Using the structure and motion of stereo point clouds for the semantic segmentation of images

by

Matthew Dockrey

B.Sc., The University of Washington, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in
The Faculty of Graduate Studies
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)
December 2009
© Matthew Dockrey 2009
Abstract

The segmentation of images into semantically coherent regions has been approached in many different ways in the over 40 years since the problem was first addressed. Recently systems using the motion of point clouds derived from laser depth scanners and structure from motion have been described, but these are monetarily and computationally expensive options. We explore the use of stereo cameras to achieve the same results. This approach is shown to work in an indoor environment, giving results that compare favorably with existing systems. The use of stereo instead of structure from motion is shown to be preferable in this environment, while the choice of stereo algorithm proves highly critical to the quality of the results. The use of aggregated voting regions is explored, which is shown to moderately improve the results while speeding up the process considerably. Experiments are also run biasing the randomized input to the classifier generation process, which show further improvements in both performance and execution time. Overall, the approach is shown to be feasible, but not currently practical for robotic navigation in this environment.
# Table of Contents

1. Abstract ................................................................. ii
2. Table of Contents ...................................................... iii
3. List of Tables ........................................................... v
4. List of Figures .......................................................... vi
5. Dedication ................................................................. viii
6. Acknowledgments ....................................................... ix
7. 1 Introduction ............................................................ 1
   1.1 Motivation ............................................................ 1
   1.2 Objectives ............................................................ 2
   1.3 Contributions ........................................................ 3
   1.4 Outline ................................................................. 4
8. 2 Background .............................................................. 5
   2.1 Image Segmentation .................................................. 5
   2.1.1 Definition .......................................................... 5
   2.1.2 Segmentation evaluation ....................................... 6
   2.1.3 Semantic segmentation ........................................ 7
   2.1.4 Motion-based segmentation ................................... 7
   2.2 Point Clouds .......................................................... 8
   2.3 Point Cloud Semantic Segmentation ............................. 8
   2.3.1 Point cloud generation ........................................ 8
   2.3.2 Feature extraction .............................................. 9
   2.3.3 Forests of randomized trees .................................. 10
   2.3.4 Final classifier .................................................. 12
   2.3.5 Results ............................................................ 12
# Table of Contents

3 Implementation and System .................................................. 15
   3.1 System Overview ......................................................... 15
   3.2 Implementation Details ................................................ 15
      3.2.1 Data capture ......................................................... 15
      3.2.2 Point tracking ..................................................... 16
      3.2.3 Feature extraction ................................................ 18
      3.2.4 Tree generation ..................................................... 19
      3.2.5 Image evaluation .................................................... 20
      3.2.6 Comparisons ......................................................... 20
      3.2.7 Extensions ........................................................... 20
   3.3 Training Data ............................................................. 23
      3.3.1 Acquisition ........................................................ 23
      3.3.2 Labeling ............................................................ 24
      3.3.3 Data sets ........................................................... 24

4 Evaluation and Results ...................................................... 29
   4.1 Methodology .............................................................. 29
   4.2 Results ......................................................................... 29
   4.3 Analysis ................................................................. 29
      4.3.1 Tree generation size ................................................. 31
      4.3.2 Performance ......................................................... 34
      4.3.3 Forest size .......................................................... 35
      4.3.4 Generalizability of training data ................................ 36
   4.4 Comparisons .............................................................. 36
      4.4.1 Structure from Motion ............................................. 36
      4.4.2 Alternative stereo algorithms ................................... 39
   4.5 Extensions ............................................................... 43
      4.5.1 Aggregated voting .................................................. 43
      4.5.2 Appearance features ................................................. 48
      4.5.3 Biased window distribution ...................................... 50

5 Conclusion and Future Work .................................................. 53
   5.1 Contributions ............................................................. 53
   5.2 Future Work .............................................................. 53
      5.2.1 Real time operation ................................................ 53
      5.2.2 Output evaluation ................................................... 54
   5.3 Conclusion ............................................................... 55

Bibliography ........................................................................... 56
List of Tables

3.1 Label distributions by data set . . . . . . . . . . . . . . . . . 28
4.1 Confusion matrix of object classes . . . . . . . . . . . . . . . 34
List of Figures

1.1 Idealized view of the segmentation system ............... 3
2.1 Illustration of the properties of a sample ................. 12
2.2 Illustration of a tree with sampling windows and feature thresholds shown for each node .................. 13
2.3 Example of the output from the Brostow et al. classifier. Top row: Input frame. Middle: Ground truth. Bottom: Output. Used with permission. ......................... 14
3.1 Example point cloud generated from stereo disparities. Point size is proportional to proximity to the camera. .......... 17
3.2 Superpixel example ........................................ 21
3.3 Illustration of the properties of a texton sample ........ 22
3.4 The Powerbot platform used for data acquisition ........ 23
3.5 Map of the path followed in the corridor-new data set along the ICICS X wing 2nd floor corridor. Nine of its frames are shown, coupled with the matching hand-labeled ground truth segmentations. Light blue is floor, green is ceiling, dark blue is wall and red is furniture. ......................... 26
3.6 The lab4 data set which was acquired in the ICICS X210 computer vision lab. All eleven hand-labeled segmentations and their source frames are shown. The segmentation color scheme is the same as in Figure 3.5, with the addition of obstacle in salmon. ......................... 27
4.2 Timing data as a function of generation parameter K .... 31
4.3 Node measurements as a function of generation parameter K 33
4.4 Classifier output measurements .......................... 33
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Experimental results as a function of forest size $M$</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison of correctness for cross-trained classifiers on the corridor-new (C) and lab4 (4) data sets</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Structure from motion experimental results</td>
<td>38</td>
</tr>
<tr>
<td>4.8</td>
<td>Number of nodes per feature in the structure from motion classifiers</td>
<td>39</td>
</tr>
<tr>
<td>4.9</td>
<td>Correctness for loopy belief propagation and Gaussian summation mask stereo input</td>
<td>40</td>
</tr>
<tr>
<td>4.10</td>
<td>Example stereo depths for frames 110, 160, 190, 220 and 290 from the corridor-new data set. In both versions lighter shades represent greater proximity to the camera. Left: Input image. Middle: Point Grey Research cross-correlation. White pixels are invalid and have no depth associated with them. Right: Loopy belief propagation. A depth value is given for all pixels.</td>
<td>41</td>
</tr>
<tr>
<td>4.11</td>
<td>Comparison of correctness for cross-trained classifiers on different stereo algorithms: The normal Point Grey Research cross-correlation (N), loop belief propagation (L) and cross-correlation with Gaussian summation mask (J)</td>
<td>43</td>
</tr>
<tr>
<td>4.12</td>
<td>Superpixel classification examples</td>
<td>44</td>
</tr>
<tr>
<td>4.13</td>
<td>Grid classification examples</td>
<td>44</td>
</tr>
<tr>
<td>4.14</td>
<td>Comparison of correctness for superpixel experiments</td>
<td>45</td>
</tr>
<tr>
<td>4.15</td>
<td>Comparison of correctness for grid sizes 20, 40 and 60</td>
<td>46</td>
</tr>
<tr>
<td>4.16</td>
<td>Correctness surface for all grid runs</td>
<td>47</td>
</tr>
<tr>
<td>4.17</td>
<td>Correctness as a function of voting size</td>
<td>48</td>
</tr>
<tr>
<td>4.18</td>
<td>Comparison of correctness with and without the texton extension</td>
<td>49</td>
</tr>
<tr>
<td>4.19</td>
<td>Sampling window attribute distributions</td>
<td>50</td>
</tr>
<tr>
<td>4.20</td>
<td>Biased and normal classifier results</td>
<td>51</td>
</tr>
</tbody>
</table>
Dedication

To Michelle, who will always amaze me.
I would like to extend my thanks to Dr. James J. Little for much patient guidance over the last 18 months. I would also like to thank the occupants of the LCI X wing lab for their help and support, particularly David Meger for putting up with a lot of stupid questions and Walter Wohlkinger for providing some of the stereo results used in my analysis.
Chapter 1

Introduction

1.1 Motivation

As computer vision has been increasingly integrated into mobile robotics, there is growing interest in the use of semantic information for the purpose of navigation. Simultaneous Localization and Mapping (SLAM) algorithms can provide accurate mapping, but they do not provide any deeper understanding of the environment. Direct surface measurements, such as with laser range finders, are able to label which areas are safe for robotic navigation, but they have a much harder time saying where it is appropriate for a robot to drive. Even vision-based mapping approaches can provide the shape of the environment but little else.

This represents a serious problem, as an understanding of how different parts of the environment are intended to be used is required for just about any task more complicated than “go to point X”. Humans do not navigate based on an abstract map of the obstacles in their way. They move through a dense structure of meaning. They know that this one flat surface called a sidewalk is not a good place to sit down and eat their lunch, while this other flat surface, called a lawn, is. Successful robotic operations in uncontrolled environments will have to accomplish similar feats of semantic reasoning.

The most basic way of achieving this is to label scene features with meaningful names based on their properties or potential uses. One of the more promising lines of research which can accomplish this is that of segmentation, or the division of an image into contiguous regions that share some intrinsic property. Semantic segmentations consist of regions such as floor or person. While ultimately derived from image data, these classifications aim to capture a much higher level of scene understanding than what could be directly extracted from pixel brightness levels.

Semantic segmentation has been approached in many ways, but recently it has been partially achieved using only the motion of three-dimensional scene structure derived from laser depth scans \cite{PSN07,PSN08} and structure from motion algorithms \cite{YCWE07,BSFC08}. In contrast to earlier approaches that only used visual appearance cues, this has the advantage of gathering...
training data independent of lighting conditions and visual quirks of the local environment. However, 3D time of flight devices like laser range scanners are expensive and also slow to run a complete scan. Structure from motion only requires a single camera for input, but it consumes significant computational resources to calculate and is only possible for static scenes. While promising, neither of these currently represent practical approaches to the problem.

1.2 Objectives

It is the goal of this thesis to adapt these existing segmentation techniques that make use of the 3D structure of the scene and adapt them to use stereo cameras as a source of data. There are several implications to this seemingly minor change.

Stereo cameras produce point clouds, which differ from the output of range sensors in several important ways. While depths are calculated for every pixel, they can still be sparse because stereo algorithms often fail in regions of the scene with little visual texture. In this regard their output is more like that of a structure from motion process than the regular mesh of a laser scanner. Stereo point clouds can also contain large errors, as the algorithm can easily be confused by boundaries or repeating textures. This is particularly true using faster algorithms. They are a depth estimate instead of a direct measurement and do not allow for a precise reconstruction of scene structure.

The advantages of the original approach of segmenting images based on 3D structure would remain, namely the lack of dependence on lighting conditions and the exact visual appearance of the environment in which it was trained. This simplifies the collection of training data and makes the system more robust when dealing with novel environments.

There also some not inconsiderable practical advantages to this approach. Stereo cameras are considerably cheaper than laser scanning units. They can generate point clouds in much closer to real time than structure from motion systems, which take significant computational resources. Laser depth scanners return entire 3D meshes instantaneously, but the scan itself can take several seconds as a laser needs to raster the entire viewing area. Also, in addition to providing the point clouds, stereo cameras can also be used like any other camera to acquire pictures. This appearance information is likely to be valuable elsewhere in the system.

In order to achieve this goal, an existing system was extended to use point clouds derived from a stereo camera. This new system operates as
1.3 Contributions

follows: First the point clouds are generated from stereo depth estimates and a point-tracking system. Next the structure and motion of these point clouds is abstracted into six types of features. From these features the system builds a forest of randomized trees, an ensemble decision-tree method discussed later. This forest is finally used to classify each pixel in test images to generate a segmented output image. (See Figure 1.1.)

1.3 Contributions

In this thesis we show that semantic segmentation is possible using only features derived from the structure and motion of stereo camera point clouds. The output from this system will be tested against a hand-labeled ground truth using data sets acquired for this purpose. These results will be com-
pared to those generated using structure from motion point clouds in the original system.

In order to further examine the nature of this system, a series of alternatives will be analyzed. Here we will look at the contributions of the stereo algorithm, and the classifier being used. Several extensions aimed at better performance and speed of evaluation will also be explored. These extensions consist of the geometric aggregation of classifier results, the inclusion of appearance-based features and the biasing of sampling windows to improve the randomized tree generation process. The system will also be run against a data set completely different from the one it was trained on, to test its generalizability.

1.4 Outline

This thesis is composed of the current introduction plus four other chapters. Chapter 2 will detail the background and related work of this project. This includes a thorough description of the workings of the original system upon which this work is based. Chapter 3 describes the implementation details of the system. The acquisition and labeling of training data will also be covered. The system performance will be analyzed in Chapter 4, along with the contribution of system components. The various proposed extensions will also be tested. Finally, in Chapter 5, the thesis will be concluded and possible avenues of future research will be discussed.
Chapter 2

Background

As the goal of this thesis is to describe a semantic segmentation which is generated using point clouds, the relevant background needs to cover all of these components. First a summary of image segmentation will be given, with particular focus on the existing semantic segmentation work and those based on motion cues. The problem of how to evaluate the output of segmentation systems will also be covered, as that is a recurring issue with this system. Next the use of point clouds in segmentation and robotic navigation will be covered. Finally we will look at a system which uses the structure and motion of point clouds for semantic segmentation. It is this work upon which the majority of this thesis is based, so it will be covered in depth.

2.1 Image Segmentation

2.1.1 Definition

Image segmentation is one of the older subjects in computer vision, with the first attempts occurring over 40 years ago [MA68]. It can be defined as the process of partitioning the pixels of an image $I$ into a set of $n$ sub-regions $R_i$ such that

\[ \bigcup_{i=1}^{n} R_i = I \]  

(2.1)

and

\[ R_i \cap R_j = \emptyset, i \neq j \]  

(2.2)

The pixels within each region should share some property such as illumination levels or object label in the underlying image. Since the problem was first addressed there have been a wide number of approaches. These have been divided into clustering, region growing, and boundary detection.
2.1. Image Segmentation

[FM81]. Until recently most attempts have focused entirely on local image properties, looking to exploit the natural mathematical properties of contiguous regions [HS85, FMR+02].

2.1.2 Segmentation evaluation

Before a segmentation is produced, the question of how to evaluate its quality must be addressed. This has become an increasingly important issue with the proliferation of sources of hand-labeled ground truth data sets such as the Berkeley Segmentation Dataset and Benchmark [DB], ImageNet [DDS+09, Imaa] and ImageParsing.com [Imab]. Some of these only provide unlabeled segmentations, which makes the evaluation question even harder.

There is a wide range of algorithms for producing quantified results, but the correct solution is not obvious [Zha96, Zha01]. Evaluation systems can be broken into two categories: those which look only at the generic image properties of a proposed segmentation and those which make a comparison against a hand-labeled ground truth. As we are interested in semantically-rigorous segmentations, the latter is of the most interest here.

The most direct way to measure the segmentation quality is a simple count of how many pixels in the image were classified correctly when compared with a hand-labeled ground truth, expressed as a percentage. This is easy to calculate, but as Yasnoff et al. showed in [YMB77] it does not always match human perception. To account for this they proposed a pixel distance method that weights invalid pixels by how far away they are from the nearest region of the given label. This compared favorably to human perception for some image classes, but only increased the problem on others.

Levine et al. [LN85] defined a suite of evaluation tools for analyzing different segmentation techniques. This was done without reference to a labeled ground truth, instead focusing on generic segment properties such as pixel brightness values and textural uniformity within a region, contrast between regions, contrast across lines, and line connectivity.

Sezgin [SS04] used a combination of several measures. This included the percentage of correct pixels as well as edge map comparisons and generic measurements of region uniformity. These were average in an ad hoc manner to generate a composite evaluation.

Zhang et al. [ZFG04] proposed an entropy-based evaluation that looks at the uniformity of the regions being proposed as well as the complexity of the segmentation. Jiang et al. [JMIB06] used techniques inspired by cluster distance measurements in machine learning.

Crevier [Cre08] approached the problem for unlabeled segmentations be-
2.1. Image Segmentation

ing compared against *multiple* hand-labeled ground truth images. This is a slightly different problem, as it gets into the intricacies of multi-scale segmentations that are of less concern in our domain space. The lack of labels also complicates things, which is not a problem for us either.

A common theme throughout all of these proposed evaluation techniques is their lack of semantic information. As will be seen later, this can be problematic when trying to assess the quality of segmentations, particularly in the context of trying to solve a specific problem. Not all errors are equal, because not all the semantic categories being used will be equally disparate from each other. Likewise, mistakes concerning some classes will be far more serious than others.

2.1.3 Semantic segmentation

It has been recognized for some time that segmentation is an inherently semantic process that must be based on more than just local pixel values [PP93]. More recent approaches to the problem have attempted to take this into account. Al-Haj et al. [AHAA08] attempts this by first segmenting using a clustering algorithm in the HSV color space and then organizing these segments into a hierarchy. Most new attempts are based on machine learning concepts to let the statistical relation between abstract features and semantic content be discovered automatically. Shotton et al. [SJC08] used simple “texton” features and a forest of randomized trees trained on hand-labeled ground truth to generate semantic segmentations. This approach is very similar to the one used in this thesis, and texton features will be one of the components tested later.

2.1.4 Motion-based segmentation

In parallel to the exploration of semantic segmentation systems, there have been several attempts to incorporate motion cues into the processes of segmentation and detection. This is often of interest because of the additional information that can be harnessed if the differences between frames of video are examined instead of treating each frame independently.

Viola, Jones and Snow [VJS05] described a pedestrian detector that used simple features spanning pixels in both the current and adjacent frames. Kang et al. [KCY05] used the parallax of a moving camera to segment and track moving objects. Yin et al. [YCWE07] achieve foreground/background video segmentation using “moton” features, which are clusters of pixel spatial and temporal gradients.
2.2 Point Clouds

Most of the work using dense point clouds relates to airborne laser scanning and comes from the remote sensing literature. The problem of segmenting point clouds has been investigated, but usually with the goal of creating terrain maps or discrete three-dimensional models of objects \cite{WKWL02, SV03, VGSR04}.

Recently there has been interest in ground-based laser scanning systems found in robotics or autonomous vehicles. These are focused primarily on navigation using a SLAM type system and the labeling of obstacles \cite{BWW03}. The DARPA Grand Challenge and Urban Challenge were important motivating forces for this research, as vision systems are still not up to the task and cost was not a limiting factor in that domain. Heavy use of multiple 3D laser scanners was a standard feature for the winners \cite{TMD06, UAB08}.

More relevant to this thesis, there has also been work done on the semantic segmentation of images using the point clouds. Posner et al. \cite{PSN07} uses the local geometry of laser scan point cloud as well as simple image features such as color and texture to train an support vector machine (SVM) for semantic segmentation. This showed promise, but lacked the ability to take the global arrangement of objects into account during classification.

2.3 Point Cloud Semantic Segmentation

Bringing all these trends together, Brostow et al. \cite{BSFC08} describe a method for segmenting images into labeled, semantically significant regions. This system is based purely on the structure and motion of point clouds as derived from structure from motion methods. The classifier used is a forest of randomized trees generated using six simple features. The final segmentation is achieved by running this classifier on every pixel in the test image.

The work in this thesis is primarily based on the Brostow et al. system. It will now be explained in detail, as many of the later extensions and improvements require detailed knowledge of its workings.

2.3.1 Point cloud generation

The point clouds used in the original system are generated using a structure from ego-motion process. Harris-Stephens corners \cite{HS88} are used to define 20x20 pixel patches. Matching is performed on these patches between frames
with a simple cross-correlation thresholded at 0.97. A commercial camera tracking product from 2d3 [Tra] is used to estimate the camera pose for each frame. This gives a complete 3D trajectory for each tracked point as well as the camera itself.

### 2.3.2 Feature extraction

Using the point clouds generated from the video sequence, features that describe the structure and motion of these points are extracted. Later, these abstract features will drive the classification process. All these features are derived only from the point cloud and its motion over time, with no reference to the image-level appearance of those points.

- **$f_H$** Height of the point above the ground. This assumes the vehicle or robot is moving across a regular, level surface. The calculation of this feature depends on a configurable value set for each data collection that gives the measured height of the recording camera above the ground plane. This allows training data gathered from multiple platforms to be used. This feature is strongly predictive of several semantic categories such as floor and ceiling.

- **$f_C$** Closest approach of the point to the path of the camera. This feature is useful for distinguishing classes that are arranged around the camera path in a regular pattern, like sidewalks and buildings along a road.

- **$f_{O_x}, f_{O_y}$** Local point cloud orientation. A Delaunay triangulation is performed on the 2D projection of the points for each frame. The X and Y orientation of the resulting facets are used for this feature. This is calculated for every pixel in the image instead of just per tracked feature point.

- **$f_D$** The track density, or the number of tracked image features in a region. This is an indirect measurement of the visual texture of the classes.

- **$f_R$** The 2D variance of the back-projection for each point. This feature looks at the difference between where a tracked point is predicted to be based on where it was and how the camera moved, and where it actually is found. Classes that violate the static-world assumption (such as people, animals and vehicles) will have a much higher back-projection error than others.
These features are defined on a grid of the same size as the original input images. However, with the exception of the orientation features, they are all only set at the location of the tracked points. This results in very sparse feature tables and makes the initial density of tracked points critical to the success of the algorithm.

During classifier generation the features are always referenced as the sum of their values across a given sample window. To improve computational efficiency all feature data is stored in the form of a summed-area table, also known as an integral image. In this form, every element of the array is the sum of all values above and to the left of that point. (That is, all the values with a row or column index less than or equal to the current index.) This provides a very fast and efficient way to compute the sum of values within any rectangular subset of the array. After the tables have been generated, this can be done in only four arithmetic operations independent of the size and shape of the subset being summed [Cro84, VJ02].

### 2.3.3 Forests of randomized trees

The classifier used in this system is a forest of extremely randomized trees. Like all binary-tree classifiers, an extremely randomized tree is built out of nodes that apply a simple binary test to the data, usually a value threshold. This test determines if the evaluation will progress to the left or right child of the current node. This process is performed recursively until a node with no children is reached. This is called a leaf node, and it contains the final output for the process, either as an absolute classification or a distribution of possible classes.

Traditional methods for building decision trees looked to optimize a metric such as information gain at each node in the process. This is defined as the difference between the entropy of all the data \( H(X) \) and the conditional entropy given this split \( H(Y|X) \):

\[
IG(Y|X) = H(Y) - H(Y|X) \tag{2.3}
\]

where

\[
H(Y) = -\sum_j p_j \log_2 p_j \tag{2.4}
\]

and

\[
H(Y|X) = \sum_{val} P(X = val)H(Y|X = val) \tag{2.5}
\]
2.3. Point Cloud Semantic Segmentation

This total reduction in entropy is the gain in information which results from splitting the data at this point. By maximizing this at every step, the classification power of the tree is theoretically maximized as well. However, building trees based on information gain or a similar measure has two major problems: it can be very computationally expensive \cite{HR76} and it tends to over-fit the data \cite{Bra07}.

One alternative that has recently become very popular is the use of randomized trees as described by Guerts et al. \cite{GEW06}. The key insight is that instead of trying to generate one very good tree at high cost, it can be better to quickly generate several low quality trees and let them vote on the final answer. During tree generation a random break-point is chosen at each node instead of looking for the optimal one. This is, of course, very fast to compute. The resulting tree is of low quality, but both generation and evaluation are cheap enough to generate an entire ensemble of them, also known as a forest. This reduces over-fitting as well as variance in the output. It is also trivially parallelizable, both in generation and evaluation, an increasingly important attribute.

To generate a tree, first a set of $K_s$ samples are randomly chosen. These define a window from which feature values will be sampled. These windows are defined by their width, height, X offset and Y offset (Figure \ref{fig:window}). The offset values are limited to no more than $\frac{1}{16}$ of the image size while the window dimensions are limited to $\frac{1}{8}$. The offset is measured from whichever pixel is being evaluated at the time. Each sampling window can be applied to any pixel for any feature type, resulting in the numeric value used later for classification. The value returned is the sum of values for the given feature type within the sampling window. By making use of the pre-generated summed-area table for the appropriate feature type, this sum can quickly be calculated.

For each sample, a series of $K_t$ tests are chosen. Tests are a selection of a random training frame and a random pixel location within that frame. This frame can be any frame for which we have labeled ground truth. (In practice, this is available for every tenth frame.) The pixel location is unconstrained within the image.

At each of these test locations, the sample is applied for every feature type. This gives a numeric value for the sample/test/feature combination. Since we limit the test location to labeled frames, we also get the correct classification for that pixel as determined in the ground truth. The complete vector generated is $v_i = \{\text{sample, test, feature type, feature value, pixel label}\}$. The complete collection of vectors is $C$, with $|C| = K_s \cdot K_t \cdot \text{num features}$. These form the corpus used in the recursive tree generation process.
2.3. Point Cloud Semantic Segmentation

![Illustration of a sample](image)

Figure 2.1: Illustration of the properties of a sample

### 2.3.4 Final classifier

Once the forest of randomized trees is created, it can be used for segmenting test images. First the test images must be put through the same feature extraction process described above to generate the summed-area tables of feature values. To perform the classification itself, each pixel in the image is run through each tree in the forest. Every node in the tree contains a sampling window and a feature threshold. (See Figure 2.2.) The sampling window for the node is applied to the current pixel using the feature type specified for the node. This results in a numeric value that is compared to the threshold of the node. If it is below the threshold, evaluation moves to the left child of the current node, otherwise it progresses to the right.

This process continues until a leaf node is reached. It will contain a discrete distribution of possible classifications for the current pixel. After repeating this process for every tree in the forest we have a collection of $K$ distributions. These are combined and the maximum a posteriori (MAP) estimate is taken. This becomes the classification for the current pixel and it is colored appropriately in the output image. The process repeats until all the pixels have been classified.

### 2.3.5 Results

The Brostow et al. system was trained and tested on sequences taken from 86 minutes of video that was acquired from an automobile driving through an
urban area. The classes used were: building, tree, sky, car, sign-symbol, road, pedestrian, fence, column-pole, sidewalk and bicyclist. This is very different from the indoor environment and simplified class list tested with the system presented here, but it can still serve as a useful comparison.

An example of the output from this system is shown in Figure 2.3. As can be seen, it does a reasonable job handling the larger segmentations which define the scene, but many of the smaller details are missing or incorrect. Quantitatively, reported as a percentage of pixels labeled correctly, they achieved an average score of 61.8%. This is quite good given the challenging domain and class list they used, but it leaves plenty of room for improvement.
2.3. Point Cloud Semantic Segmentation

Figure 2.3: Example of the output from the Brostow et al. classifier. Top row: Input frame. Middle: Ground truth. Bottom: Output. Used with permission.
Chapter 3

Implementation and System

3.1 System Overview

Processing begins with capturing matched frames of video input from a stereo camera. Image points are extracted and tracked through multiple frames. The real world depth for each tracked point is determined using the stereo depth estimate, giving us a full three-dimensional point cloud for each frame where each point within the cloud represents one slice of a four-dimensional track. Six different features based on the structure and motion of these points are then abstracted from these point clouds. In the training phase, these feature values are used along with hand-labeled ground truth frames to generate a forest of randomized trees. In the evaluation phase, the trees in an existing forest are used to classify each pixel in turn.

3.2 Implementation Details

The system is composed of the following components: data capture, point tracking, feature extraction, tree generation and finally image evaluation. The first two steps were written in C++ in order to interface with existing libraries, while the rest was implemented in Python. The original Brostow et al. system was used as a reference, but all aspects were implemented from scratch.

3.2.1 Data capture

Video frames are captured from a Point Grey Bumblebee2 stereo camera over Firewire using the Point Grey pgrlibdcstereo library [Inc]. In order to assure a regular capture frequency, this is limited to about 12 frames per second in the current implementation. Frames are captured at a resolution of 1024x768 and all subsequent processing happens at the same scale.
3.2. Implementation Details

3.2.2 Point tracking

Our stereo point clouds are extracted using the Pascal Fua “back and forth” algorithm [Fua93] with the addition of surface validation [ML00] as implemented in the Point Grey Triclops library. This is not the highest quality stereo algorithm, but it is fast and widely available. As long as there is sufficient visual texture for which the cross-correlation to match it generates reliable output. As can be seen in Figure 3.1, the resulting point cloud gives reasonable results along the wall and ceiling, where there is enough visual texture to be matched from the left image to the right. The highly specular floor, however, gives much less sensible results. This is a critical limitation in our input data.

Feature tracking is performed using Speeded Up Robust Features (SURF) [BETVG08]. These are an improved version of David Lowe’s SIFT features [Low04]. SURF makes extensive use of summed-area tables to achieve a significant improvement in speed with equivalent performance. It also allows a wide degree of customization where additional improvements in speed can be gained by disabling aspects of the image descriptions such as orientation invariance. In our case, the blob response threshold is reduced in order to ensure a larger pool of points from which to work.

After they are extracted, points are tracked between frames using the SURF library [surf], which implements a basic nearest neighbor search. To help prevent false positives, this search is performed in both directions in the same manner as the Fua “back and forth” algorithm used for stereo matching. Only the matches between SURF points that are symmetrical are accepted and become part of the point cloud.

Because the three-dimensional information is critical to our process, image features that fall in a section of the image without a validated stereo disparity are dropped. This eliminates around 30% of the points, depending on the natural texture of the scene.

In addition, points must also be tracked long enough to pass through a labeled ground truth frame. This is required so that a correct segmentation is known for each point. Another ≈ 30% of the remaining points are eliminated because they fail to pass through a labeled frame, and ≈ 30% because they are only found in a single frame, thus not providing any useful motion information. A final check makes sure that, if a point passes through multiple labeled frames, the given labels must all be consistent. Demonstrating the stability of the SURF points being tracked, this check fails at negligible frequencies (< 0.1%). Despite all the points that are eliminated, frames still have an average of 150 valid points at the end of this process. It would
3.2. Implementation Details

Figure 3.1: Example point cloud generated from stereo disparities. Point size is proportional to proximity to the camera.
be possible to boost this considerably if desired by hand-labeling a larger percentage of the training data.

The final step of point tracking is to calculate the motion estimation between frames. This is performed using [AHBS7]. This approach was chosen for the simplicity and computational efficiency of the implementation. It uses a singular value decomposition of a 3x3 matrix to find the rotational matrix $R$ and translational vector $T$ that minimize

$$\Sigma^2 = \sum_{i=1}^{N} \|p_i' - (Rp_i + T)\|^2$$

(3.1)

where $p$ and $p'$ are the sets of three-dimensional points between frames.

### 3.2.3 Feature extraction

With the image points tracked through the frames of the video, each labeled with a classification, and the motion estimation being made between each frame, we now have all the information needed to extract the features used by the classifier.

$f_H$ The height above the ground plane is calculated directly from the world-coordinate $Y$ value for each point.

$f_C$ The closest approach to the camera path is found in the minimum offset in camera coordinates for each point along its entire path. In addition to the frames it appears in, the next 100 frames of camera motion are applied to the last known position of the point. This is done because the closest approach of a point is almost always after it is no longer visible in the camera’s frame of view.

$f_{O_x}, f_{O_y}$ The orientation features are generated using the Delny Python package [pac], which is a wrapper for the C library libqhull [lib]. The triangulation is run on the $X$ and $Y$ coordinates for every tracked point, returning a triangular mesh spanning all of them. For each triangle in this set the unit normal vector is calculated from vertices $v_1$, $v_2$ and $v_3$ as

$$\tilde{n} = \frac{(v_3 - t_1) \times (v_2 - v_1)}{|(v_3 - v_1) \times (v_2 - v_1)|}$$

(3.2)

and the orientations in $X$ and $Y$ are simply the dot product of $\tilde{n}$ with unit vectors in those dimensions. Unlike the other features, these
values apply to all the pixels within the triangle and not just at the position of the tracked points.

\( f_D \) The density feature is a simple count of how many tracked points there are in a region. If a pixel coordinate is a tracked point, the feature table gets the value 1, otherwise it gets a 0.

\( f_R \) The back-projection residual compares the world coordinates for a tracked point from the previous frame against the calculated position using its position in the current frame and the inverse motion estimation. The feature value is the Cartesian distance between the observed and estimated locations.

These are accumulated in the form of summed-area tables and saved to disk for later use during tree generation and frame evaluation.

### 3.2.4 Tree generation

Tree generation follows the procedure outlined in [GEW06]. First a randomized sampling of the training data is taken, parametrized as \( K_s \) random sampling windows each used to generate \( K_t \) tests. (In the experiments below these are always set equal to each other for a single \( K \) parameter.) For each sample/feature combination in this set, a random value is chosen within its range as a threshold. These are evaluated in terms of a metric such as information gain. The system being described here follows the Geurts paper and uses the Wehenkel and Pavella normalization [WP91]. The sample/feature pair which maximizes the metric is selected. This sets the sampling window, feature type and threshold for this node.

The pool of training vectors is split in two based on this decision point, and the process is applied recursively on both. This is repeated until one of the following conditions is met:

- All remaining training vectors have the same classification. This is the ideal case, resulting in a perfect classification output at this leaf node.

- The feature values for all the remaining training vectors are all identical. This makes further splitting impossible.

- The number of remaining training vectors falls below \( n_{\text{min}} \), a configurable value that is 5 by default.

Because it is being done for each tree in the forest, this process parallelizes easily.
3.2.5 Image evaluation

In order to evaluate a test image, the summed-area tables of feature values must first be generated using the same process as outlined above. With this done, it is a matter of walking through the image, pixel by pixel, and evaluating every tree in the forest at that point. Each tree returns a probability distribution over the possible segment labels. These are combined and the maximum is chosen as the label for this pixel.

As with tree generation, this process is trivially parallelizable.

3.2.6 Comparisons

Structure from motion

For comparison with the original Brostow paper, an alternative feature extraction process was implemented, based around point clouds generated from structure from motion. This is accomplished using the Bundler library [SSS06, SSS08]. This only changes the early part of the system, providing both point cloud motion and inter-frame motion estimates.

Alternative stereo algorithms

The default system uses a fairly basic cross-correlation stereo processing library to build its point clouds. This has the advantage of requiring only very moderate computing hardware to run in near real time. There are other stereo algorithms available, however, most of which score higher on evaluations. With the advent of affordable, highly parallelized GPU systems even the more advanced of these other algorithms are becoming practical to run on systems like the one in this thesis. In order to test the dependence on stereo results, the same corridor-new training data had stereo disparities calculated using a loopy belief propagation algorithm [SZS03, FH06] on an NVIDIA GeForce GTX 295 GPU. This took roughly 13 seconds per frame to compute. An enhanced cross-correlation algorithm was also tested, which uses a Gaussian summation mask instead of a box filter.

3.2.7 Extensions

Aggregation voting

Another extension that was explored is the aggregation of classifier results over larger regions to both improve performance and reduce variance in the
3.2. Implementation Details

classifier output. The types of regions tested are superpixels (Figure 3.2) and a regular grid.

Superpixels are an oversegmentation that give a set of internally similar segmentations that respect natural boundaries in the image RM03. Superpixels were computed using the package provided by Greg Mori Mor, MREM04, Mor05. Settings were changed to generate output with both large and small segments.

![Superpixel example](image)

(a) Source image  (b) Superpixels (small)  (c) Superpixels (large)

Figure 3.2: Superpixel example

In this mode of operation, input images are first partitioned into regions. During classification, each region is evaluated as a single unit and assigned the same class. This is done by evaluating a small number of randomly selected pixels within each region. These vote on the classification for the entire region. It is hoped that this will result in a much smoother and more semantically relevant output. It will also have the advantage of dramatically reducing the amount of pixel evaluations performed, though the superpixel version incurs the extra cost of the segmentation step.

Appearance Features

The base system works entirely from point cloud structure and motion, completely ignoring all appearance information in the original images. (Except for indirectly through the $f_D$ feature.) In order to test the potential value of using appearance as well as structure, an extension is explored that adds texton features to the classifier.

In this context, textons are localized image features based on pixel color differences. They have been demonstrated to be useful in segmentation algorithms SWRC06, SWRC09, SJC08. Textons are used in a very similar way to the image features already used in this system, and as such present an excellent way to test the inclusion of image features.
3.2. Implementation Details

Instead of looking at features abstracted from point cloud movement, textons deal directly with pixel values. They are based on sampling windows much the same as the ones already described, with the distinction that only a single sample pixel is referenced instead of an entire window. The X and Y offsets are the same, but the width and height fields are always set to 1 as shown in Figure 3.3. In the version implemented here the textons only deal with greyscale pixel brightness values instead of individual color channels.

![Target pixel](image)

![Sample pixel](image)

Figure 3.3: Illustration of the properties of a texton sample

For target pixel $p_t$ and sample pixel $p_s$ the four texton features are tested in this system are:

- $f_P$ Sample pixel value: $p_s$
- $f_{P_s}$ Sum of the target and sample pixel values: $p_t + p_s$
- $f_{P_d}$ Difference of the target and sample pixel values: $p_t - p_s$
- $f_{P_{da}}$ Absolute value of the difference of the target and sample pixel values: $|p_t - p_s|$

### Biased Window Distribution

One inefficient aspect of randomized tree generation is that many more sampling windows are generated than end up being used. This is required to provide a wide range for the tree generation process to choose from, but if it was possible to ensure the pool of samples is more relevant, that could be a gain in both accuracy and computational efficiency. By default the sampling window properties are randomly chosen from a uniform distribution. This could be trivially biased to match any systemic preferences shown in the tree generation process.
To test this hypothesis, the distribution of the properties of sampling window selected for inclusion in trees will be examined, and a version of the classifier that biases its training pool to mirror this distribution will be run.

3.3 Training Data

3.3.1 Acquisition

Training data was acquired using a Mobile Robotics Inc. Powerbot mobile robot base and a Point Grey Research Bumblebee2 stereo camera. (See Figure 3.4.) This platform was manually driven through a variety of common office environments while capturing the stereo video frames. In order to keep the frame rate both higher and more regular, only the raw left and right frames were captured during acquisition. Later the Point Grey Research Triclops library was used to extract rectified reference frames as well as stereo disparity maps.

Figure 3.4: The Powerbot platform used for data acquisition
3.3. Training Data

3.3.2 Labeling

All training data was hand-labeled using InteractLabeler 1.2.1 [Int], a toolkit written by Gabriel Brostow and Julien Fauqueur. This process was performed by a single annotator, the author. Frames were manually segmented and labeled using the set of classes listed below. Examples of the resulting ground truth can be seen in Figures [3.5] and [3.6]. Labeling took several minutes per frame, depending on the scene complexity and the resolution of the image. This was performed for every tenth frame in order to balance the need for adequate coverage for training purposes with the amount of effort needed to complete the process.

The classes used are:

- **floor**: Any horizontal surface upon which it would be safe for the robot to drive.
- **wall**: Any vertical surface stretching from floor to ceiling.
- **ceiling**: Any structure or material stretching between walls, including pipes and other utilities.
- **furniture**: Any freestanding object that could be moved, such as an office chair, sofa, desk or bookshelf. For mapping purposes, knowing that an object might not be in the same position later could be valuable.
- **obstacle**: Any non-movable object that projects from the floor, such as a guard rail or other partition. This includes horizontal surfaces like stairs upon which it would not be safe to drive.
- **person**: Any autonomous entity that moves relative to the fixed background. This could be a person, animal, other robot or a vehicle. For mapping purposes, it would be assumed that this object would not be in the same location at later times.

These classes were chosen for their simplicity and unambiguity in an indoor environment. The demands of robotic navigation were also a primary consideration, with the goal of providing a basic go/no-go overlay for routing purposes. As a tertiary goal, the possible future development of robotic mapping using this platform was considered.

3.3.3 Data sets

The following datasets were used in the development and testing of this system:
3.3. Training Data

**lab-glide** 100 frames taken in the ICICS X209 robotics lab. This was an early data set and was captured at a lower resolution than the others. It is annotated with different set of labels and served primarily as the standard training sequence during development.

**corridor-new** 862 frames moving along the 2nd floor corridor in the X wing of the ICICS building. This is the data set used for the majority of the following experiments. As is seen in Figure 3.5, this is a challenging environment with little visual texture and many reflective surfaces. Earlier versions of this data set had severe point tracking failures due to motion blur during the turns. The final version avoided that problem by taking each turn in small segments, between which the robot was allowed to come to a complete rest. All the intermediate frames with excessive motion blur were then deleted, leaving just the 862 used here. This worked for the point tracker used in this system, but as described later the structure from motion processing still had problems with the turns.

**lab4** 102 frames taken in the ICICS X210 computer vision lab. This serves as a comparison for corridor-new. Sample frames and segmentations are shown in Figure 3.6. This was a reasonably cluttered environment with more visual texture than before. Here the robot was driven in a straight line to avoid the motion blur problems described above.

The corridor-new and lab4 data sets were captured at a resolution of 1024x768. Most of the experiments were performed on down-sampled 320x240 versions to increase execution speed. An increase in output correctness was also observed running at the lower resolution.

The breakdown of the data sets used for testing into their classes is given in Table 3.1. They are mostly very similar with the exception of the wall/furniture ratio. The corridor-new set contains very little furniture, only two recycling bins and a fire extinguisher, while the lab4 set naturally has much more.

As is also shown, neither of the data sets used for testing in this thesis contain any instances of the **person** class. This is primarily because they were captured on a weekend, but earlier acquisition attempts had shown that people do not behave normally when confronted with a large robot driving down the corridor. This demonstrates a possible sociological problem with using motion for classification – the behavior of people around the robot is likely to be very different before they have become accustomed to it, changing their profile in the classifier. Training data would need to be...
3.3. Training Data

Figure 3.5: Map of the path followed in the corridor-new data set along the ICICS X wing 2\textsuperscript{nd} floor corridor. Nine of its frames are shown, coupled with the matching hand-labeled ground truth segmentations. Light blue is floor, green is ceiling, dark blue is wall and red is furniture.
3.3. Training Data

Data set:
lab4

Frame #000

Figure 3.6: The lab4 data set which was acquired in the ICICS X210 computer vision lab. All eleven hand-labeled segmentations and their source frames are shown. The segmentation color scheme is the same as in Figure 3.5 with the addition of obstacle in salmon.
3.3. Training Data

<table>
<thead>
<tr>
<th></th>
<th>corridor-new</th>
<th>lab4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceiling</td>
<td>11.4%</td>
<td>16.1%</td>
</tr>
<tr>
<td>Barrier</td>
<td>0%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Floor</td>
<td>18.5%</td>
<td>22.5%</td>
</tr>
<tr>
<td>Wall</td>
<td>67.8%</td>
<td>33.4%</td>
</tr>
<tr>
<td>Furniture</td>
<td>2.3%</td>
<td>27.6%</td>
</tr>
<tr>
<td>Person</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 3.1: Label distributions by data set

carefully collected and possibly updated at regular intervals, or the system would have to use online training in order to avoid this problem.
Chapter 4

Evaluation and Results

4.1 Methodology

All experiments use classifiers trained on the first 101 frames in the chosen data set, usually corridor-new. This value was chosen because preliminary tested showed that longer sequences of training frames only drive up the amount of training time needed to reach a given level of output quality. The classifiers were then tested against 20 frames from the same data set but outside the sequence of frames that were used to train them. They are also tested on sequences from outside their own data set for comparison. The output generated by the system was compared against the corresponding hand-labeled ground truth segmentations. The percentage of correctly-labeled pixels, termed correctness, was returned. All performance numbers shown here are averaged across these 20 frames.

4.2 Results

Output for the system was good ($\approx 81\%$) as measured quantitatively by the correctness metric. However, the output is distinctly lacking in a qualitative sense. The segmentations are very rough when compared to the hand-labeled ground truth. As could be expected for a system where each pixel is evaluated independently, the regions have very uneven boundaries and show a lot of noise and variance. A sample of system output is shown in Figure 4.1.

4.3 Analysis

In this section the classifier output will be explored in depth, with the goal of understanding its basic behavior and finding a default set of parameters for use in the rest of the experiments.
4.3. Analysis

Figure 4.1: Example output for frames 110, 160, 190, 220 and 290 from the corridor-new data set. Left column: Input image. Middle: Hand-labeled ground truth. Right: Output segmentation for $K = 400$.  

30
4.3. Analysis

4.3.1 Tree generation size

The most important parameters during tree generation are $K_s$ and $K_t$, the number of samples and tests used. The higher $K_s$ is, the greater the number of different samples the trees have to choose from, and the higher $K_t$ is, the more examples there will be of how the samples function in the training data. Neither $K_s$ nor $K_t$ is valuable without the other. A large number of samples is useless without tests to demonstrate the statistical properties of each sample, and a large number of tests is useless if the number of samples from which to choose is too low.

To simplify analysis the $K_s$ and $K_t$ values will be kept equal for all experiments. For this reason we will only refer to $K$ from this point. It is important to remember that the total number of training vectors used in the tree generation process rises as the square of $K$, because for every one of the $K_s$ sampling windows there are $K_t$ tests applied at random locations.

In order to explore the exact influence of $K$ an experiment was run wherein it was raised from 50 to 900 in increments of 25. For each value a complete forest of classifier trees was generated and the 20 test frames were evaluated. This is the standard evaluation framework that most of the later experiments will follow as well.

Classifier attributes

The results for classifier execution time were not surprising. Tree generation time increases in a clean quadratic curve (Figure 4.2a). This is to be expected as the total number of vectors being processed in the recursive tree generation process is going up by the square of $K$.

(a) Tree generation times in seconds   (b) Frame evaluation times in seconds

Figure 4.2: Timing data as a function of generation parameter $K$
4.3 Analysis

The frame evaluation times (Figure 4.2b) show some more interesting results. The evaluation of binary trees is in theory a log \( n \) operation, where \( n \) is the number of nodes, but that assumes a perfectly balanced tree. The randomized tree generation process does not guarantee balanced trees as output. This figure plainly shows that the trees are not balanced. Many pixels being classified by the trees built with a larger \( K \) have a short path to a leaf node, resulting in the lowered frame evaluation times. This is fundamentally a sign of natural optimization in the tree generation process. Given enough samples it can find a very quick and efficient series of tests with which to classify most input.

The average size of the trees being generated shows a curve that appears to be in the process of converging (Figure 4.3a). This also implies a maximum efficiency that is being approximated by the randomized tree generation process. At some point adding more samples during tree generation will not result in larger or better trees. For each node in the tree, we can assume that there exists an optimal sample which could be used if it was available. Once the pool of samples from which to choose is large enough, there is a high confidence that this sample (or one very similar to it) will be included. After that point it is redundant to add more samples, as that will just slow down the process without improving the results.

One thing the \( K \) parameter does not seem to affect is the relative proportion of features chosen during tree generation. As shown in Figure 4.3b, these remain reasonably constant across the entire testing range. As would be expected, the orientation features \( f_{Ox} \) and \( f_{Oy} \) are highly salient. The popularity of the back-projection residual \( f_R \) is surprising, given the static-world nature of the testing environment. It is possible that, if \( f_R \) is largely constant as it is here, it ends up being used to convey the same information as the track density feature. This would also explain the low salience of \( f_D \).

Output quality

When we look at pixel correctness results, the trends are not so clear. In the direct pixel correctness values (Figure 4.4a) there is a lot of noise. Overall the influence of the amount of training input seems to be minimal. There appears to be a maximum around \( K = 400 \), but that could be an illusion of this noisy data. There is not much here to help us optimize the value of \( K \).

As all of these performance values are averaged over the 20 test frames, it is important to remember that a great deal of output instability could be being smoothed over. The variance of the correctness is shown in Figure 4.4b. While the raw correctness values do not show a great deal of change
4.3 Analysis

Figure 4.3: Node measurements as a function of generation parameter $K$

across the different values of $K$, here we can see its influence much more plainly. Classifiers built with fewer training examples show a much higher variance, which intuitively makes sense. Again we see a change in behavior around $K = 400$ as with the frame evaluation times.

Given these results, 400 was chosen as the optimal value for $K$. This closely matches the predictions made in the original Guerts et al. paper [GEW06]. There they found a large category of classification problems which exhibited a similar response to what was seen in Figure 4.4a. The optimal value of $K$ found for those problems was $\sqrt{n}$ where $n$ is number of attributes. In the case of the system being tested here, that is the total number of
4.3. Analysis

different sampling windows possible. We can define

\[ n = \frac{\text{image width}}{8} \cdot \frac{\text{image height}}{8} \cdot \frac{\text{image width}}{16} \cdot \frac{\text{image height}}{16} \] (4.1)

As we are working with 320x240 images, this works out to be \( n = 360000 \) and \( \sqrt{n} = 600 \). Given that the distinctions between neighboring attributes (those with small difference in window size or offset) is likely to be minimal in our system, 400 seems very reasonable.

Even though the optimal value of \( K \) has been chosen, most of the following experiments will still be run on the full 50-900 range. This is done to provide the most complete comparison and to learn more about the nature of this classifier.

4.3.2 Performance

With a value for \( K \) selected, we can focus more attention on the exact nature of the output being generated. The confusion matrix given in Table 4.1 breaks down the correctness results by object class. Both floor and ceiling are often confused for wall, but pixels that are labeled wall have a high confidence of being correct. Note that there were very few furniture objects in this data set, so those results are not statistically significant. Overall the system scores a 81.8% global accuracy.

<table>
<thead>
<tr>
<th></th>
<th>Floor</th>
<th>Ceiling</th>
<th>Furniture</th>
<th>Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floor</td>
<td>0.687</td>
<td>0.001</td>
<td>0.006</td>
<td>0.306</td>
</tr>
<tr>
<td>Ceiling</td>
<td>0.002</td>
<td>0.565</td>
<td>0.001</td>
<td>0.432</td>
</tr>
<tr>
<td>Furniture</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wall</td>
<td>0.091</td>
<td>0.065</td>
<td>0.008</td>
<td>0.836</td>
</tr>
</tbody>
</table>

Table 4.1: Confusion matrix of object classes

These results compare very favorably to those from the Brostow et al. [BSFC08] paper upon which this system is largely based. They reported a 61.8% global correctness for classifiers built using only 3D structure and motion. They also reported class-level correctness which does not weight the results by how common pixels of each class are in the test set. There they scored 43.6% as compared to 69.6% for our system. The results may not be directly comparable, however, as the Brostow paper was dealing with the more complicated environment of busy city streets and was using a much more detailed set of classes.
4.3. Analysis

4.3.3 Forest size

Another parameter that should have an important impact on performance is $M$, the number of trees within the forest. The greater the size of the ensemble, the longer tree generation and evaluation will take, but also the lower the expected variance in the results will be [GEW06]. This is the most important property of the randomized trees used in this system. Any one of them is going to give low quality results, but when they vote together on a response they can be quite good.

All experiments so far have been run with a $M$ of 7, simply to optimize execution time on an 8-core server. To evaluate the influence of $M$ a series of tests were run where this value was incremented from 1 to 42. This was done on the corridor-new dataset, with a fixed $K$ of 400. As before, 20 frames from outside of the training set were evaluated and compared against their hand-labeled ground truth.

The results from this experiment show that the correctness of the forests converges fairly quickly as a function of the number of trees, as can be see in Figure 4.5a. The value of adding new trees quickly drops off once the forest size is much past 10. We can be confident that the larger forests are not overfitting [Bre01], but neither are we gaining any added performance for all the extra computation.

By contrast, Figure 4.5b shows the cost of adding new trees. The frame evaluation time scales linearly as the ensemble size grows, which is to be expected. The exact cost-benefit can be selected as appropriate, but the rest of the experiments will continue using $M = 7$.

![Graphs showing output correctness and frame evaluation time](image)

Figure 4.5: Experimental results as a function of forest size $M$
4.3.4 Generalizability of training data

One theoretical advantage to using point cloud motion is that it should reduce the dependence on similar lighting conditions and obstacle appearance between training sets. The system relies only on the structure of the point cloud and its motion between frames, not the visual appearance. Thus it should generalize well between scenes that have similar structures but disparate appearances. In order to test this hypothesis, data sets from different locations were acquired. The system was then trained on one and tested against another, a process referred to here as cross-training.

The results of completely crossing the corridor-new and lab4 data sets in both directions are shown in Figure 4.6. As can be seen, contrary to expectations, cross-training seriously degrades performance. It is important to note that the lab4-trained classifiers did quite poorly even when working on frames from their own data set. However, the cross-trained classifiers still did significantly more poorly than either “homozygous” classifier. It is possible that the data sets being used are neither long nor structurally varied enough to support cross-training in this fashion.

While disappointing, these results are in line with those in the Brostow et al. [BSFC08] paper. They reported 45.5% and 56.4% global accuracies on their two cross-training experiments, down from 63.0%. Both of these were far larger than their results using an appearance-only classifier cross-trained in this fashion on the same data, something not replicated here.

4.4 Comparisons

In this section we will explore the contributions of different aspects of the classifier. This will be accomplished by replacing components with alternatives and examining the output generated using this new composite system. The goal here is to test if any of these major structural changes to the classifier would be an improvement and to gain an understanding of how these sub-systems interact.

4.4.1 Structure from Motion

One important question raised by this system is the importance of the source of point clouds. The system in the original paper used a commercial structure from motion tool, while we used the output of a stereo vision library. In order to compare the two, the Bundler tool was used to generate both point clouds and camera motion estimates for our training frames. The forest of
classifier trees was rebuilt using this underlying data. These trees were built and tested using the same 50-900 tree generation size parameter $K$ range as the previous experiments.

The output quality (Figure 4.7a) shows a drop of roughly 10% across the board. Frame evaluation times (Figure 4.7b) show an increase of 1.5-2.5 seconds. The trees generated using the structure from motion data are generally larger than the stereo counterparts (not shown), implying that the tree generation process is struggling to find meaningful split points. The results from using structure from motion are inferior by every measure. Combined with the added burden of running a computationally expensive structure from motion system, it seems like a plainly an inferior option.

However, these results may not be completely comparable, as the structure from motion system could only be tested against 10 frames instead of the standard 20. This was because the Bundler tool used to estimate camera motion could not handle the tight turns in the corridor. Even though

Figure 4.6: Comparison of correctness for cross-trained classifiers on the corridor-new (C) and lab4 (4) data sets
care was taken to provide frames during each turn free from motion blur, it was still too much for Bundler. It treats the sequence as a global problem, looking for the one optimal motion solution. In contrast the frame-by-frame approach used to estimate motion using tracked stereo points is much more flexible. It also gets confused during the sharp corridor turns, but it is able to recover gracefully afterward. Bundler just gives up. Because of this, the longest stretch of frames that could be processed using structure from motion was 200, giving 100 for training and the last 100 for testing. (With the standard one frame with a labeled ground truth every ten, giving us ten test frames.) An alternative approach would have been to find multiple stretches of frames, training on one while testing against the other, but this was not done due to practical considerations.

This limitation undermines the validity of the comparison here, but it ultimately speaks to the unsuitability of structure from motion (or at least the Bundler tool) for this task. In addition to taking far longer to calculate, it is just too brittle for the domain space being targeted here.

The structure from motion data also drastically changes the relative importance of the different features, as shown in Figure 4.8. The orientation features $f_{O_x}$ and $f_{O_y}$ are much more salient here than in classifiers trained using the normal stereo results (Figure 4.3b). The overall ordering has only been slightly changed; the closest approach to camera $f_C$ is exchanged with height above the ground plane $f_H$.

Given the poor results for the structure from motion system, one would expect the feature rankings to be more radically altered. The continued salience of the orientation features is also surprising, as those would be the
4.4. Comparisons

ones most harmed by incorrect point cloud structure. Thus the cause for the poor structure from motion segmentations is more subtle than just inferior data.

![Graph showing the number of nodes per feature in the structure from motion classifiers](image)

Figure 4.8: Number of nodes per feature in the structure from motion classifiers

4.4.2 Alternative stereo algorithms

The system being described depends entirely upon the output of the stereo processing algorithm it uses, as that is the basis for the point clouds that are the source of all the training features. For all of the experiments so far, this has been the Point Grey Research Triclops library which implements a cross-correlation algorithm. There are many other algorithms available, however, and even the more computationally expensive are becoming increasingly practical. To determine the exact dependence on the stereo output, the trees were rebuilt using the output from loopy belief propagation and Gaussian summation mask systems. The standard tests were run using these classifiers.
As can be seen, the results using either of the alternatives fall well below that of the baseline system (Figure 4.9). One intriguing difference between the results is how much more consistent those of the alternative stereo are. They show a slight improvement as $K$ increases, but from test to test the results vary far less than those for the default system.

The reason for the very consistent results is made obvious upon closer examination of the stereo results. Despite its superior results in more textured environments, the loopy belief propagation algorithm has completely failed in the domain of corridor-new. The cross-correlation stereo results are not great, but they are far superior in comparison (Figure 4.10). The point clouds being generated here lack contrast in the Z axis and are obviously incorrect much of the time. Unlike the cross-correlation output, the loopy belief propagation system used here does not label any pixels as invalid. (This is due to the specific implementation used and is not inherent in the algorithm. Adding a backwards validity check would not be a ma-
4.4. Comparisons

Figure 4.10: Example stereo depths for frames 110, 160, 190, 220 and 290 from the corridor-new data set. In both versions lighter shades represent greater proximity to the camera. Left: Input image. Middle: Point Grey Research cross-correlation. White pixels are invalid and have no depth associated with them. Right: Loopy belief propagation. A depth value is given for all pixels.
4.4. Comparisons

The lack of invalid pixels produces a denser point cloud but one that is cluttered with completely inaccurate points. The Gaussian summation mask does label invalid pixels, but otherwise its results look very similar.

Another important distinction is that the sparse Point Grey stereo depth estimates are concentrated around sharp edges in the scene. This is a drawback in many situations, but it may be uniquely suited to this classifier as applied to the corridor-new environment. The scene being reconstructed is almost entirely planar with very little texture at smaller scales. With valid stereo depth estimates given for the edges, the classifier can reconstruct the rest with a high degree of accuracy. This would make the surface orientation features $f_{O_x}$ and $f_{O_y}$ highly salient. In contrast, the loopy belief propagation output includes a depth estimate for every pixel. In a planar environment like this, this means the scene reconstruction can at best be as good as the Point Grey results, and often will be much worse. The Gaussian summation mask results, which are still much more dense despite having invalid pixels marked, have a similar upper bound.

Because of all this, the output frames using the alternate stereo point clouds have almost uniformly been labeled as wall, with only a small scattering of floor and ceiling pixels. As shown in Table 3.1, the wall class dominates this training set. Without useful point clouds the classifier breaks down and can only generate a uniform prior guess.

To better understand how the stereo algorithm affects the system results, classifiers were cross-trained on one stereo output and tested against the others. The results, shown in Figure 4.11, show a striking difference based on the stereo algorithm used. Both of the alternate algorithms show the same flat response when used to classify the original data (L on N, J on N) as when run against themselves. Normal cross-correlation fails when run against the loopy belief propagation (N on L), but with a variance that shows it is doing something more interesting than uniformly classifying the output as wall. Most intriguingly, when normal cross-correlation is run against the Gaussian summation mask (N on J) the results are only slightly worse than when run against itself. Despite the Gaussian summation mask failing when used to train classifiers, it turns out to be perfectly useful in the test phase of operation. Building the trees used for classification is obviously a very delicate process, much less tolerant of faulty input than testing is.
4.5 Extensions

In this section a series of extensions to the original system will be explored. Not as major as the changes tested in the previous section, these extensions look to improve the performance and computational efficiency of the system.

4.5.1 Aggregated voting

In an effort to improve performance, an optimization was explored that aggregated the classifier results over much larger regions. In these experiments, the image was first broken into regions using either a superpixel algorithm or a regular grid. Within each region, $V$ pixels were randomly chosen and run through the classifier. The results were then used to vote for the correct classification of the entire region.
4.5. Extensions

This was done with two main expectations: that evaluation times would be radically improved and that output quality could also be improved. With only $V$ pixels being classified in each region, frame evaluation times should increase based on the average size of each region. The larger they are, the greater the speed improvement. Output quality should be improved because the classification would be consistent within each region, reducing the problem of noise and irregular boundaries. In the case of the superpixel version, it was also hoped that by forcing segmentations to follow natural boundaries the results would be improved in both a quantitative and qualitative sense.

![Superpixel classification examples](image)

Figure 4.12: Superpixel classification examples

![Grid classification examples](image)

Figure 4.13: Grid classification examples

**Superpixel aggregation**

For the first set of experiments, $V$ is set to be 9. An example of the output for the superpixel experiment can be seen in Figure 4.12. On a purely qualitative level, these segmentations are a great improvement. They lack the fuzzy, jagged outlines of the standard output and they directly reflect shapes obviously present in the test frame.

Figure 4.14 shows the correctness values for superpixel-based aggregation (using large and small region sizes) compared against a baseline of the
standard output. The small superpixel regions make basically no difference, but the large ones are often a small improvement. This is one of the tricky situations common in segmentation evaluation where the quantitative and qualitative measures are in conflict. It is unclear if this extension is a real improvement. Evaluation time is reduced because many fewer pixels are evaluated, but that excludes the cost of computing the superpixel segmentation in the first place.

![Figure 4.14: Comparison of correctness for superpixel experiments](image)

**Grid aggregation**

An example of the grid version output can be seen in Figure 4.13. This version was run for square grid sizes between 5 and 95 pixels in increments of 5. The output is definitely an improvement in that it is more regular than the standard system. The blocky pixelization created by the grid is not very pleasing, however, and overall it does not end up feeling subjectively like a great improvement in output quality.

In contrast with the superpixel version, the results for these simple grids
4.5. Extensions

show a clear improvement. Figure 4.15 shows a detailed sample of these results. The complete range of values tested is shown in the surface plot in Figure 4.16.

![Figure 4.15: Comparison of correctness for grid sizes 20, 40 and 60](image)

The best results are achieved with a grid size \(B\) of 20 pixels. As with previous experiments, there is a definite preference for tree parameter \(K\) around 400. The improvements there are not very high, going from a correctness of 82.3% to 82.9%. However, due to the fact that we are sampling just 9 pixels out of a 400 pixel region, the frame evaluation time has dropped from 8.62 seconds to 0.38. The small performance enhancement that comes with this extension is an additional bonus, but it is the speed improvement of over 20 times that makes it absolutely worthwhile.

**Aggregation voting size**

With \(K\) and \(B\) now optimized, we can explore the influence of voting size \(V\). We would expect that, like forest size \(M\), the more voting members contributing to the final classification, the better the results would be.
This was tested by running a series of grid segmentations with $K = 400$, $B = 20$ and $V$ incremented from 1 to 42. The results are shown in Figure 4.17. Similar to the results seen when testing $M$, the initial improvement in quality is several percent given just a few extra votes. This quickly levels off with little improvement past 5, making additional votes pointless. Setting $V = 6$ would be entirely reasonable, and would show even greater speed improvements than reported above.

Analysis

Contrary to our original expectations, the grid aggregation outperformed the superpixel version, even without accounting for the computational overhead of generating superpixel segmentations. It has previously been seen that classifications on a grid outperform more advanced segmentations such as superpixels [CipBus+14]. That paper speculates that the reason for this is that the regions perform better when they are less likely to contain multiple object classes. This could explain the preference for relatively large blocks.
4.5. Extensions

in this system, as the objects being classified (wall, floor, ceiling) are all quite large. The grid segments can be much larger than the superpixel segments, which might explain their improved relative output.

4.5.2 Appearance features

The default system tested up to this point has only used the structure and motion of point clouds. The appearance of the original frames has been completely ignored. While there are advantages to doing this, it ignores a vast wealth of information that could help improve classification results. In any generalized system the visual features would certainly be used for other aspects of visual navigation (such as recognizing specific locations), so it is reasonable to test their usefulness here.

In order to see how useful the addition of appearance-based features could be a series of experiments were run testing their contribution. The classifier trees were rebuilt for the entire 50-900 \( K \) parameter range, but this time the four texton features were included along with the previous six.
4.5. Extensions

With the trees rebuilt, they were used to segment the same 20 test frames as in the other experiments.

As shown in Figure 4.18, the texton extensions were a mostly positive addition to the system. They improved results by up to a full percent in output correctness, particularly in the $K$ sweetspot around 400 that has previously been identified. This improvement does not come without a price, however. Tree generation took between 2 and 10 times as long. Evaluation times were not significantly changed, so the question of whether the improvement in correctness is worthwhile would purely be a matter of how much pre-processing was practical.

![Figure 4.18: Comparison of correctness with and without the texton extension](image)

The Brostow et al. system also tested the addition of texton features. They reported a much more impressive gain of almost 5% improvement in their results, bringing their global correctness to 66.5%. This disparity might be because they had much more room for improvement to begin with, but it might also be due to differences in the training data. The corridor sequence
used here has a fairly uniform appearance throughout it. There might just not be enough appearance information for texton features to work from.

### 4.5.3 Biased window distribution

The sampling window properties of size and offset have, up until this point, been randomly chosen from a uniform distribution. It could be a great performance boost if these were biased to more closely match the preference of the randomized trees. By doing so, higher quality trees could be generated at a lower $K$ setting by providing better sampling windows from which to choose.

In order to investigate this possibility, the distribution of the sampling windows selected in the tree generation process was examined. Using the trees generated for previous experiments, the sampling windows for each node were extracted. This gave a total of 74,274 examples.

![Figure 4.19: Sampling window attribute distributions](image-url)

(a) X offset  
(b) Y offset  
(c) Width  
(d) Height
4.5. Extensions

As can be seen in Figure 4.19, there are indeed patterns to the sampling windows most likely to be chosen. The offsets in X and Y are both fairly symmetric, which is to be expected. None of the features should have an asymmetric bias — if class A is usually above class B, then class B should usually be below class A! The X offset is mostly uniform, but the Y offset does show a slight preference for values away from zero. This demonstrates that there is somewhat more information at longer ranges in the vertical arrangement of classes as compared to the horizontal arrangement. Particularly in an indoor environment, this is reasonable. The arrangement of floor/wall/ceiling is much more reliable in the vertical axis than for the horizontal, where it can be more easily confused by the exact scene geometry and camera position.

Looking at the preferred width and height values, these show an asymmetrical bias against smaller sizes. This is also to be expected as most of the features used in the system are fairly sparse, being abstracted from tracked image points. Particularly in low-texture environments, the gap between these can be quite large. Smaller sampling windows would be less likely to overlay a tracked point, resulting in uninformative features values of constant zero.

Experiments were performed with the pool of windows generated biased to reflect these patterns. Instead of the original uniform distribution the Y offset sampling used a piecewise triangular distribution 10% of the time, while width and height used one 25% of the time. This resulted in window pools that more closely matched the observed preferences.

The results are very encouraging. The biased version is up to 0.5% more correct (Figure 4.20a) and over a second faster per frame in places (Figure 4.20b).

Figure 4.20: Biased and normal classifier results
4.5. Extensions

[4.20b]. The frame evaluation time comparison is particularly telling. The biased window version does not show the sudden drop at around $K = 500$. This strongly implies that the biasing is accomplishing exactly what it was designed to do. With this bias, the best sampling windows are more likely to be available to the tree generation process at lower $K$ values. This not only improves output quality, but it means the timing improvements that go with it are also gained early on. Thus the frame evaluation curve is much smoother than with the default system. Tree generation times and tree size (not shown) were not noticeably changed.

The gain from this extension is not great, but the cost of adding it is completely negligible. There would be no reason not to leave this enabled in future versions of the system.
Chapter 5

Conclusion and Future Work

5.1 Contributions

This thesis has demonstrated the abstract possibility of generating semantic segmentations using the structure and motion of point clouds derived from stereo camera input. This was tested in an indoor environment, with the aim of creating output useful for robotic navigation. While the results score well using the quantitative evaluation, they are a long way from being practical for this target domain.

The results were compared against a variety of extensions to better understand the strengths and weaknesses of the system. All of these showed the existing arrangement to be the best option. It was shown to be superior to the version based on structure from motion point clouds. Input from alternative stereo algorithms was also shown to be unhelpful.

A series of extensions to improve system performance were tested. All of these showed some amount of promise. The aggregated voting approach was both more accurate and faster to evaluate, particularly using a grid segmentation. The use of appearance features in the form of textons also provided a minor performance boost. Possibly most intriguingly, it was shown possible to improve performance and particularly evaluation time through the biasing of the sampling window pool generated during the training of the classifier. By making this pool more closely reflect the distribution of sampling window naturally chosen the resulting trees ended up being both more accurate and better formed.

5.2 Future Work

5.2.1 Real time operation

The performance of the current system is far from what could be desired. Even writing off the issue of tree generation times as allowable pre-processing, frame evaluation is still over seven seconds per frame. In order to be of practical use it would need to be sped up many times into the 5-10 frames per
5.2. Future Work

There is nothing inherently slow about the process, however. It is likely that switching away from the overhead of the Python numerical processing libraries would improve things greatly.

More troublesome is the fact that the current system assumes future knowledge when classifying pixels. Most of the features work using the path of the tracked point through all of the frames it has been seen in, not just the current. The $f_C$ feature, which encodes the closest approach of a point to the camera’s path, can only work given the camera’s path dozens of frames into the future, usually well past when the tracked point itself has disappeared from view. This is obviously impossible in an online system. If the approach covered in this thesis is ever going to work in a real time fashion, these aspects will need to be changed. Many of the other features might still work without the knowledge of future movement, but there are likely other feature sets better suited to that condition.

The features used here were taken directly from the original source papers. While they have proven suited to the task it is not obvious that they are the best possible set. As was seen with the texton extension experiment, the choice of features can have a large impact on the results. As well as new features, the relative strengths and contributions of the current set should be explored. $f_R$ and $f_D$ may be redundant in a static world environment.

The results from the grid aggregation show that there is a lot to be gained from simple improvements. The window biasing results demonstrate that not only can performance be improved but so can the general quality of the trees in the classifier, resulting in shorter evaluation times for the same size. This was done by trying to match the sampling window distribution to that seen in those chosen by the tree generation process. This is only the most basic and naïve approach that could be taken. Further experimentation would almost certainly be a fruitful avenue.

5.2.2 Output evaluation

In this thesis the technique used for the evaluation of segmentation output has been that of reporting the percentage of pixels labeled correctly. This is a venerable approach to the problem, but it is flawed in many ways. It does not take into account any of the properties of natural segmentation boundaries that we might wish to capture in our evaluation. It does not penalize noise or highly irregular boundaries such as those that naturally result when each pixel is classified independently. It also fails to account for the local geometry of the segments in question. Pixels of class A that are labeled as class B should be evaluated as “less wrong” the closer that actual
5.3 Conclusion

pixels of class B can be found.

Semantic information also needs to be taken into account. Not all mistakes are equivalent, particularly for purposes of robotic navigation. Falsely labeling a wall pixel as ceiling is far less serious than labeling it as floor. While ideally the system would generate the correct classification for everything, the most important thing is knowing where it is safe to drive. This is application dependant, but in generally terms, the more semantically related two classes are, the lower the penalty for a misclassification should be.

A hierarchy of classes is needed to formalize this intuition. Towards this aim a prototype image annotation application was developed. Called Segmentree, it allows the user to segment images into a spatial hierarchy with arbitrary labels. This was never put into practice, but it was hoped that a user study could collect hierarchical segmentations from multiple people. These hierarchies could then be combined by adapting the pyramid method from computational linguistics [NPM07] to recover a global semantic hierarchy. An evaluation based on this would be both principled and an improved reflection of how people think about physical reality.

5.3 Conclusion

Humans create meaning. Even the simplest of our environments is overlaid with uncounted layers of semantics. Without an understanding of this meaning, safe and useful interaction with the environment is impossible. Treating it simply as a collection of obstacles will never be enough – as important as not running into a stop sign is, recognizing what the stop sign means will always be more important.

Computer vision systems are the obvious choice for acquiring this level of understating. Most human environments are designed with vision as the primary modality. Furthermore, the only other option is to use devices like 3D laser range finders, which are prohibitively expensive for most applications. Systems will only be practical when built around an inexpensive input device like a camera.

As Moore’s Law continues to make computer vision systems more and more prevalent, the need for them to provide semantic understanding grows. Until they are capable of this, it seems unlikely that truly autonomous applications will be possible in anything but the most controlled of environments. Systems like the segmentation described in this thesis provide an important step towards achieving this goal.
Bibliography


Chapter 5. Bibliography


Chapter 5. Bibliography


Chapter 5. Bibliography


Chapter 5. Bibliography


