. The object consists of two separate segments that have been set to the pre-set tracking color.

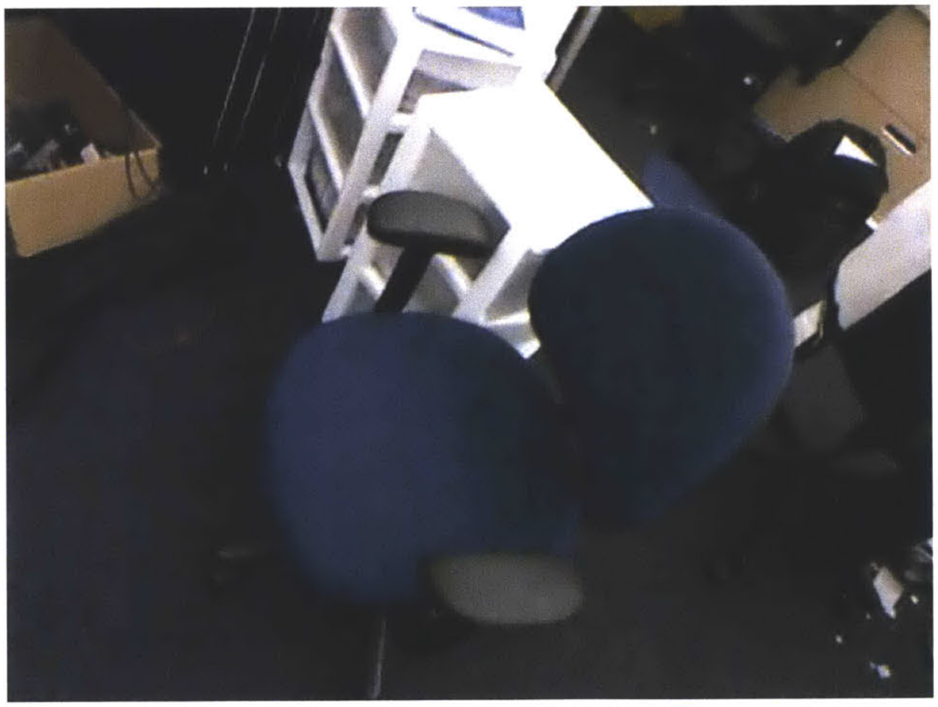

Figure 5-6: Later view of the chair in the sequence from a different viewpoint.

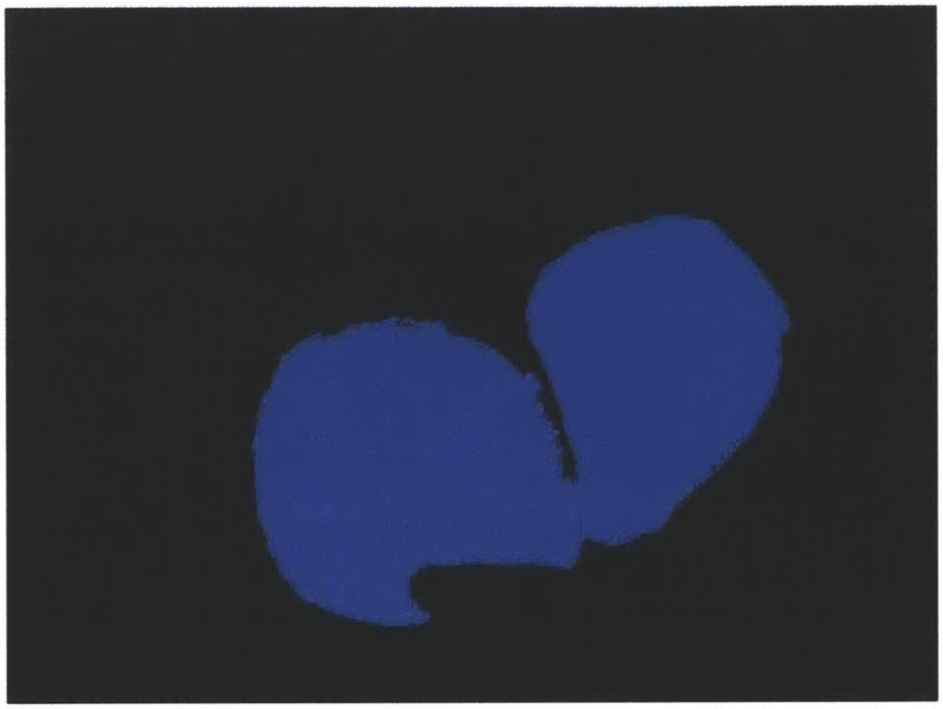

Figure 5-7: Temporal segment labeling still tracking seeded segmentation from Figure 5-5.

$counter_o$ is the counter for how many edges between matching segments match for an object. SegmentMatch and EdgeMatch are described below in sections 5.4.1 and 5.4.2 respectively.

Now the question arises of when to decide whether an object is found. Requiring every part to match might not allow for partial occlusions or vastly different viewpoints, conversely, requiring only one segment to match might might lead to many false positives. Also, it is not enough to define a global threshold since objects vary greatly in the number of segments. If an object model has multiple segments, then $counter_o$ will be large, while if the object matched has only one segment, then $counter_o$ will be 0. As such, the *Object_Match* function depends on the number of segments in the object O in addition to $counter_o$. We say an object is found if $counter_o >= ||O|| - 1$, which has the convenient property that it returns 0 if there is 1 segment in the object, 1 if there are 2 segments in the object (a single edge), and beyond 2, offers a relaxing requirement for $counter_o$. The intuition is that larger

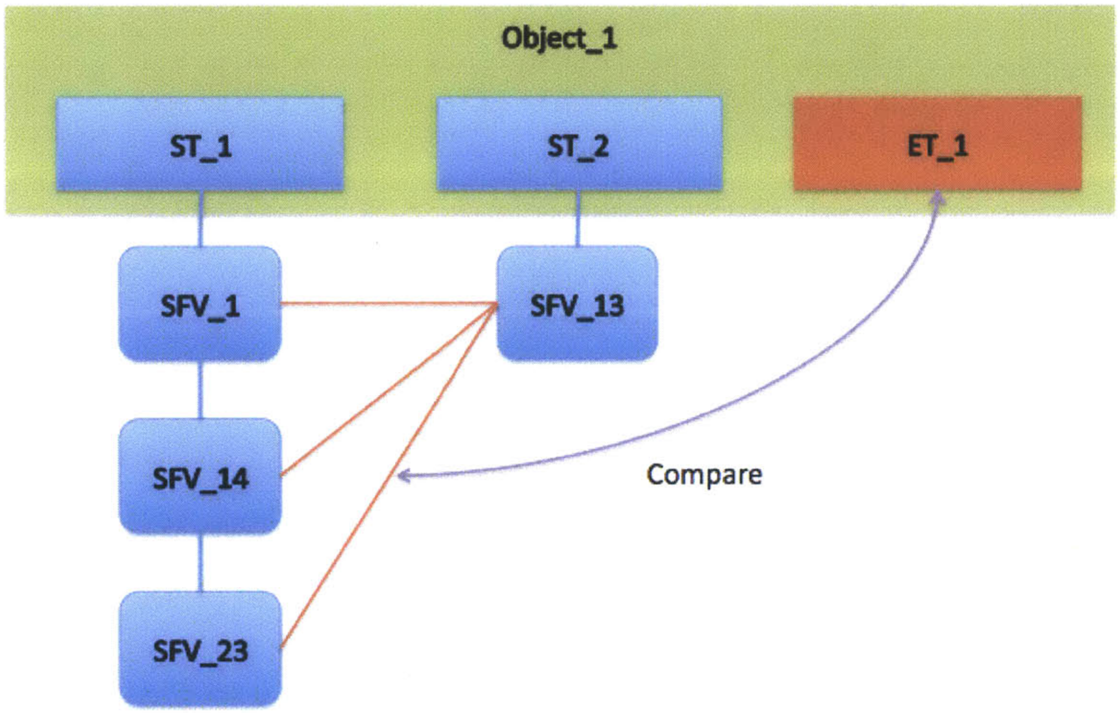

Figure 5-8: Matching Example. Given an object Object_1 with two STs, and one ET, the algorithm looks through each frame to find SFVs that match each ST. In this example, SFV_1, SFV_14 and SFV_23 all match ST_1, and SFV_13 matches ST_2. Then the edges (in red) are defined between the ST_1 and ST_2 candidates and every edge is compared against ET_1 to determine if the SFVs are part of the object. Not all edge comparisons are shown.

Algorithm 2 Object matching algorithm

```
/** for every ST in the current object template O*/
for Every ST $st_j \in$ O do
  /** for every SFV in the current frame */
  for Every SFV $sfv_i \in$ Frame do
    /** If the SFV matches ST, add it to a matching set for that ST */
    if SegmentMatch($st_j$, $sfv_i$) $> \tau_1$ then
      $set_j = set_j \cup sfv_i$
    end if
  end for
end for
$counter_o = 0$
/** for every ST in the current object template and every matching SFV in the
matching set for the current $st_j$ */
for Every ST $st_j \in$ O do
  for Every $j \in set_j$ do
    /** for every other ST in the current object template and every matching SFV
    in the matching set for the new $st_l$ (could be the same ST as $st_j$) */
    for Every ST $st_l \in$ O do
      for Every $l \in set_l$ do
        if ($j == l$) then
          continue
        end if
        /** Define the edge between SFVs $j$ and $l$*/
        Edge $E_{j,l}$ = edge between $j$ and $l$

        /** If the edges match, increment the counter */
        if EdgeMatch($E_{j,l}$, $Model\_Edge_{st_i,st_l}$) then
          $counter_o$++
          Add j, l to matching_objects
        end if
      end for
    end for
  end for
end for

/** If enough of the edges match, return the set of object SFVs, otherwise, return
null */
if (!Object_Match(O, $counter_o$)) then
  exit
end if
return matching_objects
```

clusters of segments are far more unique than matching a few segments, so matching a subset is acceptable.

5.4.1 Segment Matching

The SegmentMatch function in Algorithm 2 is a similar problem to determining similar segments in the temporal filtering module (Section 5.1.1). The new comparison scoring function is normalized so that the new function returns 1 when the same segment is compared to itself and the function is strictly monotonically decreasing for any difference from the mean. For this scoring function, we define i as the index within the feature vector, j as an ST within an object model, k is the SFV of the segment within the frame, and $\delta_{i,j,k}$ is $||\mu_{i,j} - \kappa_{i,k}||$, where $\kappa_{i,k}$ is the value of the i-th feature in k. Again using a normal distribution, when a feature value s_i is compared against an ST feature, the score of a single feature is the area under the normal distribution defined by $\mu_{i,j}$ and $\sigma_{i,j}$ from $\mu_{i,j} - \delta_{i,j,k}$ to $\mu_{i,j} + \delta_{i,j,k}$.

$$\begin{aligned} score_{j,k} &= \textstyle\prod_i (1 - A_{i,j,k}) \\ A_{i,j,k} &= \int_{\mu_{i,j}-\delta i,j,k}^{\mu_{i,j}+\delta i,j,k} \frac{1}{\sigma_{i,j} * \sqrt{2\pi}} * e^{\frac{(x-\mu_{i,j})^2}{2*\sigma_{i,j}^2}} dx \end{aligned} \tag{5.2}$$

If the value of $score_{j,k}$ is greater than a threshold τ (0.25), then the segment is stored into a set set_j specific to the segment with ST j. The score is calculated and the segments sorted between every SFV k and every ST j. It is reasonable to compare SFVs and STs since every feature i in an ST is the same feature i in an SFV for $i \in 0, ..., |ST|$

5.4.2 Edge Matching

Once the set of segments within an object are picked out from the frame, the edges are then checked in a similar manner. Every segment within set_j is paired with another segment from within set_l for all $l \neq j$. The edge feature vector (the L_2 distance between centroids, and dot product of surface normal vectors) between j and l is calculated and compared to the ET as in equation 5.2, with $i \in 0, ..., |ET|$, j as the

ET, k as the edge feature vector, and τ as 0.75.

The edge matching leaves ambiguities in 3D space due to only constraining segments to a sphere around the centroid of a segment with a cone of surface normals along the sphere. While this has the potential for false positives, the uniqueness of the segments combined with edges makes it a minor problem in practice for objects with edges. A more detailed analysis is in the next chapter.

Chapter 6

Results

The implemented system allows a robot to recognize categories of objects in real-time. All of the details discussed in the previous sections have been implemented and merged into a working prototype. The RGB-D video sequences used to evaluate this work were collected in three scenarios: a stationary camera above a table top, a handheld camera taken through an office, and a robot mounted camera driven through a building. The next several sections describe the results of the segmentation and object matching method.

6.1 Segmentation

To evaluate the segmentation algorithm, some form of 'ground-truth' labeling is generally needed. However, defining a 'correct' segmentation is subjective, so there is no such ground-truth measure. Instead, we measure the consistency of the objects between frames. In our first scenario, an RGB-D camera is placed looking at a table top scene with variously shaped and colored objects, as seen in Figure 6-1. The segmentation output f_i for some frame i is recorded and all future segmentations are compared against f_i (for example, Figure 6-2). The comparison is done by taking the largest consistent segment $s_m \in f_j | j \geq i$ that intersects the segment $s_n \in f_I$ (for all m and n segments in f_j and f_i respectively) and computing the similarity between pixel masks the two segments using the Jaccard Similarity Index $\frac{s_n \cap s_m}{s_n \cup s_m}$ [41]. The similarity

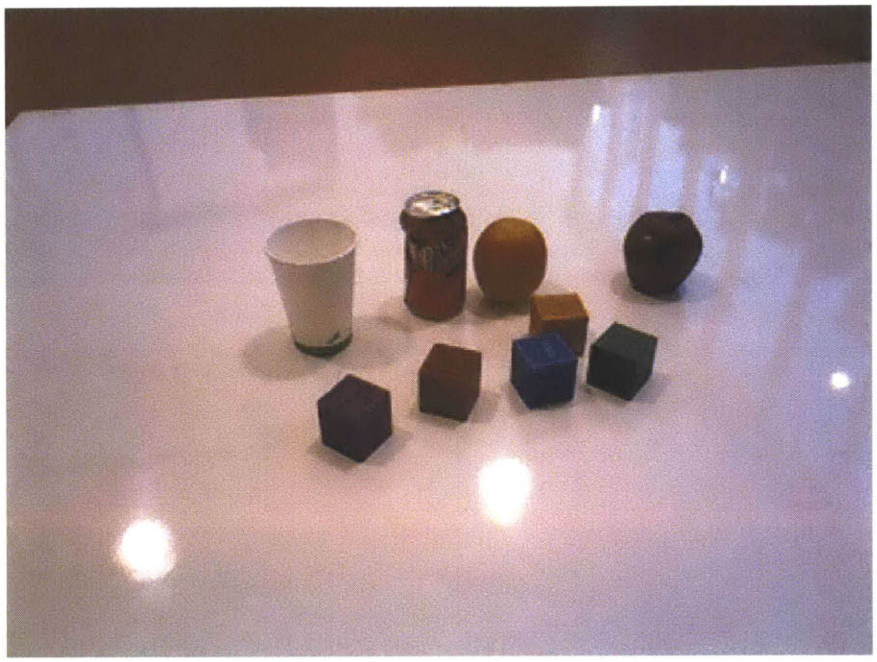

Figure 6-1: RGB Image of a tabletop scene.

j =	**Similarity score (all segments)**	**Similarity score (labeled objects)**
i	1.0	1.0
$i+1$	0.85	0.95
$i+5$	0.85	0.93
$i+15$	0.83	0.94
$i+30$	0.84	0.93
$i+90$	0.82	0.93

Table 6.1: Table of similarity scores averaged over 25 video sequences.

measure returns 1 if two regions are exactly overlaid and 0 if the two regions have no overlapping consistent segment. This is done for 25 choices of i and j was chosen as $i+1, i+5, i+15, i+30, i+90$ for every i. The results of the segmentations were averaged over all i and n, and put into Table 6.1 for both the entire segmentation and a set of human-labeled object segments across 8 varying object positions and camera orientations. The human-labeled objects were recorded by having a user select pre-segmented objects from our method at a starting frame rather than selecting all segments.

The values in Table 6.1 are dependent on the objects used in Figure 6-1 and the

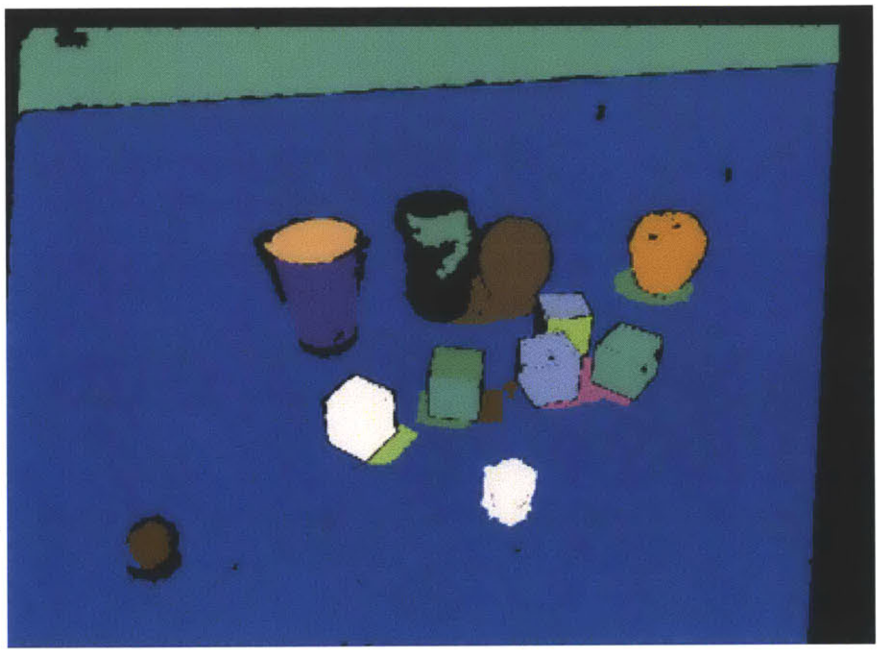

Figure 6-2: Segmentation output of the RGB image in Figure 6-1. Note the shadows and light reflections that segmented separately, with the former being unstable due to the small color gradient. The crumpled metal soda can is half labeled as inconsistent since the odd angles and shiny surface do not provide stable values in the depth image.

Units: changes per segment	**Office 1 (36 sec)**	**Office 2 (55 sec)**	**Hallway Traverse 1 (83 sec)**	**Hallway Traverse 2 (54 sec)**
All segments	7.3	8.5	5.1	4.9
Segments longer than 12 inches	1.6	1.9	0.9	1.4

Table 6.2: Table of the average number of label changes per segment over the length. Larger segments (defined here to have a principle axis longer than 12 inches) have more stable IDs over time than smaller segments.

viewing scene, but since the segmentation method is independent of previous frames, the similarity values show little difference between the following j frames. The primary source of variance in segment consistency in the data was in the shadows of the objects, which helps to explain why the labeled-object results are comparatively higher. The rest of the variance is from the sensor noise and randomness in the underlying segmentation method. Performance decreases as clutter in the scene increases, since aliasing occurs and the segmentation method breaks down.

Determining the consistency of the segmentations from the handheld and robot-driven datasets are difficult to determine quantitatively since the viewpoint is always changing, so raw pixel masks will not work. We indirectly determine consistency by using the consistency module described in Chapter 5 and measuring the number of segment ID changes within the range and FOV of the sensor. The intuition is that segments that are consistent will have fewer ID changes. The results are averaged over all segments in each data set listed in Table 6.1. Due to variations in motion and time (listed in seconds), we show the results from multiple data sets separately.

6.2 Object Matching

The object category matching method was tested by first learning a set of objects (chair, file cabinet, door, 11"x8.5" paper, and refrigerator) and then matching them against a labeled data set. The objects were learned using a single training example and collecting statistics on its features from different angles (as described in Section 5.3.1). The data sets contained objects qualitatively similar, but not identical,

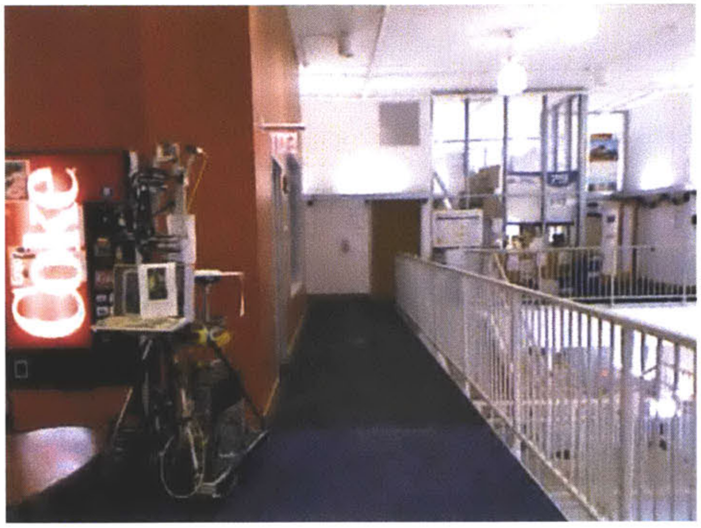

Figure 6-3: RGB Image of a corridor in the Stata Center during Hallway Traverse 2.

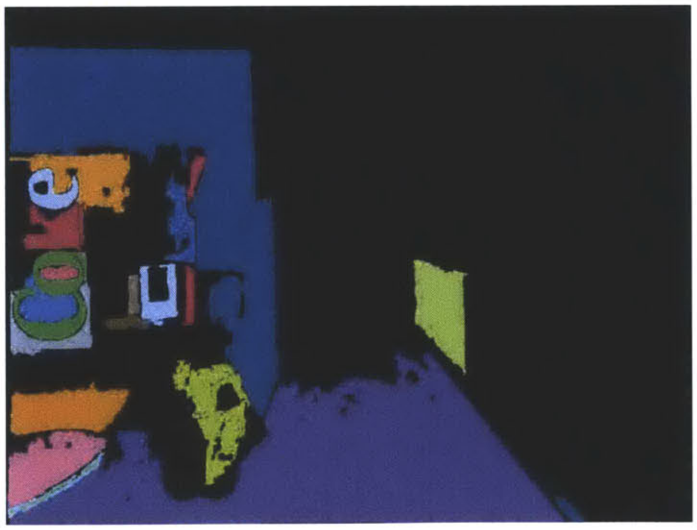

Figure 6-4: Example segmented image of Figure 6-3.

Object Category	False Positives	False Negatives
Chair	5%	68%
File Cabinet	7%	27%
Door	0%	60%
Paper	53%	14%
Refrigerator	0%	22%

Table 6.3: Table of the false positives and false negatives. False positives are the number of frames a false detection occurred in divided by the total number of frames there was a detection for that object. The false negative percentage is the number of frames the object was in the camera's FOV and range, and there was no positive detection.

features, as seen in Figure 6-8 and Figure 6-14.

Across multiple handheld and robotic traverses, we collected data on the false positive and false negative percentages for each of the objects. We did this by counting the number of detections against our human-labeled data sets for every frame. We only counted objects that were within the camera's FOV since our algorithm throws out all segment features touching the edge. The results are listed in Table 6.2 for over 15 minutes of traverses.

The percentages for paper listed in Table 6.2 are particularly bad since the size and shape of paper is common. Many graphs on posters, or rats' nests of cables may be segmented in such a way that it passes the detection threshold. For the chairs, the amount of variance between different chairs is significant and thus the single training example is not general enough. When trained on four chairs of wide variance, the false positive rate increased to 37% and the false negative rate dropped to 42%. For qualitative evaluation, Figures 6-5, 6-6, and 6-7 show an example of a door being detected. Figures 6-8, 6-9, and 6-10 show the scale invariance of the segment matching method when compared to Figures 6-11, 6-12, and 6-13. Figures 6-14, 6-15, and 6-16 show a different cabinet being matched. Figures 6-17, 6-18, and 6-19 show matches for drawers of a file cabinet, but the edges between the drawers do not match so the drawers are not labeled as a file cabinet. All examples shown and discussed use one object category template for all object matches.

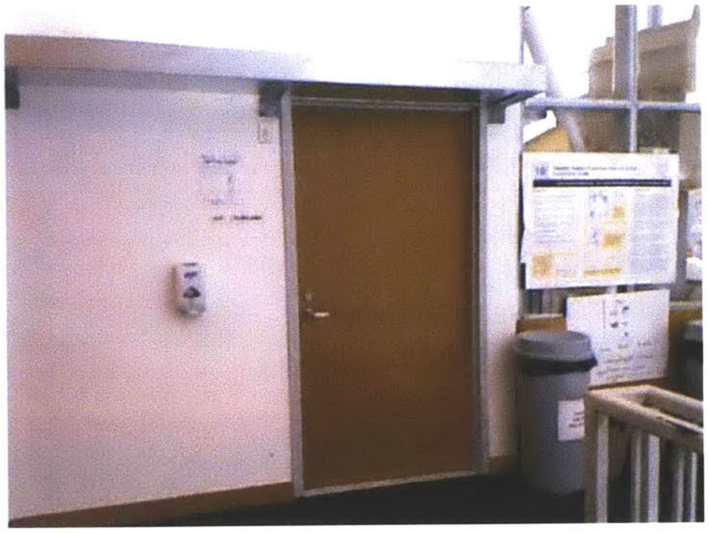

Figure 6-5: RGB Image of a door.

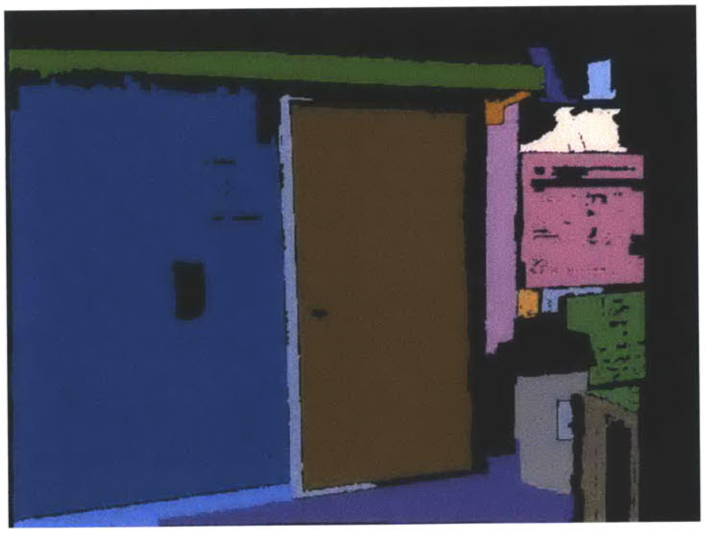

Figure 6-6: Segmented image of Figure 6-5.

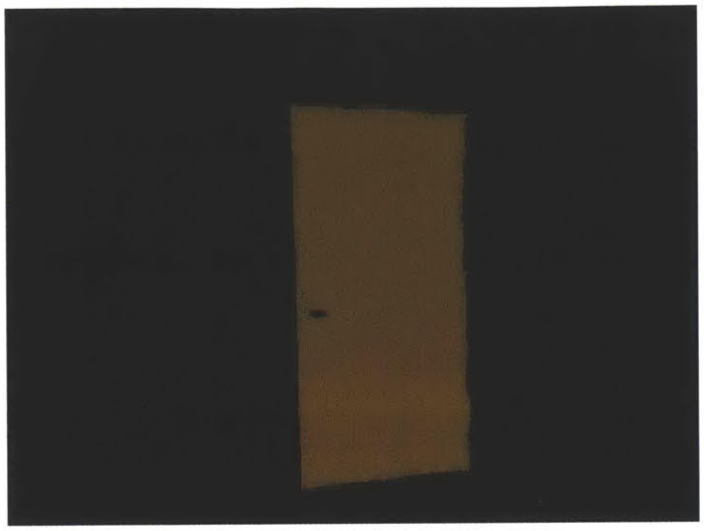

Figure 6-7: Matched door segment of Figure 6-6.

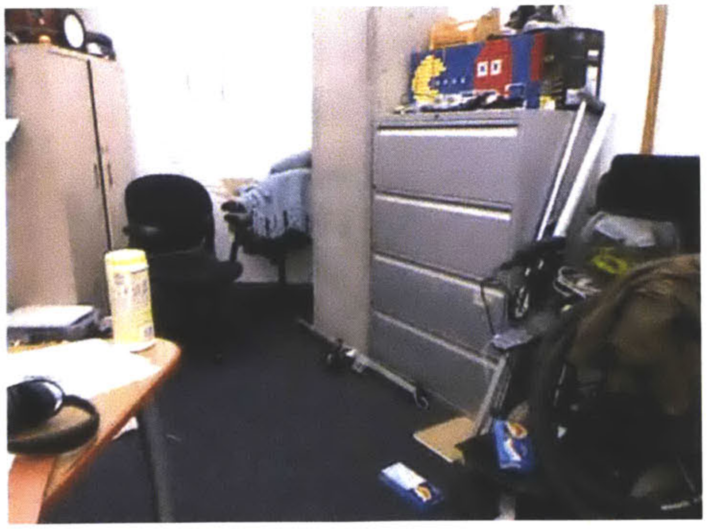

Figure 6-8: RGB image of a far cabinet.

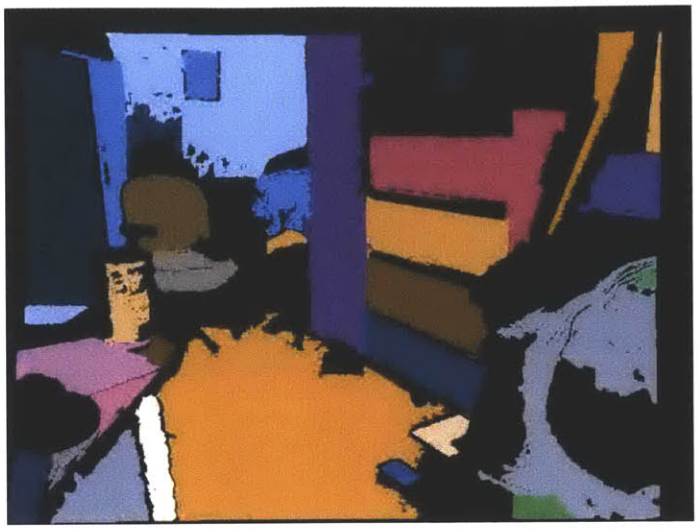

Figure 6-9: Segmented image of Figure 6-8.

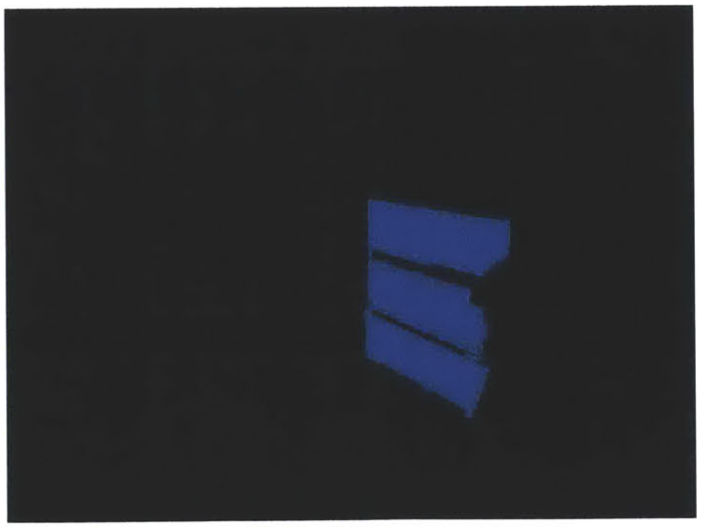

Figure 6-10: Matched cabinet from Figure 6-9.

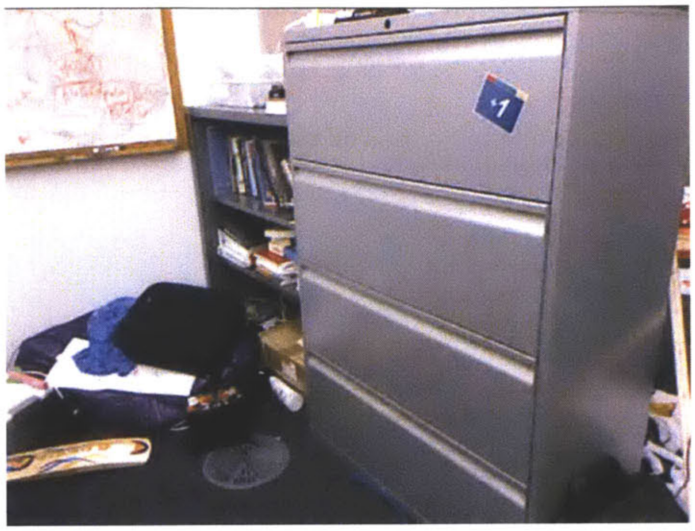

Figure 6-11: RGB image of a close cabinet

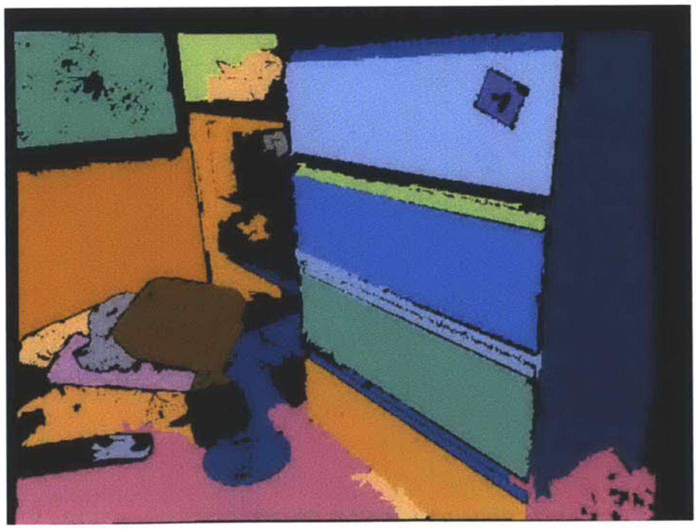

Figure 6-12: Segmented image of Figure 6-11.

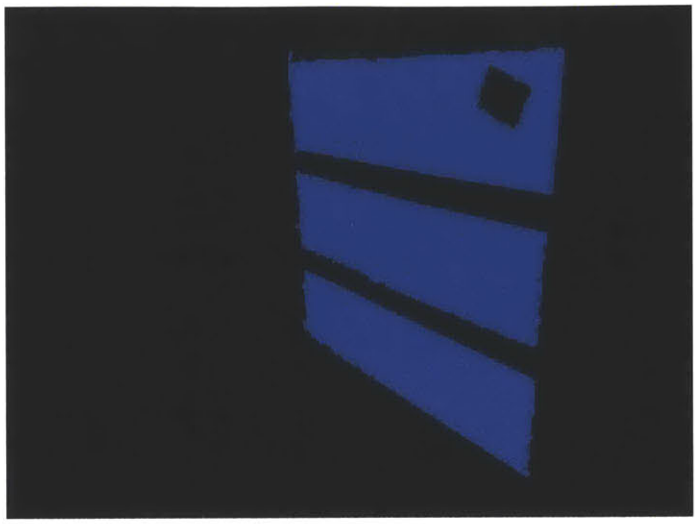

Figure 6-13: Matched cabinet from Figure 6-12.

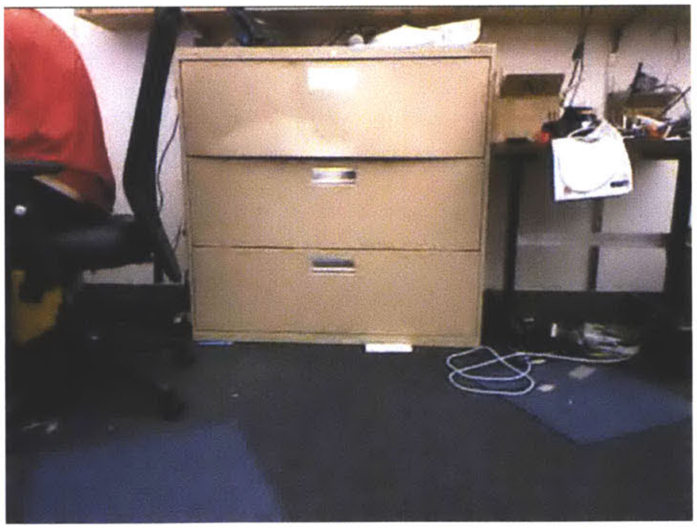

Figure 6-14: RGB image of a different, wider cabinet.

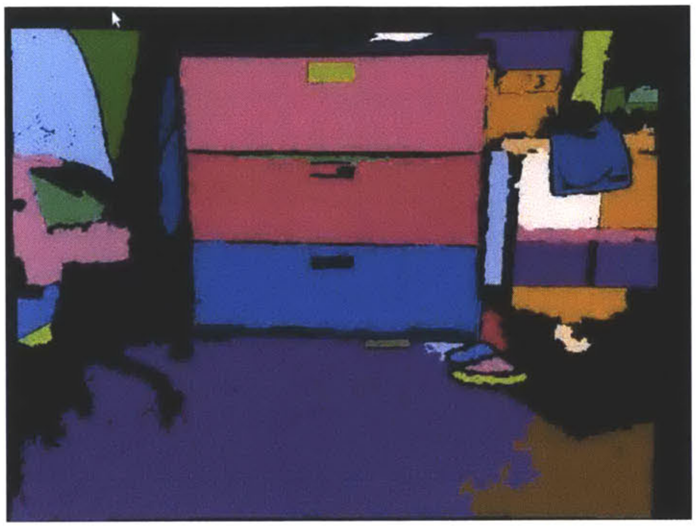

Figure 6-15: Segmented image of Figure 6-14.

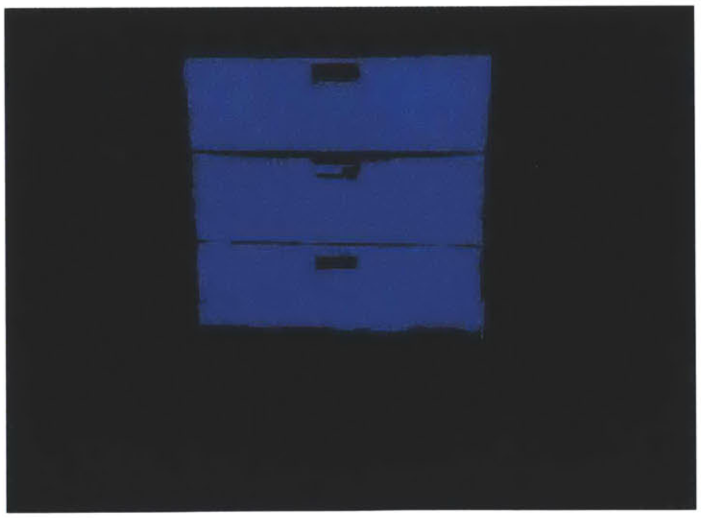

Figure 6-16: Matched cabinet from Figure 6-15.

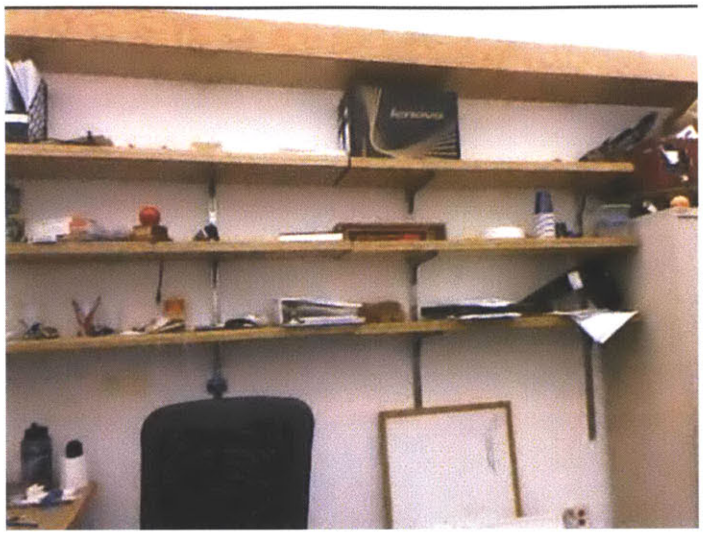

Figure 6-17: RGB image of false positive segments.

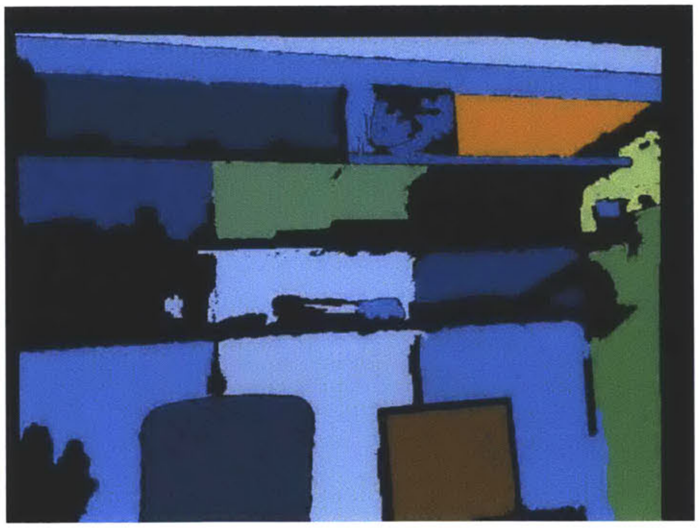

Figure 6-18: Segmented image of Figure 6-17.

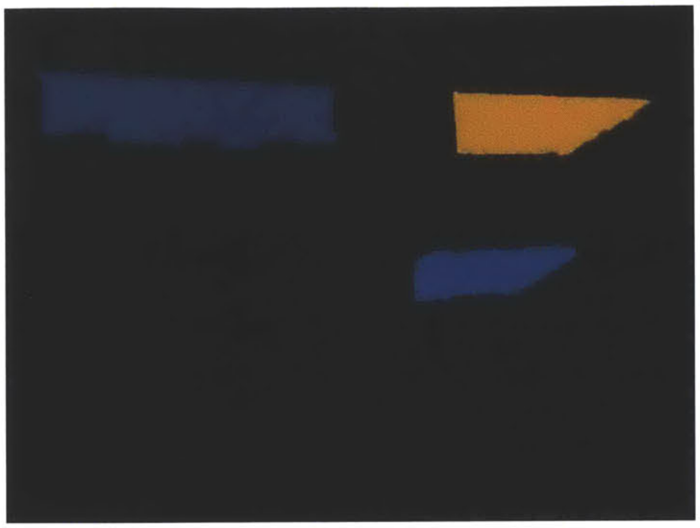
Figure 6-19: Potential matches highlighted from Figure 6-18, but the edges do not match up, so they are not colored the object category color blue.

The system was further tested by driving a robot around an office building with the object category detectors running. One of the paths taken is shown in Figure 6-20. In these trials, a robot was manually driven around with a forward facing kinect sensor. Figure 6-21 shows an example of a detected file cabinet. Figure 6-22 shows an image of a false positive detection for a door, but the cubicle wall has roughly the same dimensions as standard door.

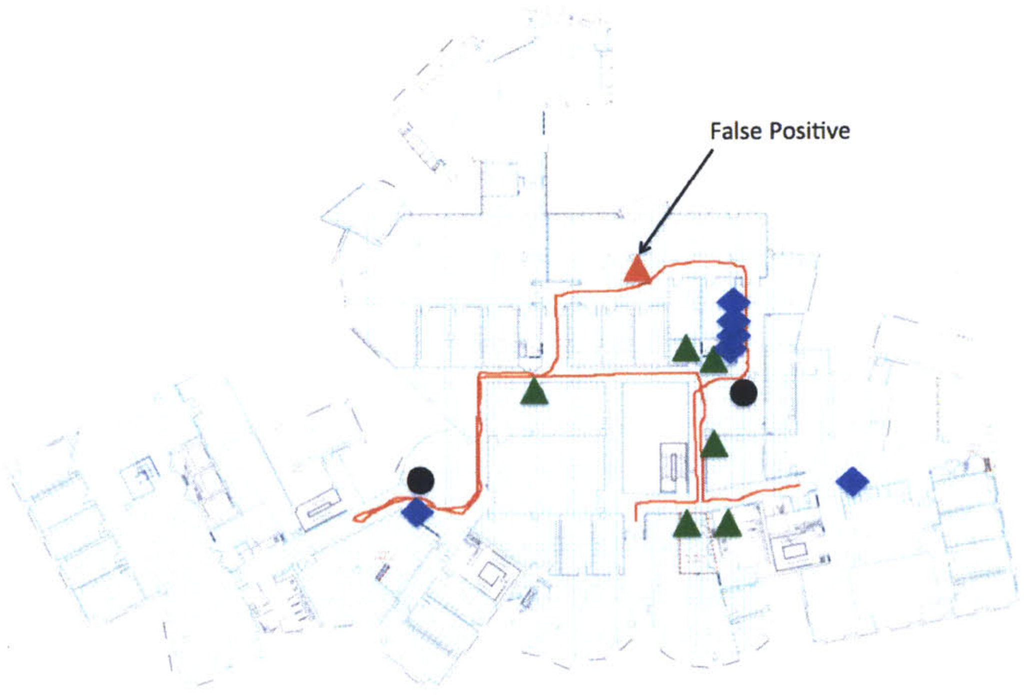

Figure 6-20: A labeled trajectory (in red) around the third floor of the Stata building of the Envoy robot highlighting detected objects. Green triangles are detected doors, blue diamonds are detected file cabinets, and black circles are detected chairs. A false positive on the door detector is highlighted as a red triangle and shown in Figure 6-22.

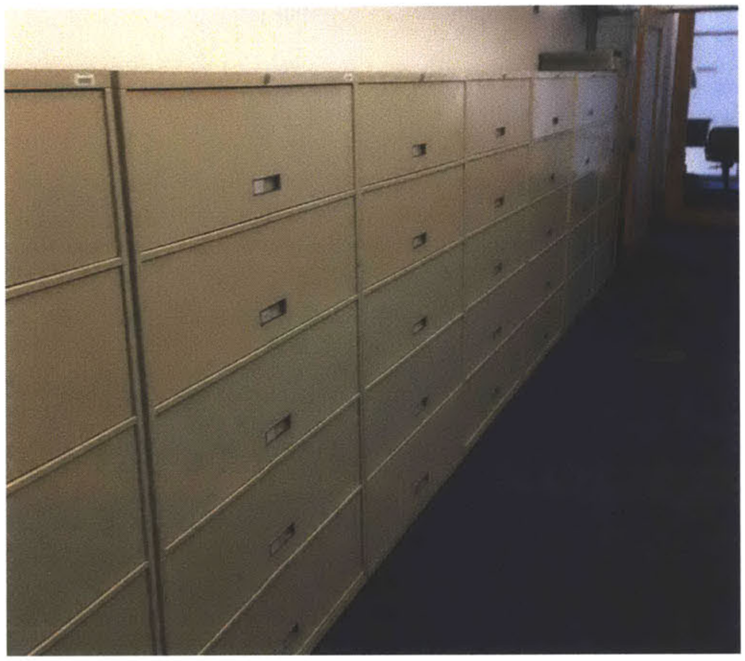

Figure 6-21: RGB image of detected cabinets during the traverse shown in Figure 6-20.

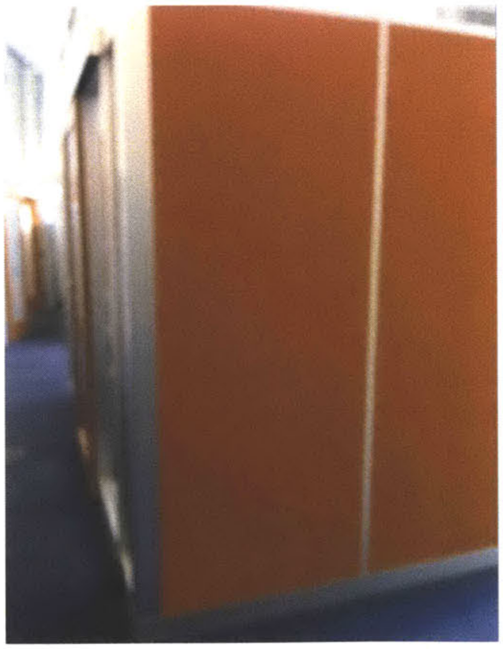

Figure 6-22: RGB image of a false positive detection for a door from the traverse shown in Figure 6-20.

Module	**Time (ms)**
Build Graph	5
Contour Filtering	3
Initial Segmentation	35
Segment Joining	17
Feature Creation	29
Temporal Filtering	41
Consistent Segment Filtering	<1
Match Segments	<1
Match Edges	<1
Total	**133**

Table 6.4: Table of each module's average time over all traverses.

6.3 Run-Time Evaluation

The run time for the algorithm is listed in Table 6.4 for 640x480 resolution RGB-D images. The matching algorithm is fast since each feature vector is 47 bytes, so an entire image can be represented in a compressed format and therefore quickly compute object model comparisons across a whole image. In practice, the run-time of each method is consistent and the whole system outputs labeled object categories at 7-8 Hz on a 2.54 GHz quad-core Dell laptop.

Chapter 7

Conclusion

This thesis has presented a new approach to object category recognition that provides robust, real-time performance in complex indoor environments. The algorithm provides real-time RGB-D features using a segmentation-based feature creation method. The feature descriptor in this thesis is particularly useful due to its generalizability that encodes the approximate shape of an object. The features detected within an image are scale and rotationally invariant, and robust across wide changes in viewpoint. Their computation is efficient, running in real-time on a standard laptop. This thesis also presented a method using the described RGB-D segmentation features for object category recognition. The general feature descriptors and edges between them make category recognition possible for objects that are similarly shaped. The object category recognition approach we described runs in real-time and was validated on multiple real world data sets containing several categories of objects.

7.1 Future Work

Although the algorithm performed well for detecting object categories, it still has limitations due to occlusions, inconsistent segmentations, and computation speed that future work will address.

Handling occlusions within the camera FOV and along the border of the frame would increase the robustness of the matching algorithm. Currently, the object matching algorithm only works if the complete object is unobscured and within the FOV of the camera. We envision two higher level modules to handle such scenarios. First, each segment label keeps an additional SFV stored that records the maximum feature values of the SFV and the object templates are matched against the maximum SFV instead of the current SFV. The idea is that once the camera has seen the full segment, the system can remember the feature until the segment label is lost. Second, each frame can have multiple landmark segments to which all other segments are connected. Each landmark would store the relative location of the other segments and remember the other segment's SFVs to match against. There can be multiple landmarks per frame to handle occlusions of any subset of the landmarks. One metric for determining which segments are landmarks is the lifetime of the segment. The idea behind this module is that not all segments will be occluded so a few landmarks can remember the whole scene.

This work relies on a consistent segmentation algorithm to determine the SFVs, however the segments are not always consistent. Currently, most inconsistent segments are filtered out in the consistency filter module in the algorithm, but some are consistent for slightly longer than the thresholded number of frames and change their labels afterwards. These errors come out of both the segmentation and super-pixel joining modules. For the segmentation module, more sophisticated segmentation algorithms can be tested to compare consistency against the current implemented algorithm, however many segmentation algorithms do not run in real-time. Also, using a weighted smoothness term within the RGB segmentation algorithm could lead to improved results. The HSV distance in Section 4 could be augmented with an XYZ distance value and combined using some α weighting parameter. For the segment joining, currently if one pair of pixels has an HSV distance less than a global threshold, then the super-pixels are joined. A more advanced and principled approach could be taken instead of the current global binary joining threshold. One such way would be to run a new Felzenszwalb algorithm on only the edges of the super-pixels. The decision to join two segments can be made by computing the ratio of the number of edge segments to the number of total pixel edges between the two segments. The proof of whether the edge segments are too fine or too course would follow from Felzenszwalb's paper [23]. The intuition behind this is that super-pixels with high color variation along their borders will have more edge segments than super-pixels with little color variation along their borders so the ratio of edge segments to the number of pixel edges would be greater.

While the current system runs at 7-8 Hz on a standard laptop, the temporal filtering module could be improved by having a faster frame rate since there would be less motion between frames. Smaller objects change labels more frequently because there are fewer particles voting for them when the camera is in motion. The current implementation is on a CPU, however, all parts of the algorithm can be run in parallel except for the segmentation module (which has dynamic thresholds that can be changed every iteration). Implementing most of the algorithm on a GPU could speed the algorithm up to the camera's 30 Hz. From Section 6.3, the segmentation algorithm runs at approximated 30 Hz, however, that is under 100% CPU load with other processes competing.

In addition to the limitations of the current system, we will also discuss extensions of the system that will be done in future work.

The small size of each SFV (47 bytes) and the number of SFVs per frame (50 SFVs/frame on average) mean that a frame can be easily compressed into a simplistic format. Even if the algorithm ran at 30 Hz, the amount of data would be $47\frac{bytes}{SFV} * 50\frac{SFV}{frame} * 30\frac{frame}{second} \approx 70\frac{kB}{second}$. This can be used to stream a SFV feed from a mobile robot over an intermittent wifi connection to a server for more extensive computation. The location and 3D orientation is already stored within the SFV, so a simple blob reconstruction of the scene can be formed for a human user to view.

Finally, the system can be extended to do unsupervised learning of structure in indoor environments where structure is either an object or a grouping of objects. The system currently has to learn objects from human labeled data. While this process is simple, it would be useful for the robot to independently learn objects as it traverses a building. One way to achieve this goal would be to take the SFVs and group them into predetermined numbers of SFVs (say groups of two SFVs) with their corresponding edges (which are the same as used in this thesis: the L_2 distance between centroids and the dot product of the surface normals). The two SFVs and corresponding edge between them make a *Pair*. Then, every *Pair* in a frame should be clustered in *Pair* space and when enough are clustered together, a new object is spawned from that *Pair*. The object would then be grown to potentially include more SFVs by searching for other consistent SFVs in all frames which the spawned *Pair* was in. For example, say a robot sees many different chairs with seats and backrests. A *Pair* consisting of the seat and backrest SFVs is seen often enough to label it as an object. Then, the algorithm searches over all frames that contain seats and backrests and SFVs for armrests are seen nearly as often, so armrests are joined in as part of the object. So the final object learned in this example is a chair made up of a seat, backrest, and two armrests. This technique would find both objects and structure within an environment. Computer monitors and keyboards would probably be grouped together despite being separate objects. When to spawn a new object and when to grow an existing object would be determined by some threshold.

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 2012.

[2] H. Grabner, J. Gall, and L. J. V. Gool, "What makes a chair a chair?," in *CVPR*, pp. 1529–1536, 2011.

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[4] P. M. Roth and M. Winter, "Survey of Appearance-Based Methods for Object Recognition," tech. rep., Institute for Computer Graphics and Vision, Graz University of Technology, 2008.

[5] *British Machine Vision Conference, BMVC 2009, London, UK, September 7-10, 2009. Proceedings*, British Machine Vision Association, 2009.

[6] A. Torralba, A. Oliva, M. S. Castelhano, and J. M. Henderson, "Contextual guidance of eye movements and attention in real-world scenes: the role of global features in object search.," *Psychological Review*, vol. 113, pp. 766–786, October 2006.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, pp. 346–359, June 2008.

[9] A. Johnson, *Spin-Images: A Representation for 3-D Surface Matching.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[10] P. Viola and M. Jones, "Robust real-time object detection," in *International Journal of Computer Vision*, 2001.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *In CVPR*, pp. 886–893, 2005.

[12] P.-E. Forssén and A. Moe, "View matching with blob features," in *2nd Canadian Conference on Computer and Robot Vision*, (Victoria, BC, Canada), pp. 228–235, IEEE Computer Society, May 2005.

[13] *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*, IEEE Computer Society, 2007.

[14] D. Lowe, "Object recognition from local scale-invariant features," *International Conference on Computer Vision, 1999*, pp. 1150–1157, 1999.

[15] M. Walter, Y. Friedman, M. Antone, and S. Teller, "Vision-based reacquisition for task-level control," in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, (New Delhi, India), December 2010.

[16] K. Lai, L. Bo, X. Ren, and D. Fox, "Sparse distance learning for object recognition combining rgb and depth information," in *IEEE International Conference on on Robotics and Automation*, 2011.

[17] "Microsoft kinect. http://www.xbox.com/en-us/kinect."

[18] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1627–1645, 2010.

[19] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *Comput. Vis. Image Underst.*, vol. 110, pp. 260–280, May 2008.

[20] C. R. Brice and C. L. Fennema, "Scene analysis using regions," Tech. Rep. 17, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Apr 1970.

[21] T. Malisiewicz and A. A. Efros, "Improving spatial support for objects via multiple segmentations," in *British Machine Vision Conference (BMVC)*, September 2007.

[22] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 603–619, May 2002.

[23] P. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, Sept. 2004.

[24] D. Holz, S. Holzer, and R. B. Rusu, "Real-Time Plane Segmentation using RGB-D Cameras," in *Proceedings of the RoboCup Symposium*, 2011.

[25] D. Holz and S. Behnke, "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," in *International Conference on Intelligent Autonomous Systems*, (Jeju Island, Korea), 2012.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[27] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Computer Vision and Pattern Recognition*, June 2011.

[28] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

[29] K. Lai, L. Bo, X. Ren, and D. Fox, "Detection-based object labeling in 3d scenes," in *IEEE International Conference on on Robotics and Automation*, 2012.

[30] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient hierarchical graph based video segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[31] A. Abramov, J. Papon, K. Pauwels, F. Wrgtter, and B. Dellen, "Depth-supported real-time video segmentation with the kinect," in *IEEE workshop on the Applications of Computer Vision (WACV)*, 2012.

[32] L. Zhang, B. Curless, and S. M. Seitz, "Rapid shape acquisition using color structured light and multi-pass dynamic programming," in *The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission*, pp. 24–36, June 2002.

[33] C.-K. Liang, L.-W. Chang, and H. H. Chen, "Analysis and compensation of rolling shutter effect," *Trans. Img. Proc.*, vol. 17, pp. 1323–1330, Aug. 2008.

[34] "Softkinetic depthsense. http://www.softkinetic.com/solutions/depthsensecameras.aspx."

[35] S. Hemachandra, T. Kollar, N. Roy, and S. Teller, "Following and interpreting narrated guided tours," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Shanghai, China), 2011.

[36] "Openni. http://www.openni.org."

[37] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[38] A. R. Smith, "Color gamut transform pairs," *SIGGRAPH*, May 1978.

[39] G. Strang, *Introduction to Linear Algebra.* Wellesley-Cambridge Press, second ed., 1998.

[40] M. E. Wall, A. Rechtesteiner, and L. M. Rocha, *Singular Value Decomposition and Principal Component Analysis*, pp. 91–109. A Practical Approach to Microarray Data Analysis, Academic Publishers, 2003.

[41] P. Jaccard, "Etude comparative de la distribution orale dans une portion des alpes et des jura," in *Bulletin del la Socit Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.