

A System for Unsupervised Color Based Object Classification

by

Stephen Randolph Preer

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

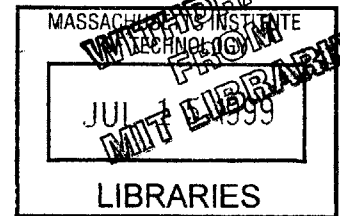
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999

[June 1999]

© Stephen Randolph Preer, MCMXCIX. All rights reserved.

The author hereby grants to MIT permission to reproduce and
distribute publicly paper and electronic copies of this thesis
document in whole or in part.



Author
Department of Electrical Engineering and Computer Science
May 18, 1999

Certified by ...
W. Eric L. Grimson
Professor
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A System for Unsupervised Color Based Object Classification

by

Stephen Randolph Preer

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis describes a system for unsupervised color based object classification and recognition. Color histograms are generated for all image data related to an object. These histograms are normalized to minimize lighting effects. They are compared and clustered with a recursive associative minimum cut algorithm. The resulting hierarchy is thresholded and the clusters formed are used to create prototype histograms that subsequent objects may be matched against. A variety of color histogram configurations were tested to determine the best color space and bin configuration. This system is an extension to the Forest of Sensor visual surveillance and monitoring project.

Thesis Supervisor: W. Eric L. Grimson
Title: Professor

Acknowledgments

I would like to thank:

- Eric Grimson for making this thesis possible.
- Chris Stauffer for his advice and support.
- Vis Taraz for letting me go in depth into any sort of problem I was having, and taking the time to listen and provide encouragement.
- John Rieffel for letting me bounce ideas off him, and giving me good advice, as he worked in a similar vein elsewhere.
- My housemates, Alica, Heather, Luke and Bubba who helped me decompress and have a life outside of lab.
- To our cats, Luna, Miso and Kozmo who were willing test subjects and a constant source of entertainment and affection.
- Everyone at pika for keeping me fairly sane through the four years it took me to get to this point.
- Most importantly, thanks of course to my family for all of the many kinds of support that they have provided over the years.

Contents

1	Introduction	8
1.1	The Forest of Sensors Project	9
1.2	Goals	9
1.3	Assumptions	11
1.4	Overview	12
1.5	Conclusion	13
2	Background	14
2.1	The Forest of Sensors	14
2.2	Color Histogram Intersection	16
2.3	Clustering	17
2.4	Conclusion	18
3	Color Analysis	19
3.1	System Overview	19
3.2	Retrieving Data	19
3.3	Color Analysis Methods	20
3.3.1	Why Color Histograms?	20
3.3.2	Creating the Histogram	21
3.4	Design Decisions	21
3.4.1	Color Spaces	21
3.4.2	Histogram Bins	22
3.4.3	Bit distribution	23

3.5	Real World Data	23
3.6	Conclusion	25
4	Clustering	26
4.1	Why Min-Cut?	26
4.2	The Algorithm	27
4.2.1	A 2-D example	28
4.2.2	Generalized to N Dimensions	28
4.3	The Distance Metric	29
4.4	Outputting the Data	30
5	Thresholding and Output	31
5.1	Getting the Data Back	31
5.1.1	Parsing the File	32
5.1.2	The Data Structure	32
5.2	Thresholding	33
5.3	Conclusion	34
6	Visualization and Other Side Issues	35
6.1	Color Histogram Visualization	35
6.1.1	A Three-Dimensional Approach	36
6.1.2	A Two Dimensional Approach	37
6.2	Visualizing the Hierarchy	38
6.3	Visualizing the Clusters	39
6.4	Conclusion	40
7	Results	41
7.1	The Data	41
7.2	Cars	42
7.3	Cats	43
7.4	System Components	44
7.4.1	Color Spaces	44

7.4.2	Bin Distribution	44
7.4.3	Normalization	48
7.4.4	Thresholding	48
7.5	Conclusion	50
8	Conclusion	56
8.1	Future Work	56
8.1.1	Other Applications	56
8.1.2	Histograms	57
8.1.3	Clustering	57
8.1.4	Visualization	58
8.2	Summary	58

List of Figures

1-1	The view	12
4-1	Clustering on the plane	29
6-1	Histogram visualization	37
6-2	A car clustering and its 2D histogram	37
6-3	A sample hierarchy	39
7-1	A sample car sequence	42
7-2	A sample cat sequence	43
7-3	The min-cut hierarchy for HSI 321	46
7-4	The min-cut hierarchy for RGB 222	47
7-5	A clustering of the cat data HSI 231	49
7-6	Clustering with old thresholding method.	51
7-7	Clustering with new thresholding method.	52
7-8	Clustering of the test data.	53
7-9	A clustering of a half hour of real data.	54
7-10	A clustering of headlights.	55

Chapter 1

Introduction

It has long been the goal of Artificial Intelligence to imbue machines with human abilities. One of these abilities is that of vision; the ability to detect and classify objects in a scene. This problem is a central one. If a machine is to be fully aware of its surroundings it must be able to see. Unfortunately this is a difficult problem, and very much an open one. It is agreed by most, however, that one of the key elements to vision is color. Color is one of the most powerful descriptive tools in the human lexicon, and one of the most useful tools in object identification.

The application of color vision that will be studied in this thesis is that of visual surveillance and monitoring. There are many situations in which it is desirable to place a camera to monitor a scene. Whether it is to study animal behavior or to maintain security in a building, there are thousands of cameras pointing at scenes twenty-four hours a day. Rather than placing a human intelligence behind all of these cameras, it might be better to place an intelligent visual system behind each. This means more cameras can be placed and more data can be gathered, in a way that would be difficult and time consuming for a human. The key to such a system is the ability to recognize and classify common objects and behaviors in the scene, and consequently to be able to recognize unusual objects and events.

1.1 The Forest of Sensors Project

The Forest of Sensors Project as described by Grimson and Stauffer [2] endeavors to create such a system. This system uses attributes such as size, position, velocity and direction to classify objects in a scene. The input to the system is from a five cameras placed around the seventh floor of the AI Lab. The final goal is to build a system that can classify objects and actions in a scene. This classification can then be used to recognize objects and actions that do not fit the established pattern. A query based engine can then be used to retrieve instances of these objects and actions over a given time. The additional trait that will be discussed in this thesis is color. The goal of this extension is to create a system that will be able to inspect the many objects over long periods of time, and compare them to determine which objects are of the same type, and which are not. This information will be used to create templates of object types that any new objects may be compared against. This enables the gathering of statistics on the activities of object types, and allows new objects to be recognized without having to be compared with all past data. The two key techniques used will be color histogram intersection and set partitioning using a recursive minimum cut algorithm.

1.2 Goals

The question of object recognition using color histogram matching is one that has been explored extensively. Much effort has gone into determining the best color space and the best distance metric to use when comparing histograms. Most of this work, however, has been in the context of recognition of known objects. For example, an image or a collection of images of the object in question will be compared against input images, and the resulting histogram distance will be used to determine whether or not there is a match. The Forest of Sensors project is an unsupervised system, and therefore color based recognition should be approached in an unsupervised manor.

The final goal is, of course, the same. The desired output of the system is a

set of prototype histograms that may be compared against input images and rated for a possible match. The challenge is to extrapolate these prototypes directly from the data. The obvious starting point is a clustering algorithm. The system should attempt to cluster like images, in order to build up a thorough data set for each object.

There are many design choices to be made in the construction of such a system, and it is necessary to enumerate a set of goals so that we may decide if the choices made were good ones. Here are the primary goals of our system:

- It should run with minimal, if any, human input.
- It should be robust to lighting changes.
- It should run in real time.
- The effects of artifacts of the underlying vision system should be minimal.
- It should perform in a way that corresponds closely to human intuition.

First, in the specification of this system, it was stated that it was to be unsupervised. Ideally it should be possible to place this system in front of an arbitrary scene and have it do a good job of classification. Second, since the system will be running all day and night, it should be possible to recognize an object at dusk as the same one that was seen at noon. Third, since this system may be running for long periods of time, it would be best if it could process the data it receives as fast as it comes in. That way there is no backlog. Fourth, the system should not be thrown off with an occasional piece of erroneous data.

The first four criteria seem fairly clear, but the fifth is a bit confusing. What is meant by human intuition? The difficulty addressed here is that there is no correct answer. When a vision system is charged with recognizing a specific item, it is easy to judge its performance. The number of incorrect matches and missed matches can be used to create a quantitative measure of the effectiveness of the system. However, if there is a large quantity of data, say the cars going past a building, the goal is not

to recognize a particular car, but only to have a general idea of what, for example, a red car is. If ten humans were given the task of dividing a set of a hundred cars into set of the cars that “looked the same,” no two people would give the same answer. However, there would probably be a strong similarity. Consequently, all that can be hoped for is a system whose output exhibits a strong similarity to what a human might construct. This is the meaning of the goal of human intuition.

1.3 Assumptions

It is necessary to make some assumptions about the underlying vision and classification system. First, it must be assumed that the great majority of the time it outputs data concerning valid moving objects. The color system should be able to deal with some artifacts, but for the most part should not have to worry about them. It can correlate between frames to pass in image data for a single object over multiple frames. It will be much easier to identify an object if there are 100 frames of data, rather than only one. Finally, it can effectively classify objects using other traits, so that the color classification make take place over only objects that have other similar properties. For example, in the primary application of this system, the two primary types of objects are cars and people. While it would be possible to simply use all of the data for both in color classification, the results will be much better if instead the two groups can be clustered individually. As will be seen in the description of the Forest of Sensors project, all of these assumptions are valid.

For a period of time in the Summer and Fall of 1998 there were five cameras pointing from the seventh floor of MIT building NE-43 at the street and parking lots below. One of these views is shown in Figure 1-1. This was the primary data set used, and it shaped many of the design decisions made in the construction of the system.

Figure 1-1: The view



1.4 Overview

The details of the implementation will be discussed in depth later, but here is an overview of how the system operates. The unit of time that the system will consider will be a half hour. During peak times around two hundred cars would pass the cameras. This is a period long enough to gather a significant amount of data, but short enough that significant global lighting changes would not take place. All of the data for each moving car is used to create a color histogram, which is a quantization of the colors in the object. An average of all of the histograms is taken and subtracted from each histogram. This normalizes the histograms and ensures that what is being measured is the distance from the mean. All pairs of these histograms are then compared using a Euclidean distance metric to create a matrix of distances. These distances are transformed into measures of similarity for use by the clustering algorithm. The clustering algorithm recursively divides the set of histograms in two by determining the minimum cut over all the histograms; in other words, how to divide the set in two such that the sum of the similarities between the histograms that are separated is the least. This creates a classification tree that may then be thresholded at a certain cut value so that only cuts better than a certain value will be made. This leaves several groups of histograms, each of which corresponds to a particular color type. Each of these groups is then averaged to create a prototype histogram that may then be compared with new data.

1.5 Conclusion

This section has given an overview of the project. The context and goals of the project have been laid out, and a general view of the process used has been presented. The following chapters will discuss background, implementation and the conclusions that may be drawn from the results in greater depth.

Chapter 2

Background

In order to better understand the context of this project, it will be useful to include a discussion of some past work. First, the Forest of Sensors project will be described in some depth. A full description will be given of its goals, its operation and its interfaces. Second, past work in the field of color based recognition will be discussed, in order to understand the design tradeoff that must be considered. Finally grouping algorithms will be discussed, including the wide range of methods available to achieve the desired result.

2.1 The Forest of Sensors

The Forest of Sensors project was outlined in the previous chapter. Its objective is to create a system that can run in real time and automatically classify actions and objects in a scene. It is composed of a tracking system which masks out moving objects from their backgrounds, and tracks them from frame to frame. At a higher level, there is a classification system that creates a classification hierarchy based on the objects' sizes and aspect ratios, as well as where they are and how they are moving. The objective is to create a system that can recognize unusual objects or behaviors, as well as locate instances of known objects and behaviors. The object of the system to be described in this paper is to extend the classification hierarchy to include the coloring of objects, which can be a powerful identification tool.

The Forest of Sensors project's goal is to create a video surveillance and monitoring system that fulfills three primary goals: persistence, full coverage and automatic configuration and processing. In other words, the system should be able to cover more than one camera's field of view and run constantly for long periods of time without human supervision.

At the deepest level of the system there is the tracking program. This is the portion of the system that is able to mask out moving objects from their background. The BTracker program written by Chris Stauffer and described in Grimson and Stauffer [2] uses an adaptive process where each pixel is treated as independent statistical process that can be modeled by the mixture of multiple Gaussians. This method has all of the benefits of a single Gaussian model, which allows for a match given a certain amount of noise, but also has certain advantages. The primary advantage is that the use of multiple Gaussians allows the system to be robust in the face of periodic changes such as transient lighting effects or swaying tree branches.

The tracking system is also adaptive. As time passes, each new value of each pixel is compared against the previously calculated distribution, and if no match is found the new value is added as a new but improbable distribution. This adaptation allows the system to be robust over global lighting changes, such as cloud movement or sunrise and sunset.

The pixels that do not match the background are then clustered into connected components, and matched to components in previous frames to create correspondence over time. At this point the system can recover a large amount of data about a particular moving object. This data includes size, position, velocity, direction and aspect ratio. This data will then be used to create a classification hierarchy. This classification can effectively separate object types with different attributes and behaviors. For example, cars are large, fast, are wider than tall, and stay on roads. People are small, slow, taller than wide, and stay primarily on sidewalks.

2.2 Color Histogram Intersection

Since Swain & Ballard [7] published their seminal paper on color histogram intersection, there has been a fair amount of related work, as well as number of commercial applications of similar techniques. As this work is an important basis for the system to be described, it would be wise to survey it.

The basic goal of color histogram intersection and related techniques is to compare images of objects on the basis of the colors they contain. Simplistic approaches might involve comparing the average color or the most commonly occurring color in an object or image. These techniques, while fast and effective for quickly eliminating far misses, are not particularly accurate.

Basic color histogram intersection as described by Swain and Ballard works like this: a bin is created for each color that could be present in the image. The image is then inspected a pixel at a time, and for each pixel of a particular color, the value in that bin is incremented by one. When the process is done, the result is normalized by dividing by the total number of pixels in the image. What is left in each bin is the fraction of the pixels that are that particular color.

Now that histogram have been produced, it is necessary to compare it to other histograms. Swain and Ballard use a “city block” distance method of comparison. For each pair of corresponding bins in the two images the minimum of the two values is found. All of these values are added together, and the result is a measure of similarity. It is clear that if all of the bins contain identical amounts, the sum will be one, and the sum can only be zero if the two images have no pixels in corresponding bins.

Color histogram intersection is a technique that has many diverse applications, from the analysis of healing skin wounds[1], to screening out “objectionable” image content[8], to doing content based image searches. This last application of color histogram intersection is probably the most widely used by the public. Content based images searches, such as images.altavista.com, can quickly pull out images that are similar to test images, despite the immense size of the database.

There has also been work in the realm of color based classification, and some it has

found its way into industry. There is one company that is using color classification to do quality control in the food processing industry[4]. For example, instead of counting how many red cars pass on the street, they may count how many red jelly beans go into a bag.

2.3 Clustering

The final body of work that is necessary to discuss is that of grouping. The problem, generally stated, is how to locate clumps or clusters in some distribution of points. There are many approaches to this problem. Some are incremental and some work in batches. Some are exact and some are approximate. This section will simply look at a few of them.

First it is necessary to come up with a formulation of the desired solution. One possible formulation is that the solution should minimize the average of the distance between all pairs within each cluster. The formulation has the disadvantage of favoring smaller clusters. Another is to minimize the Gaussian entropy of a hierarchical classification of the data.

Let us now explore the differences between exact and incremental algorithms. The advantage of an incremental algorithm is that clusters can be updated without having to go back and recompute everything. This can be a huge advantage. As will be seen, exact algorithms are computationally demanding, so when dealing with a large data set that grows with time, an incremental algorithm sounds very tempting. It has the large disadvantage, however, of not being exact. Exact algorithms, unfortunately necessitate a move to a batch based method. This is not as bad of an idea as it first might appear. First of all, in the primary data set, global lighting changes such as sunrise and sunset will cause the bulk of the data to become darker or lighter over time. As a result there may be “smearing” in the clusters as night approaches, and the effect can result in larger and less distinct clusters, and as a result, poorer recognition.

The most basic of the exact clustering algorithms is the recursive min-cut algo-

rithm. The theory here is to divide something in two in a natural way, cut it along the weakest path. In more formal terms, the algorithm should look to partition the set of all nodes into two subsets, such that the weight of the arcs that must be eliminated to separate these subsets is minimized. In this case, the weight of an arc represents the degree of similarity of the two nodes it connects. This technique is a derivative of the Max-Flow-Min-Cut formulation. Once one cut has been made, each subset is recursively partitioned. If the minimum cut is above a certain threshold value, the algorithm stops. All nodes in the failed cut are then a cluster.

The actual implementation used is, in fact, an associative min-cut, and the details and motivation behind the use of this variant will be discussed more fully in the implementation section.

2.4 Conclusion

There is a variety of work upon which this thesis draws. It is built to take input created from the BTracker program. It relies heavily upon traditional color histogram intersection, as well as a variant of a recursive min-cut algorithm. These pieces are put together to form a system that does a good job of unsupervised color based object classification.

Chapter 3

Color Analysis

Now it is time to discuss the meat of the program itself. This chapter will begin with an overview of the system and its parts, and how they interact with the rest of the system as it exists. Next it will discuss in depth the Color Histograms used, and the tradeoffs involved there. Then, the clustering method will be detailed. Finally a number of smaller issues such as output and data visualization will be discussed.

3.1 System Overview

The system exists in three main parts. The first creates color histograms from tracking data. The second creates a clustering hierarchy of this data. The third thresholds this hierarchy to create clusters and generates output. The bulk of the code was written in C++, as that was what the BTracker program had been written in. The second portion of the code was written instead in matlab, which was chosen for its power in the context of matrix manipulation. The output is in a number of forms, including track files, html and inventor.

3.2 Retrieving Data

The first order of business is retrieving the data to be studied. The BTracker stores this data in files called “track files” in a hierarchical file structure. The root of this

structure specifies the year, and subsequent subdirectories specify the day of the year and the ten-minute interval in which the data was captured. The final directory contains a track file for each camera. The track file is a relatively simple structure that for each object that is tracked, contains a certain amount of data. This data is, naturally, divided up by frame. For each frame in which the object is present, the track file contains the size, position, velocity, direction and aspect ratio of the object in that frame. It also contains a 32x32 bit image chip of the portion of the image that contained the object. If the object is larger or smaller than 32x32, it is normalized to fit in that space. The alpha channel specifies whether a particular pixel was part of the moving object or not. The system is also able to load images, for demonstration purposes, and the implementation of this feature is too trivial to discuss.

3.3 Color Analysis Methods

Next the details of the color analysis system will be discussed. The first subsection will discuss why color histogram intersection was used. Later subsections will discuss the relevant design decisions, including the color space, the number of bins and the bin distribution.

3.3.1 Why Color Histograms?

While there are many ways to quantize color in an image, the most commonly used and effective is that of color histogram matching. There are other methods stress the mean or mode color in an image. For example, it is possible to represent each object as a Gaussian cluster in color space, as do Haralick and Kelly [3], where there is an average color and some lesser probability of increasingly dissimilar shades. In order to compare an image against this formulation, each bin, or some subset of the bins, is multiplied by a number. This number falls off as a Gaussian with the Hamming distance from the average color. The results are summed to create a distance metric. These methods are generally only desirable in cases where computational efficiency is at a premium, or where objects tend to be of a single color. It would seem that in the

case of looking at cars in real-time, this would be the case. However, it is important to remember that the scope of the Forest of Sensors is broader than simply looking at cars, and the system should be adaptable enough to deal with more interesting objects.

3.3.2 Creating the Histogram

Naturally it makes sense to use all available data to construct a color histogram. It could be argued that it could in some contexts be better to calculate histograms on a per frame basis, here that seems like a bad idea. With only a maximum of 1024 pixels per object per frame, and in practice closer to half that, this leaves little data with which to create histograms. Instead, for each object the system looks at all image chips to create our histogram.

The assumption is made that in the final working state, the tracker will also pass classification information on to the color classifier. That, at present, is not the case. Fortunately, it is relatively easy to set constraints that insure that an object is, for example, a car. That is what was done here, as a stopgap measure. All objects too small, too slow or too briefly seen are eliminated, leaving objects that are almost exclusively cars for classification.

3.4 Design Decisions

There are many choices to be made in the design of the color analysis. For the most part, the system was design is such a way the many possibilities could be studied, and the best among them could then be selected. The following sections will discuss some of those tradeoffs.

3.4.1 Color Spaces

Perhaps the most important choice to be made is that of the color space. There are many different color spaces, and even in past work with histogram intersection, many

color spaces have been used. The easiest to implement, and the least computationally intensive is RGB. This was the color space that was initially chosen, for those reasons. The primary disadvantage of RGB is that while the resolution of the histogram can be scaled as a whole, there is no way to decrease or increase the resolution of hue, for example, while decreasing the resolution of intensity. This might be a desirable ability if it is necessary to take into account the effects of differing lighting conditions. There are, however, many color spaces that do have this ability. The most obvious is HSI, (hue, saturation and intensity). More recently the Commission Internationale de L'Eclairage (CIE) has created a set of color spaces that better correspond to light as it is observed. Included among these is CIE xyY. In this color space the first two components, x and y, specify chromicity, while the third, Y, specifies luminance. This method proves to be as good as HSI in distinguishing between color and lighting, while being closer to the way the humans perceive color.

3.4.2 Histogram Bins

The second factor to consider is the number of histogram bins to use. Once again, this is a tradeoff, and it depends to an extent on the nature and quality of the data. If there are too many bins, there will be little data in each, and it will be hard to achieve a strong match. If there are too few bins, the resolution will not be great enough, and may be more difficult to distinguish similar objects. In the case of the Forest of Sensors data, there are several sequential images associated with each object. Each of these image chips is scaled down to 32x32, regardless of the portion of the field of view it occupies. Of that 32x32 chip, only half of those pixels may be in the object. As a result, the total number of pixels in, say, a car, is likely to be no greater than a few thousand pixels, which is orders of magnitude fewer than in the images considered by some color histogram intersection applications. For example, Swain and Ballard[7] used 512 x 485 images, which contain around 250 thousand pixels. In fact some objects may only be tracked for a few frames, leaving only hundreds of pixels to consider. As a result, it is desirable to keep the number of bins reasonably low. It would be silly, for example, to have a larger number of bins than of pixels.

It may seem as though having a smaller number of bins might be computationally advantageous, however, it turns out the time it takes to calculate the intersection of all pairs of histograms is dwarfed by the time it takes to cluster them. This is an issue that will be explored later on.

There are twenty-four bits of data per pixel, so the largest number of bins that could conceivably be used is 2^{24} or about 16 million. This is clearly unreasonable. The chances that two pixels would fall in the same bin at all are phenomenally small. Swain and Ballard chose an 11 bit color space, which has 2048 bin, for their images. Between 6 and 12 bits, or 64 and 4096 bins seems like a reasonable range for the number of pixels being considered. The results of a large number of bin choices applied to a representative data set will be seen later.

3.4.3 Bit distribution

This is only one aspect of the bin choice. It is also possible to change the number of bits allotted to each component of the color space individually in order to best take advantage of the properties of the color space. The object is to tweak the color space enough to make it robust against small lighting changes, while not neglecting significant differences in shade. The expectation was that a better result might be achieved by using a color space such as HSI and allocating a large number of bits to hue, while decreasing the number of bit for saturation and intensity. It will be seen later that empirical evidence bears out that assumption.

3.5 Real World Data

Unfortunately for computer vision researchers, real data is seldom as clear and as clean as one might hope. There are shadows, highlights and other effects that clutter the color space of an object. If the object is out of door, the matter only becomes worse. Now there are moving clouds and swaying trees that may cast a shadow, or worse, partially obscure the object in question. A typical color histogram of an object really only occupies a certain predictable portion of the color space. Depending on

the position of the sun, a large fraction of the pixels considered as part of the image may be part of the shadow, rather than the object. This results in low intensity bins that are fuller than they should be, as the shadow pixels tend to fall in these bins. There also tends to be a smaller grouping near pure white, as a result of reflection. The rest of the colors occupy a football shape between these two points. It is very rare to see a pixel that is, for example, pure red. Colors tend to have low saturation. Having this insight, adjustments can be made to ensure that the correct features are being matched, rather than other less interesting features.

One corrective method has already been discussed; tweaking the color space. However, this can only do so much. It will not eliminate the clumps at the pole that increase the tendency to match objects based on the size of their shadows and reflections than their actual color content. There are a few obvious way to approach this problem. The first is to simple neglect the effects of the high and low ends. In other words, throw out all pixels with intensity of less then a lower bound or greater than a higher bound. The obvious problem here is that very light or very dark objects may not have any data at all. A less drastic approach is to weight the bins, so that bins with greater saturation, for example, have greater importance. This seems like an interesting approach, but it is difficult to pick the bin weights. An interesting approach would be to use an evolutionary algorithm to determine bin weights. The problem is, there is no clear “goodness” function to optimize towards.

It is possible to take advantage of the fact that the time over which the objects are matched is relatively limited. Given this fact and since each set of objects to be compared has already been classified based on the basis of size and aspect ratio, in can be assumed their shadows will be roughly the same size. If all the histograms to be compared are averaged, this average will approximate the effect of shadows, along with any other invariants between histograms. If we subtract this average from each histogram, the result is the distance from the invariant. Some of the bins may be negative, but this is not a problem in determining Euclidean distance.

3.6 Conclusion

The final histogram generation portion of the system employs basic traditional methods, and allows the user to set many of the options, including a choice of color spaces, a variable number of bins and a configurable bit distribution. Histograms are taken across sequences of time linked image chips. As the processing will proceed in a batch-wise fashion, an average over all histograms in a batch is used for normalization to minimize the effect of invariants such as shadow.

Chapter 4

Clustering

This chapter will lay out the design of the prototype histogram creation system. It will begin with a discussion of the algorithm choice. Next it will discuss the distance metric used, and the reasons for its selection, followed by the algorithm itself. Finally practical code and I/O issues will be addressed.

4.1 Why Min-Cut?

This issue was dealt with to some extent in the background section, so here is a recap: The recursive Min-Cut algorithm is good because it is exact. Its performance, at $O(n^3)$, is not very good. Fortunately, it is good enough. A half hour was chosen as a reasonable period of time, as it is not long enough for significant lighting changes, but is still long enough to gather a fair amount of data. At peak times, around 100-200 recognizable cars pass the cameras. The current implementation can extract histograms and cluster this amount of data in around fifteen minutes. There are faster versions of min-cut, but most of them depend on a sparse graph, as opposed to the fully connected graph here. A possible future implementation using an optimized min-cut will be discussed in the future work section. For the time being $O(n^3)$ will have to suffice.

The associative variant of min-cut as presented by Shi and Malik[6] has a single but important advantage over traditional min-cut. Traditional min-cut, especially

in a fully connected graph such as the one being considered, will tend to make cuts that separate only a small part of the graph. This makes sense, as a single node in a fully connected graph is connected to all other nodes by n arcs. To cut the graph in two would require the deletion of $n^2/4$ arcs. In order to correct for this tendency, instead of minimizing the cut value, we minimize the value of the cut by a measure of associativity, in other words, how closely connected the resulting subsets are.

4.2 The Algorithm

The objective of this algorithm is to find the minimum *normalized cut* ($Ncut$). Recall that the *cut* of a graph is the sum of all arcs that connect two disjoint subsets, A and B of the graph $G = (V, E)$. It is defined as:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v). \quad (4.1)$$

The other quantity that will be used is the *associativity*, which is a measure of the connections between the nodes in A to all nodes in the graph. It is defined as:

$$asso(A, V) = \sum_{u \in A, t \in V} w(u, t). \quad (4.2)$$

The normalized cut is then:

$$Ncut(A, B) = \frac{cut(A, B)}{asso(A, V)} + \frac{cut(A, B)}{asso(B, V)} \quad (4.3)$$

A cut that separates a single node from the rest of the graph will now have a large $Ncut$ value, as its associativity will equal the cut value. It does preserve the desirable properties of the original cut formulation. In fact, if the associativities are equal, the $Ncut$ value will equal $2 * cut$.

Rather than reiterate the derivation and correctness argument from Shi and Malik [6], we will simply enumerate the steps of the algorithm. First, here are some of the terms used in the algorithm. Let W be the node-node matrix, such that $W(i, j) =$

$w_{i,j}$. The sum of all connections for a node i to all other nodes is $d(i) = \sum_j w(i, j)$. The matrix D is a diagonal matrix with d on its diagonal.

1. Create W and D .
2. Solve $(D - W)x = \lambda Dx$ for the second smallest eigenvector.
3. Find the splitting point such that this eigenvector partitions in the graph such the $Ncut$ is minimized.
4. If the $Ncut$ is below a pre-specified value, recurse over the new subsets.

Step one sets up the operation of the algorithm. Steps two and three actually find the minimum $Ncut$. The third step may not be entirely clear. After the completion of the second step, we are left with the second smallest eigenvector, λ . We try l evenly spaced possible splitting points. For each splitting point p , if $\lambda_n > p$ then n is a member of A , otherwise it is a member of B . The p that results in the lowest $Ncut$ is the one that is used.

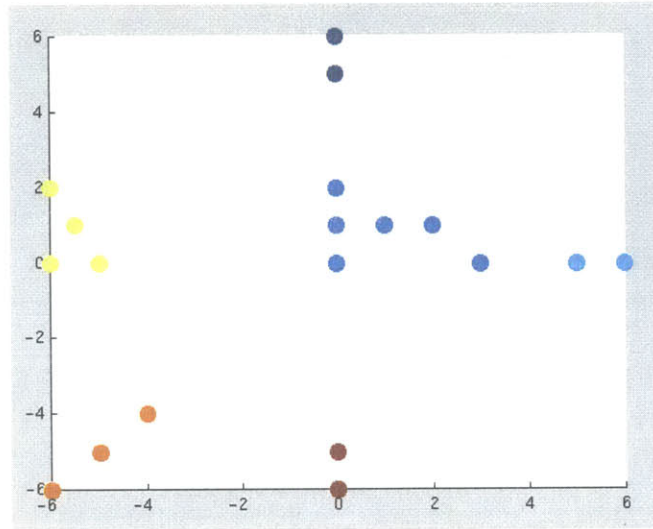
4.2.1 A 2-D example

The best way to show the operation of this algorithm is with points on a plane. The example in Figure 4-1 shows a set of points in the plane that have been grouped by the algorithm. It can be seen that the result is quite intuitive.

4.2.2 Generalized to N Dimensions

Unlike some clustering algorithms, the recursive min-cut algorithm works in arbitrarily dimensioned spaces. This is fortunate, since the dimensionality of the problem being considered is equal to the number of bins the histogram, which could potentially be quite large. All that is needed is a measure of similarity between any two nodes in the graph, and this is easily done.

Figure 4-1: Clustering on the plane



4.3 The Distance Metric

The first step in the algorithm is to create a node-node matrix of a quantity denoting the level of similarity of any two given nodes. There are two obvious color histogram comparison metrics. One is the city block method used by Swain and Ballard[7]. The other is Euclidean distance. The “city block method” chooses adds the minimum of each pair of bins to create the distance metric. Euclidean distance sums the squared differences of each pair of bins. The results of the two are usually fairly similar. Euclidean distance has the drawback of being slower, but has the virtue of demonstrably working for negative values.

The difficult is that when finding the minimum cut, the arc weights are assumed to signify degree of similarity, rather than distance. There are a few possible ways to remedy this situation. One is to recast the minimum cut problem to a maximum cut problem. Another is to use some conversion from distance to similarity. The two methods should be equivalent. Since this discovery was made after the min-cut code had been written, the second method was chosen.

The distance between two histograms, as they are initially generated, has a minimum of zero and a maximum of one. The most obvious way to make this into a

similarity metric it to subtract it from one. The subtraction of an average histogram changes the nature of the histogram a bit, and causes situations where the metric might exceed these bounds. Since Euclidean distance is being used, the distance is by definition positive. It is the other bound that is troublesome. Consider a set of ten two-bin histograms. Nine of them have a zero in the first bin and a one in the second. The tenth has a one in the first bin and a zero in the second. The average is then 0.1 in the first, and 0.9 in the second. After normalization, the nine contain -0.1 and 0.1, while the tenth contains 0.9 and -0.9. The Euclidean distance between the two is 1.28. It is true that for the vast majority of cases the distance will still be less than one, but a negative similarity measure should not be allowed to slip past. There are number of ways to correct for this. One is to simply threshold the distance at 1. Another is to try another metric. The first method seems by far the simpler, and was therefor the one chosen.

4.4 Outputting the Data

Since the algorithm does not create the hierarchy in the form of a data structure, but rather simply finds the result recursively, the options for output are limited. The approach chosen was to output nodes to a file as they were created. This was done in such a way that it would be easy to parse up into a tree structure, where it can then be thresholded and output. The next chapter will deal with this final stage of the implementation.

Chapter 5

Thresholding and Output

The clustering algorithm has output a file that specifies the hierarchical classification of the objects. This is not, however the end. It is still necessary to use this classification to create groups and then use these groups to create prototype histograms.

The initial approach involved directly thresholding the hierarchy as it was produced. The matlab code that generated the hierarchy would also threshold it, and output the clustering. This method was judged inefficient when many thresholds are to be tested. Given this method the hierarchy must be generated every time a new threshold is to be tested. Generating the hierarchy is also the slowest step in the process. If the threshold has already been decided, this method would be slightly faster, but in the testing phase it is better to have more flexibility.

The new method was to have the clustering algorithm generate the full hierarchy, and then output the results to a file which could then be parsed and thresholded by another program, which could then produce output. A novel thresholding method that requires no externally set value was implemented, and will be described.

5.1 Getting the Data Back

There are two main components to the process of moving from a hierarchy description file to a point where the hierarchy can be thresholded and output. The first is parsing the file, and the second is creating the data structure. Both must take place

simultaneously, but since the design decisions involved can be made separately, they will be described separately.

5.1.1 Parsing the File

The hierarchy file was created as the result of a recursive process that proceeds in the same order as a depth first search. As a result, the parsing process should also proceed as a depth first search. It creates a new node for the first batch of data it encounters. Every subsequent datum it encounters becomes a child of the previous, until a node that contains a single object is parsed. Once such a node has been parsed, it is necessary to backtrack and find the next position where a node should be added. If the node is a left child, this is an easy matter. The next node is its parent's right child. If the node is a right child, the process is similar. The algorithm searches up the tree until it reaches a node that is a left child. If this search reaches the root node, then the tree is complete. The new node is that node's right sibling. This ordering can also be obtained by maintaining a stack of the next nodes to be added. When a node is created, its children are pushed on the stack. When a new node is needed, it is popped off the top of the stack. When the stack is empty the search is finished.

5.1.2 The Data Structure

There is a rather clever method of fitting a binary tree into a linear array. The first node is placed, intuitively, at the first position. The index of a node's right child is $2n$ and its left child is $2n + 1$. The index of a node's parent is $\lfloor n/2 \rfloor$. For a balanced binary tree, this is dynamite. It is intuitive and easy to maintain, as well as space efficient. Unfortunately, the classification is not a balanced binary tree. In fact, in the worst case, each cut may separate a single node, making the storage demands of this method grow to an unpleasant 2^n . Another method was needed. Each node is assigned the next free index upon creation. The node must now contain the indices of its parent and children. This method is not quite as elegant, but ensures that only

the amount of space that is really needed is used.

Each node represents a new cut in the hierarchy. As a result, there is a certain amount of data that it must store. First, it contains the unique ID numbers (uids) of the objects whose histograms are at that level of the hierarchy. It also contains the value of its minimum cut. In addition to this set of nodes, there is another data structure that contains the information about the final clustering.

5.2 Thresholding

In order to complete the clustering, a threshold value must be imposed on the hierarchy. The easiest way to do this is to mimic the way the hierarchy was generated. A recursive approach was used to inspect the nodes in a depth first manner. When a node with a cut value above the threshold was encountered, the index of that node was added to an array which will hold all clustering data. When the process is finished, the array will contain the indexes of the nodes that would have been leaf nodes, given that threshold value. In order to create the clusterings, it is only necessary to pick the uid lists out of the nodes specified in the array. From there it is an easy matter to reload the histogram data and generate average histograms over each cluster.

When this method was tested, the results were less than optimal. In general the hierarchy would look good, but the thresholding would tend to give clusters to each brightly colored object, while clumping large groups of fairly dissimilar but less colorful objects together. This was initially thought to be a problem with the choice of color spaces, it was later concluded that a large part of it was actually the thresholding.

The problem with the obvious method of thresholding is that it works from the top down, and never even sees the cut values near the leaves of the tree. There may be a cut that is well below the threshold that will never be made, as nodes above it may have been just above the threshold. Lowering threshold is not an effective solution, as this may cause bad cuts near the nodes to be made.

The solution is to work from the bottom up, rather than the top down. A cut is

made when it is smaller than the average of the two minimum cuts below it. If a cut is larger than the average of the minimum cuts below it, the cut is not made, or to look at it the other way around, the uids of the nodes below it are merged. If a cut has been made, the uids on either side of the cut may not be merged with any other nodes. One complication is the root nodes themselves. There can be no cut value for them, as each contains only a single uid. In most cases it will be beneficial to merge single uids with each-other, or with larger clusters, so they are set a low but arbitrary value of 0.75. Changing the parameter affects the performance of the clustering to a degree, but not nearly as much as the old threshold. Here is the real value of this technique: No threshold value is required. The heuristic will come up with a good clustering with no outside tweaking.

Implementing this heuristic required some good recursive thinking. The idea is to determine which nodes satisfy the property described above, and then choose only the ones that have no children with the property as clusters. This algorithm was implemented in two stages. First, a depth first search was used to mark the nodes that had the property. Then a recursive procedure was used to find the lowest possible nodes that had no nodes with the property beneath them. Testing all nodes beneath a node was also implemented recursively. The theoretical performance of this algorithm is unspeakable bad, but the real running time was barely noticeable, so it was let pass. The algorithm could be re-implemented to run in $O(n)$ time.

5.3 Conclusion

Most of the mechanisms described in this chapter are central to the conceptual basis of the system. They were put in place for support and efficiency. They support the segmented nature of the system, and they provide for more efficient testing. As will be seen in the following chapter, they will also allow the construction of interesting visualizations. The one exception to this is the new thresholding procedure. It is a new method that, as will be seen later, is quite effective.

Chapter 6

Visualization and Other Side Issues

In the process of creating this system, it was necessary to create a fair number of visualization tools. In order to be able to understand and verify the correctness of the process, it is necessary to be able to look inside and see what is going on. The main data item used was the color histogram, and as a result it is the most desirable to visualize. Unfortunately, this is not an easy task. In this chapter, two visualizations will be presented. Next, in order to judge the effectiveness of the algorithm it is necessary to be able to look at its output. The various methods used will be discussed.

6.1 Color Histogram Visualization

A color histogram is three dimensional in nature. Whether those axes are red, green and blue or hue, saturation and intensity is immaterial. Each histogram bin corresponds to a particular point or region in that three dimensional space. Naturally there have been other attempts to visualize color histograms[5]. One simple method used is to create a two dimensional plot of each axis. Unfortunately, this loses quite a bit of the data, and does not create an intuitive picture of the histogram. In fact, the chances of creating a good static two-dimensional representation of this three

dimensional space seem poor. Instead, a dynamic, three-dimensional approach is needed.

6.1.1 A Three-Dimensional Approach

Fortunately, there is a great deal of software for creating and displaying three-dimensional objects. The software chosen for the task was SGI's Open Inventor. The Inventor file format provides an easy way to create files that can be automatically generated and then viewed.

The difficulty is no longer generating and viewing a three dimensional visualization, but deciding the form that the visualization will take. The method chosen was to place a sphere at each point that corresponds to a histogram bin. Each sphere would have the color of its bin, and its size would correspond to the contents of the bin. There are a few apparent problems with this approach. First, is it not possible for spheres to completely occlude each other. The answer, is surprisingly, no. The fact that the contents of the bins must add up to one allows the histogram to be sufficiently sparse that it is always possible to get a good idea of the nature of the histogram.

A few modifications were made to this method, in order to make it easier to view and more accurate. Initially the radius of the sphere was in proportion to the corresponding bin's contents. This is misleading, as volume proceeds with the cube of the radius, and the contents of a sphere with twice the radius will be seen to have three times the volume. The final version has the volume of the sphere proceed in proportion to the contents of the bin.

The histograms tend to be so sparse that not only is occlusion no a problem, but the sparse nature of the histogram makes it more difficult to understand. In order to alleviate this problem, the histogram are scaled so that the largest bin is the maximum size possible. This makes occlusion more likely, but makes things easier in the balance.

Figure 6-1: Histogram visualization

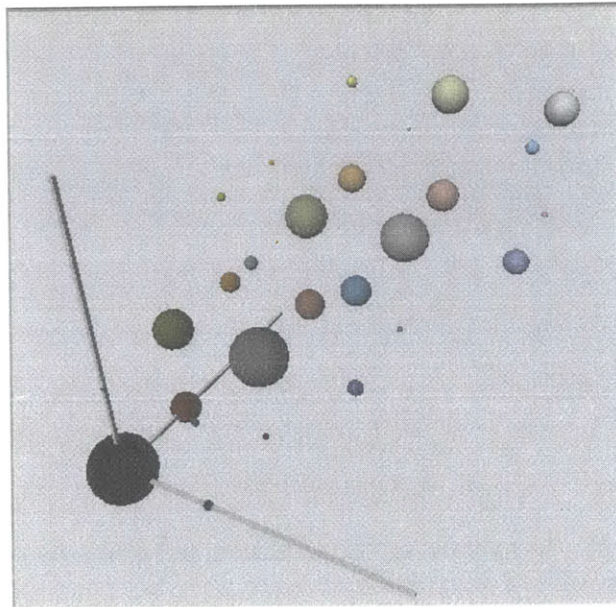
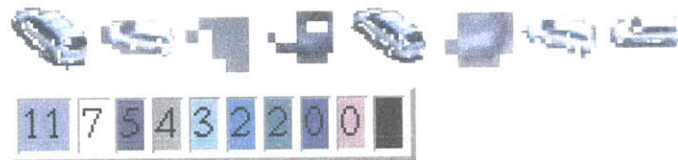


Figure 6-2: A car clustering and its 2D histogram



6.1.2 A Two Dimensional Approach

While a color 3-D model is nice, it would also be nice to have a more compact 2-D representation. This was achieved by creating a table of the ten most common colors, along with their frequency percentage. This provides a quite compact and useful representation. If the number of bins in the histogram increases we may simple expand, for example, to the top hundred.

6.2 Visualizing the Hierarchy











In the previous chapter, the method that was used to create a data structure that represents the clustering hierarchy was described. When the hierarchy was being created and thresholded recursively, it was next to impossible to visualize. Now that the hierarchy is contained in a data structure, it becomes a much easier matter.

The way a binary tree should be visualized is fairly straight forward. At the top there is the root node, and the tree proceeds down the page by levels from there. At each vertical position there should be nodes that are only a fixed distance from the root. The way to produce this effect for display is with a breadth first search. Each visualization was created as a webpage, whose html was generated automatically by the program. The display method was chosen as it is relatively easy to generate, organize and store. It also has the benefit of being a ubiquitous standard. Now, typesetting and positioning graphics is not html's strong suit, but it does have a very useful construct, the table. This structure was used for reliable positioning.

Breadth first search ordering is achieved by maintaining a queue of nodes to be inspected. Any given step is achieved by taking the first node in the queue, inspecting it, and adding its children to the back of the queue. In this context, the nodes of each level should between them contain all of the uids that were clustered. This fact was used to determine whether a level had been completed. For the first node of a row, a column element was created with the width of the number of uids contained by that node. The cut value of that node is also placed in the space. This process is repeated until all uids have been seen and the row is complete. An additional complication is that in order to keep the cuts lined up it is necessary to figure out when a node has only a single uid left, and reserve that space for the remaining rows, as that branch of the tree will not be pursued any further. An array is kept of where each uid was found in its own node.

When the entire tree has been explored, the array of single uid nodes is full. In the last row of the table contains images that represent each uid, in the order left in the array. Now at any level of the tree it is possible to look down and see which

Figure 6-3: A sample hierarchy

1.254876							1.256432		
1.214522			1.253424				1.280414		
1.293562			1.290802		1.206370				
									

objects were grouped together at that division. These images were created by taking the first image chip off of the top of each objects sequence. This usually provides a decent idea of what the object looks like. A happy accident occurred at this point. The alpha channel of the images were being used to indicate which pixels were part of the image, and this was interpreted by the web browser as gif transparency. As a result, only the pixels that were part of the object are displayed, making recognizing them quite a bit simpler. An example of the result is shown in Figure 6-3. The first cut is made between the first seven objects and the last three. The value of the first cut is not shown, as the first cut is always made. The values of the second level cuts can be seen in the top level of the table. The first frame for each object is shown in the last row.

6.3 Visualizing the Clusters

This was not a difficult matter. Each cluster is a row of the page, that contains the images for the objects in that cluster, as well as a 2-D version of the color histogram. While simple, this proved to be a very useful visualization. Often, the hierarchy would look good, but the actual clustering would turn out not to be as good.

6.4 Conclusion

These were visualization tools created out of necessity for this system. Some of them are perhaps only useful in this context, such as the hierarchy visualization. Others, such as the 3-D histogram visualization might prove useful to others trying to compare color histograms.

Chapter 7

Results

Once the system was completed it became time to test it on real data. The objective of the testing was two-fold. The first objective was to see if the system would perform in a satisfactory manner over a variety of data. The second objective was to try to determine which color space and bin choice performed best, and whether this was independent of the data. The results were quite interesting, and bring some surprising insights.

7.1 The Data

There were two primary sets of data used in the testing. The first was the traffic captured by the Tech Square cameras described earlier. The second was a few hours of video capturing the movements of three cats in a constantly lit but highly shadowed, indoor environment. The two data sets are quite different. The first is composed of many small, essentially single colored objects, that cannot be individually recognized. The goal for this data set is merely to get a picture of the types of cars in the scene, not to identify any particular car. The second set of data contains a small number large, irregularly colored objects, that can be individually recognized. In the context of the first data set, a single color approximation might suffice, but in the second it would not. Two of the cats have similar average colors, but their color histograms should differ. The objective for this data set is to see if the system can effectively

Figure 7-1: A sample car sequence



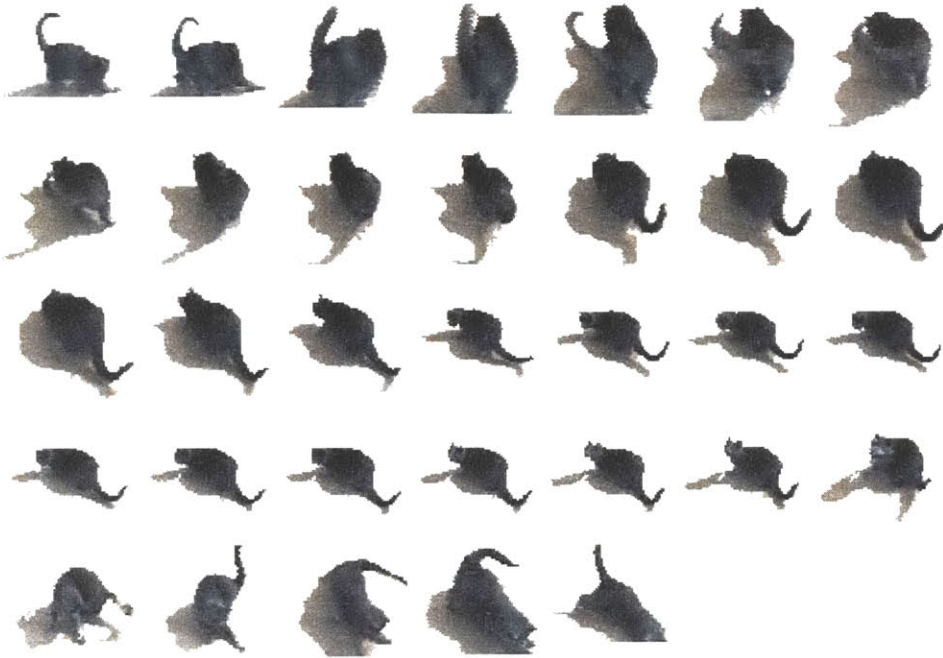
distinguish the three cats.

The main obstacle to effective testing was the lack of a good way to judge success. Especially with the cars there are many possible “correct” ways to cluster, and not all human observers might agree on all clusterings. With the cats things are a bit better, as there are three cats, each of which appears a fixed number of times.

7.2 Cars

Two sets of data were actually used for the car tests. The first was the result of a real half hour of tracking, including some extraneous people, and a less diverse set of cars. The second was an artificially constructed data set, that contained no people, and was chosen to have a wide range of colors of cars. Even this set, however, was drawn from a single hour of tracking, and not unrealistic. There were thirty-three cars of many different colors in this data set. A typical series of image chips for a white car are shown in Figure 7-1. While most were white, black, grey or some shade of blue, there were two that were orange, probably cabs or service vehicles of some kind, and four were various shades of red. While most configurations could distinguish black, grey and white, it would be these colored cars that would prove to be the real test of a configuration.

Figure 7-2: A sample cat sequence



7.3 Cats

Since the size of the data set was relatively small, all of the data was used. There were twenty-seven tracks of three cats. One cat, who is solid grey, was responsible for twenty of those tracks. A typical series of image chips for the grey cat are shown in Figure 7-2. The second cat is a dark calico and was responsible for five of the tracks. The final cat, who is primarily white, was responsible for the final two tracks. The expectations for this data set were not particularly high. It was not easy for even a human observer to distinguish the two darker cats, given the video resolution. The primary test of a given configuration will be if the two white cat tracks are put together. Of course, this would happen by accident about 1/25th of the time, but if it happens often for a particular group of configurations, it will probably be significant.

7.4 System Components

In the previous section, quite a number of system parameters were described, and quite a number of them were used in testing. In all, two color spaces, 5 different bin numbers and 63 different bin distributions were tested. The effects of normalization and min-cut parameters were also tested. The results were somewhat surprising, but not unpleasant.

7.4.1 Color Spaces

The two color spaces used in testing were RGB and HSI. They were chosen as representative of two classes of color space. One is based on cylindrical coordinates, where hue is the angle, saturation is the radius and intensity is the distance. There are several other similar color spaces. The other is based on rectangular coordinates, as is CMY. The expectation was that in contexts where each component was represented by equal numbers of bits there should be little difference between the two. It was also expected that, with control of the bin distributions, HSI could do a better job. The balance of its components could effect performance.

7.4.2 Bin Distribution

This section was the real surprise. As it turned out, nothing had as great of an effect on the success or failure of a particular trail than the number of histogram bins. Distribution could effect things to a degree, but the addition or subtraction of a single bit could have profound effects of performance. The optimal number of bits seems to be a function of the number of pixels in the track.

For the car sample used, the average number of pixels was around fifteen thousand pixels, or the size of an image 122 pixels on a side. This is just a bit larger than the average SGI desktop icon. Even a frame from a CCD camera is around 300,00 pixels. The bottom line is that there is just not that much data, and if the color space is too large, the data will be spread too thinly in the bins to generate a good match. With 14 bits of color it would be possible to put only a single pixel in each bin. The

cat data was a bit better, averaging closer to 100,00 pixels per track. This was due to the fact that the image chips were normalized to 64x64 instead of 32x32, and the cats tended to stay in the scene much longer than the cars. This was reflected in the performance of various bin distributions.

In the absence of a hard objective function, the following heuristic was used. A hierarchy that places the two yellow cars and the four obviously red cars together and has no obvious wrong placements is said to be a “good” hierarchy. While it is possible for this situation to arise by chance, it is quite unlikely. A hierarchy that clusters three of the four red cars, or has an obviously misplaced object is said to be “fair”. Hierarchies with many errors, but still some perceivable structure are “poor”. All others are “random.”

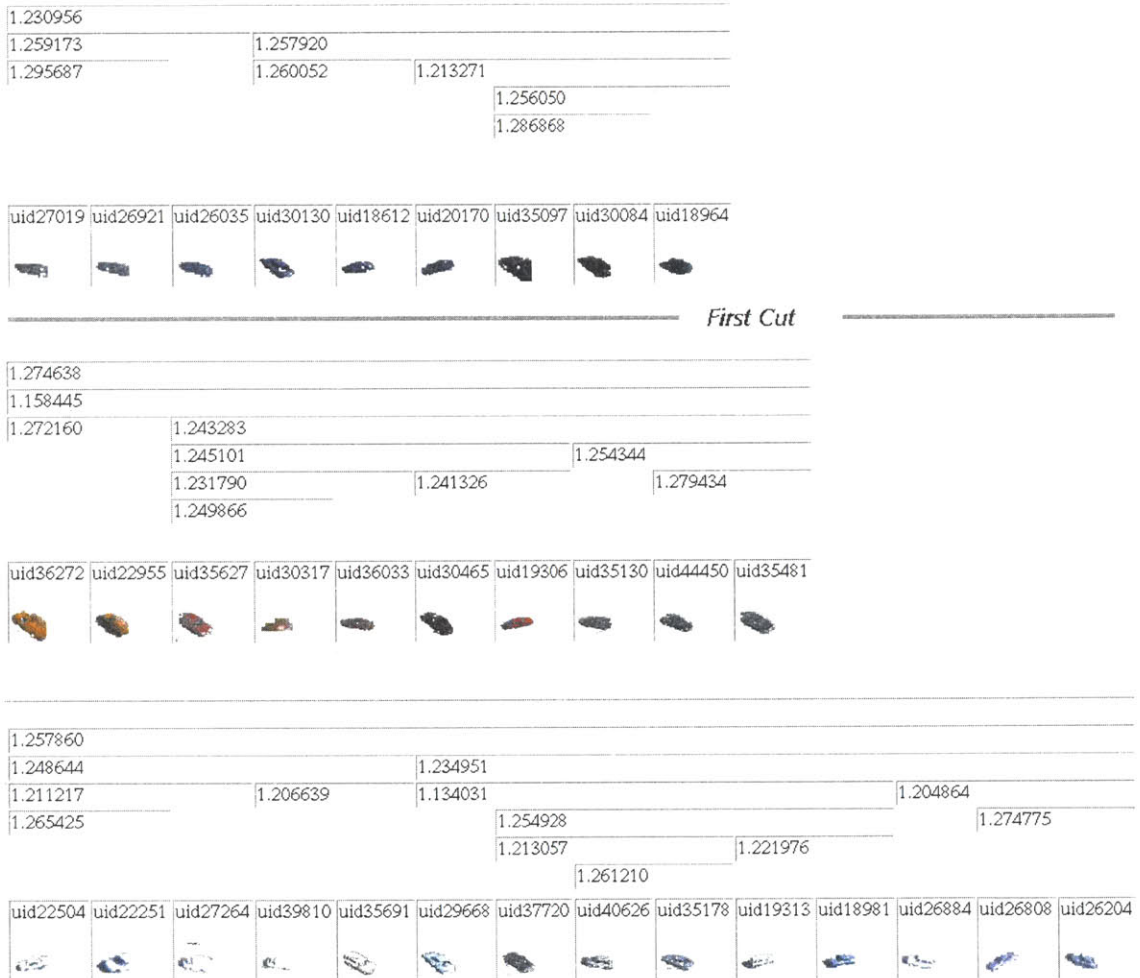
Using HSI, every possible distribution of bin that totaled four or more bits, with each component having minimum of one and a maximum of four bits assigned to it was tested.

Of the possible four bit and five bit histograms in HSI, all were fair. This was much better than expected, given only 16 and 32 bins, respectively. Even more amazing was continued fair performance given only one bit of hue. It is difficult to see, at first, how this is possible. Of the six bit histograms, all were fair, except one, three bits of hue, two bits of saturation and one bit of intensity, which was good, and is shown in Figure 7-3. Of the seven, all were poor, except one fair. Beyond seven was a mixture of poors and randoms. Almost every configuration, however, paired the two yellow cars together.

Using RGB, the evenly distributed configurations with 3, 6, 9 and 12 bits respectively, were tested. The three and six bit configuration had fair performance, while the nine was poor, and the 12 was random. Several variations of bin weightings were also tried, with results that were similar, but slightly poorer than those of the equivalently binned HSI configurations. The six bit version is shown in Figure 7-4.

The cats were tested using the same configurations as the cars. Here, the test of histogram “goodness” was if the two tracks of the white cat were paired. Grouping of the calico would be regarded as excellent, but was not expected. The results were

Figure 7-3: The min-cut hierarchy for HSI 321









1.186056		
1.232223	1.141272	
	1.263046	
	1.267815	1.249518
		1.269575

uid29810	uid27264	uid22504	uid29668	uid35591	uid22251	uid26884	uid26808
							












First Cut

1.224822	
1.236164	
1.274446	1.256891
1.236966	1.250166
1.251514	

uid35097	uid20084	uid44450	uid30465	uid19306	uid20170	uid18964
						

Second Level Cut

1.283417		
1.255030		
1.195934	1.270415	
1.247576	1.187546	1.220781
1.300975	1.263970	1.244365
	1.264001	

uid27019	uid26921	uid26035	uid35178	uid40826	uid37720	uid35130	uid35481	uid36033	uid35627	uid30317
										

Third Level Cut

1.189260		
1.260875	1.247081	
	1.135808	1.209957
	1.236645	

uid36272	uid22955	uid19313	uid30130	uid18612	uid25204	uid18981
						

similar to those for the cars, only shifted up. Six bit color spaces were good about half the time, where as seven bits was good 10/12 times. Eight bits was good only 2/12, and beyond that there were no good hierarchies. No configuration grouped more than three calico tracks together. Figure 7-2 shows one of the better six bit clusterings. It is difficult to tell, but both figures in cluster two are of the white cat.

7.4.3 Normalization

A few test sets were run with and without normalization in order to determine whether this step was having a positive effect. The results lead to the conclusion that normalization in this context is at best marginally useful. The un-normalized results looked very similar to the normalized ones. In a context with a better defined fitness function it might be easier to determine whether or not this is a beneficial step to take.

7.4.4 Thresholding

As was described in an earlier chapter, the initial thresholding method had a tendency to give colorful objects their own clusters. A partial sample is shown in Figure 7-6. The white and black cars are clustered together, while each red and yellow car is given its own cluster. On the other hand, the new bottom up thresholding approach worked quite well. The problem of splintering color clusters was eliminated, as is seen in Figure 7-7. Of course, these are examples chosen to accentuate the advantage of the new method. In most cases, results for the two methods were similar. The result of a full half hour's clustering is shown in Figure 7-9. In this case the times are from 3:00 to 3:30PM. There are 129 objects tracked, and they are divided into 35 groups, or around four objects per group. These groups are small, but fairly accurate. Note how cluster two contains what appears to be all tree-based interference. It has distinguished the color of the cabs in cluster 3 from those in clusters 11 and 12. All reddish cars are in cluster 13. There are several different shades of white, black, blue and grey that have been identified. The only obvious stand-out is the cab at the end

Figure 7-5: A clustering of the cat data HSI 231

Cluster number 1



Cluster number 2



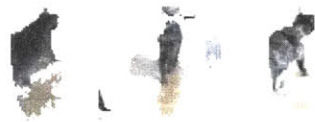
Cluster number 3



Cluster number 4



Cluster number 5



Cluster number 6



Cluster number 7



Cluster number 8



of cluster 15, which is otherwise made up of dark grey cars. This may be explained by heavy shadowing over part of the track. Shown in Figure 7-10 is a clustering taken before sunrise. Headlights of slightly different colors are clustered separately, and artifacts are grouped into a third cluster.

7.5 Conclusion

The conclusions drawn from these tests is that the relationship of the number of histogram bins to image pixels is of great importance. It is also concluded that while unevenly distributed HSI may be better than RGB, it is not remarkably so. The benefits of normalization were also not clear from the tests run here. Finally, it may also be concluded that the thresholding technique does in fact function as it was postulated that it would.

Figure 7-6: Clustering with old thresholding method.

Cluster number 5



Cluster number 6



Cluster number 7



Cluster number 11



Cluster number 12



Cluster number 13



Cluster number 14



Cluster number 16



Cluster number 17



Cluster number 18



Cluster number 19



Cluster number 31



Cluster number 35



Cluster number 36



Figure 7-7: Clustering with new thresholding method.

Cluster number 1



Cluster number 3



Cluster number 5



Cluster number 7



Cluster number 8



Cluster number 9



Cluster number 10



Cluster number 13



Cluster number 14



Figure 7-8: Clustering of the test data.

Cluster number 1



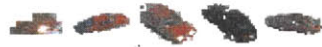
Cluster number 2



Cluster number 3



Cluster number 4



Cluster number 5



Cluster number 6



Figure 7-9: A clustering of a half hour of real data.



Figure 7-10: A clustering of headlights.

Cluster number 1



Cluster number 2



Cluster number 3



Chapter 8

Conclusion

This chapter will be a summary of the goals, implementation and results of this project, as well as ideas for possible future work.

8.1 Future Work

There are a number of features and abilities that either time did not permit, or were not conceptualized until late in the project.

8.1.1 Other Applications

It would be possible to adapt this method to the classification of other attributes. For example, consider the problem of matching the paths objects take across the scene. Instead of a collection of histograms, imagine a series of positions along the path an object takes. Two paths could then be compared by summing the Euclidean distance between the two corresponding points in each path. The number of points must be a constant, so for objects that have a greater or lesser number of frames than the number of points, which will be practically all of them, the point values will be approximated.

8.1.2 Histograms

As was discussed in the section color histograms, it seems as though some bins have a greater importance for a match than others, and it would be nice to be able to evolve a system that would take advantage of this property. It is possible to imagine a neural network or genetic algorithm approach that could evolve the appropriate weights. In order to perform this sort of reinforced learning, it is necessary to have some target solution that the output of the system is matched against. While this would not be easy given the primary data set used in this context, it is easy to imagine a data set where a match was more meaningful. A that remains is some metric to compare the clustering of a particular histogram weighting with the target. What made this idea impractical in the context of this thesis was the long cycle time. Trying to run a genetic algorithm when the generation time is up to half an hour is a depressing prospect. Given faster hardware, a smaller data set and a distributed approach, this approach might be realistic.

8.1.3 Clustering

Given the results presented, it is clear that while the current clustering algorithm is good, it is not perfect. There may be incremental approaches that yield results that are as good or better, but have better running times and do not suffer from the other drawbacks of a batch based algorithm. As was stated earlier, the batch based algorithm side-steps the issue of changing global lighting. It might be possible to address this concern, which would be a large one given in incremental algorithm. One possibility is normalize the intensity of an object given the average intensity of the scene at that time. This would require more data to be stored, but might to some extent correct the problem. Shadowing issues might be addressed in a way that is similar to the current method. Since there is no batch to normalize against, a histogram could be normalized with respect to the n histograms that come before and after it in time. The value of n should be chosen so that there is enough data to produce a good invariant, but not enough to have adverse effect on the running time

of the algorithm.

8.1.4 Visualization

While the visualizations of the color histograms proved adequate, the cluster and hierarchy visualizations had on notable drawback. Since these charts simply pulled of the first frame of any given track, it is sometimes difficult to see the whole story. For example, the entire object might not be in the frame. The front and back of an object may look quite different, or the object might at some point pass into deep shadow. It would be nice to somehow get a better picture of the entire track. One possibility is, instead of pulling of the first frame, the system could search through the entire track to find the frame that has a color histogram that is them most similar to the global color histogram of the track. This would be a bit computationally expensive, but would eliminate some of the problem. Another method would be to create a form of output that can display all of the frames, one after another. One easy option would be to create animated gif of the sequence, rather than a single animated gif. This would look rather obnoxious in the context of a webpage, but might provide a better view of what was going on. With a little javascript magic, it might be possible to animate a particular sequence only on mouse-over.

8.2 Summary

Here is a re-cap of the design goals set at the outset of the project. Having now seen the system in its entirety, it can now be analyzed to see if it meets the initial criteria, and in what ways it can be improved.

- It should run with minimal, if any, human input.
- It should be robust to lighting changes.
- It should run in real time.

- The effects of artifacts of artifacts of the underlying vision system should be minimal.
- It should perform in a way that corresponds closely to human intuition.

The first was good performance in the absence of human adjustment. When the initial thresholding method was being used, it could be argued that the presence of the min-cut threshold value violates this criteria, and it would be better to move to a system that requires no such parameter. With the new thresholding system in place, the only parameter that is required is the “cut value” of a single object, which is, as discussed before, a much less important value than the old threshold value. While with the old threshold method it might be advantageous to change the threshold depending on the color space or the size of the data set, this value may be set once and will function well in a variety of circumstances.

The second is that ambient lighting changes should be of negligible importance. Of course, it is impossible to completely satisfy this requirement. At some point in a natural setting it becomes too dark to distinguish color accurately. It is, however, possible to do a reasonably good job at it. The current system effectively cancels the effects of some lighting effects, such as shadowing and glare, but does not address global lighting changes. Instead, the data is divided up in such a way that global lighting changes have little effect. It could then be argued that the system does not live up to its design goal.

The third is that it should run in real time, and it clearly lives up to this goal. In the worst case it is possible that at peak hours the number of objects will outpace the hardware’s processing capacity, but this turns out not to be much of a problem. As the load returns to low levels, the slack will be picked up, and at some point the processing will catch back up with the data.

As to artifacts of the underlying system, most often when people or other artifacts were present in the data, the results were not effected. Usually people wearing light clothes would end up clustered with light colored cars. Sometimes, if there were multiple people, they would be placed in their own cluster. This is probably due to

the percentage of shadow in a person's track.

The last, and most important goal was is that the results should correspond closely with human intuition. To the great relief of all involved, the system did perform well in this context. The results detailed in the previous chapter paint a good picture of the strengths and limitations of the classification system.

Bibliography

- [1] W. P. Berriss and S. J. Sangwine. Colour histogram clustering technique for tissue analysis of healing skin wounds. In *International Conference on Image Processing and its Applications*, pages 693–697, Dublin, Irel, July 1997. IEE.
- [2] W.E.L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *CVPR*, 1998.
- [3] R. M. Haralick and G. L. Kelly. Pattern recognition with measurement space and spacial clustering for multiple images. *Proc IEEE*, 57:654–665, 1969.
- [4] Robert K. McConnell Jr and Henry H. Blau Jr. Color classification of non-uniform baked and roasted foods. In *FPAC IV*, Chicago, Illinois, November 1995. Food and Process Engineering Institute A Unit of ASAE.
- [5] Paula J. Reitan. 3d visualization of color image histograms. *Computer Networks and ISDN Systems*, 30:20–21, 1998.
- [6] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *Conf. on Comp Vision and Pattern Recognition*, San Juan, Puerto Rico, June 1997. IEEE.
- [7] Michael J. Swain and Dana H. Ballard. Indexing via color histogram. In *Proc. 3 Int Conf Comput Vision*, pages 390–393, Piscataway, NJ, December 1990. IEEE.
- [8] J. Z. Wang, Jia Li, and Gio Wiederhold. System for screening objectionable images. *Computer Communications*, 21(15):1355–60, October 1998.