An Investigation into Segmenting Traffic Images Using Various Types of Graph Cuts

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Jonathan Dinger
August 2011

Accepted by:
Dr. Stan Birchfield, Committee Chair
Dr. Ian Walker
Dr. Timothy Burg

Abstract

In computer vision, graph cuts are a way of segmenting an image into multiple areas. Graphs are built using one node for each pixel in the image combined with two extra nodes, known as the source and the sink. Each node is connected to several other nodes using edges, and each edge has a specific weight. Using different weighting schemes, different segmentations can be performed based on the properties used to create the weights. The cuts themselves are performed using an implementation of a solution to the maximum flow problem, which is then changed into a minimum cut according to the max-flow/min-cut theorem.

In this thesis, several types of graph cuts are investigated with the intent to use one of them to segment traffic images. Each of these variations of graph cut is explained in detail and compared to the others. Then, one is chosen to be used to detect traffic. Several weighting schemes based on grayscale value differences, pixel variances, and mean pixel values from the test footage are presented to allow for the segmentation of video footage into vehicles and backgrounds using graph cuts. Our method of segmenting traffic images via graph cuts is then tested on several videos of traffic in various lighting conditions and locations. Finally, we compare our proposed method to a similarly performing method: background subtraction.

Dedication

I dedicate this thesis to my parents, Dennis and Christine, and the rest of my family, whose support has kept me going throughout my college career.

Acknowledgments

I would like to thank my advisor, Dr. Stan Birchfield, for his help with this thesis. His support, guidance, and encouragement is what has allowed me to complete it. I would also like to thank Dr. Timothy Burg and Dr. Ian Walker for being on my thesis committee.

I would also like to thank Dr. Ali Kemal Sinop and Dr. Leo Grady for taking the time to answer some of the questions I had about banded graph cuts.

I would also like to thank my father, Dennis, for helping to proofread this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we describe a method for segmenting traffic images using graph cuts. Traffic detection is a problem that people have been working on for years, using different methods, including electrical induction loops. The ability to detect and measure traffic is valuable for planning purposes. Systems that can count vehicles and measure congestion allow for better predictions about the amount of wear on roads, leading to better scheduling of maintenance as well as roadway upgrades.

Computer vision is one method that has been used to try and solve this problem. Cameras are cheap and easy to install. Traffic does not need to be disrupted to install the cameras. Computer vision is capable of more than measuring numbers of vehicles. It can be used to measure direction and speed, as well as detecting things like traffic accidents. There are problems with using computer vision methods to measure traffic, however. The cameras must be networked to retrieve the video at a central location. Large amounts of video need to be analyzed and stored. Computer vision methods need to be robust enough to deal with camera problems; lighting conditions and precipitation should not affect the operation of the system very much. Other problems also crop up, such as occlusion. Depending on the camera angle, vehicles can be obscured by other vehicles. Even if a vehicle is partially visible behind another, it is very easy for a computer vision system to detect them as one vehicle instead of two.

Many techniques have been used to attempt a solution to the problem of traffic detection using computer vision, such as background subtraction and feature tracking. An overview of some recent papers on traffic detection using computer vision can be found in chapter 2.

Many traffic detection algorithms start with step such as background subtraction. With

minor cleanup, background subtraction allows for foreground information to be segmented out of the image. This may include traffic, shadows, pedestrians, animals, and so on. Once the background areas are filtered out, later steps in the traffic detection algorithms can work on extracting more information from the foreground areas. One of the main problems with background subtraction is that it computes a label for each pixel individually. In images, large sections tend to be very similar with respect to certain properties, including texture, lighting, and color. By using some of this information, an improvement upon background subtraction can be made.

In this thesis, we propose to use graph cuts to segment traffic images, instead of background subtraction. There are multiple types of graph cuts that we can use to solve this problem. We compare and contrast several different types of graph cuts to determine which should be used to segment traffic images. Graph cuts are multi-purpose. By changing the weighting in the graphs to use different image properties, such as color and texture, different segmentations can be achieved. We propose several weighting systems to allow for segmentation of footage into vehicles and backgrounds. In these weighting systems, we make use of a background image, allowing the foreground to be separated from the background. Then, we combine those weights with a separate set of weights that use grayscale values from our images to keep similarly shaded sections of an image in the same segment. Finally, we compare our results to background subtraction to validate our proposed method.

1.1 Outline of the Thesis

In chapter 2, we discuss various methods that have been used to track traffic. In chapter 3, we explain graph cuts and much of the terminology used when discussing them. We then explain in detail the weighting systems that we use to segment traffic images using graph cuts. Chapter 4 contains the discussion of our experimental results, including both image data and timing results. Chapter 5 contains a summary of the thesis and possible routes to explore in future work.

Chapter 2

Related Work

Saunier and Sayed [14] describe an algorithm for using features to track vehicles in intersections. Tracking vehicles in intersections can be more challenging than most highway tracking. Highways tend to be laid out in a straight line, more or less, and they usually have two entry and exit points. Traffic usually travels from one entry/exit to the other in a fairly straight line, allowing a simpler tracking algorithm to be used. Most intersections have several entry/exit points, and traffic may not travel in a straight line from one to another. Vehicles may turn to head down another road. This complicates things, as now feature tracking algorithms may not see roughly the same features throughout a vehicle's time on-camera. For example, features may initially be tracked on the front of a car, and as it turns, those features are lost in favor of features on its side.

First, a homography is computed to give a transform between world coordinates and image coordinates. Distances in world coordinates are used to group and segment vehicles, so image coordinates are not good enough. Using a Kanade-Lucas-Tomasi feature tracker [3], features are tracked through every frame in the video. Measuring with world coordinates, if features are close enough together for a long enough time, they are considered connected. This groups together features that are close to each other, while keeping features that are far apart (different cars separated by a distance, for example) disconnected. Over time, the maximum and minimum distances between each connected pair of features is computed. If the difference between the maximum and minimum distances is too large, that pair of features is disconnected. This means that if two cars are initially close together and then move apart, the two cars will be disconnected and tracked separately. Each set of interconnected features is identified as a separate vehicle. When all features in a set are no

longer in-frame, statistics are computed, such as speed and direction. Due to tracking features over time and disconnecting features that do not move together, this algorithm can deal with traffic congestion and turning vehicles.

Hsieh et al. [8] describe an algorithm that detects and tracks vehicles on roadways, while extracting from the data such information as lane-dividing lines, vehicle types, mean speed, and the amount of traffic. Using background subtraction and minor noise removing operations, vehicles and their shadows can be detected. A histogram of vehicle movement over time is then created by tracking many vehicles. After running several operations on the histogram, including smoothing, averaging, and merging operations, centers of each lane are found. Then, based on the lane centers, all of the lane dividing lines except for the outermost lines are found by splitting the difference between two adjacent centers. Lane width is also computed from the lane centers. The actual outermost lane dividing lines are found using the outermost dividing lines found thus far and the lane width.

Then, they describe how to remove shadows by using the lane dividing lines. This helps in situations where shadows connect two vehicles together. If those shadows are removed, the vehicles will segment into two separate vehicles correctly. Any given lane dividing line, which may be slightly curved, can be approximated using a straight line. That straight line is then moved side to side by changing the y-axis intercept point of the approximating equation. At all points on the current line, pixels are examined to see if they are likely to be a shadow. If, at all points on the current line, the pixels are likely to be shadow pixels, they are marked as such and removed from the segmentation. This will usually separate vehicles in adjacent lanes that are connected by shadows. Using a similar method with horizontal lines, instead of lines based on the lane dividing lines, more shadows can be removed. Finally, by examining a histogram of segmentation boundary pixels projected onto the y-axis, lines can be found that separate vehicles that are close together.

Based on the segmentations described above, the size (normalized based on the lane width) and the "linearity" (how close to a straight line one of the side edges of the vehicle is) of each vehicle are computed. These characteristics are then compared to a library of vehicle templates to determine what class of vehicle each segmentation should fall in. In short, they describe a full system that is able to detect vehicles, remove shadows, and classify each vehicle moving on roadways.

Ki and Lee [10] describe a method for recording accidents that occur in intersections. The first step in their algorithm is to detect vehicles while they move through intersections. Instead of

background subtraction, they take the difference between two consecutive frames and then threshold that to find areas of motion. They calculate the expected direction of motion from the second frame where they detect a vehicle, and then project that onto the first frame. Using the vehicle image they acquired from the second frame, they attempt to match the vehicle to an area close to the estimated projection in the first frame, while minimizing error. This gives them a vehicle location and segmentation in two frames. Using those two frames, they then calculate the expected velocity of the vehicle. The velocity, combined with the direction of motion, then allows easy matching with later frames. By updating the velocity and direction frame by frame, vehicles are tracked throughout their entire time on camera. This tracking then gives the trajectory of each vehicle.

The second step in their algorithm involves accident detection. Their method is based around the idea that in general, vehicles in an intersection will be consistent in their movement. There should not be any irregular changes in direction, size of the vehicle, speed, and position. Changes in these properties will occur, but if they are irregular, there is a high chance that an accident had occurred. For instance, if two cars move across an intersection and one crashes into the other, they will usually stop. The variance of each of these properties is projected onto a scale that goes from zero to one. While it may be possible that an accident occurred when only one of these properties varied irregularly, it is much more likely for an accident to have occurred if multiple properties changed. It is quite possible that a vehicle would swerve to avoid a collision, causing a large variation in direction, while still crashing afterward, causing a large variation in both speed and position. All four of those scales are summed and then thresholded to determine if an accident occurred.

Alessandretti et al. [1] describe a method by which vehicles and guard rails can be detected using radar and vision data. Unlike the other methods mentioned previously, this system does not make use of a static camera. The radar system is mounted above the front bumper of a vehicle, and the camera is mounted near the rearview mirror. This tracking system is not meant for measuring statistics on a certain patch of road, but actually for detection and classification of obstacles while driving along a roadway.

First, the radar is used to find areas of interest around the vehicle. These areas may include guard rails, vehicles, or other obstacles. Once objects are found using the radar, the locations are mapped from the radar coordinate system to the image coordinate system using a transform computed when the radar and camera are calibrated. Each radar object detected is surrounded by a

bounding box, creating an area of interest that could contain vehicles or other objects. It is desired that the system tracks vehicles, so other objects must be filtered out. Therefore, guard rail detection was implemented to run on the images.

The speed of an object, as measured by the radar, must be below a certain threshold for the guard rail detector to be applied. A Sobel filter is applied to the image in any areas of interest, which are then examined to find edges that are roughly oriented in the direction that guard rails are expected to be oriented. Starting at the vertical boundaries of the image, on the lower half, the algorithm searches for an edge pixel that has not been filtered out by the previous operations. A search is then run, attempting to link that initial pixel to other pixels in a continuous line, creating a likely guard rail. Due to how the radar works, any guard rails may actually be detected multiple times. In general, however, this creates overlapping areas of interest that can be filtered out, assuming that the first area is identified as a guard rail. Because guard rail detection is extremely fast, this filters out many areas of interest that are not vehicles very quickly, accelerating the vehicle detection process.

Any area of interest that contains an object that is faster than a certain threshold or classified as "not a guard rail" is then run through the vehicle detection algorithm. A Sobel filter is applied during this detector as well, leaving edges that are oriented vertically on the image. A symmetry calculation is run over the entire area of interest at various bounding box widths, checking if there are matching sets of vertical edges around a central axis. Any sets of symmetrical edges that are not far enough apart are discarded, as they may be a part of the vehicle instead of the whole. This symmetry calculation then provides the width of the vehicle as well as the center axis. The bottom of the bounding box to go around each vehicle is found by looking for horizontal edges on the bottom half of the area of interest. The top half of the box may or may not be found, in which case, the algorithm guesses based on the width of the vehicle. Basic filters, using such properties as sizes and locations of the bounding boxes, are used to discard and merge detections, leaving only the vehicles driving on the road.

In short, the system uses radar data to find areas of interest. The guard rail detector, which is relatively fast, is used to filter out large numbers of those areas, allowing the vehicle detector to be run on possible vehicles only. This increases the speed at which the entire algorithm can run.

Kim [11] describes a method of tracking objects using features and background subtraction. These two methods are used to complement each other and make the algorithm more accurate

overall. Background subtraction is performed frame by frame, with each successive background frame computed from the previous background frame and the difference between the current frame and the previous background frame. However, a major problem in using background subtraction has to do with sudden illumination changes. These can occur whenever a camera with an auto-iris is used to record the footage.

Therefore, they suggest a method for using recent image frames to correct sudden illumination changes. Using recent images, they run a computation over all the pixels in those images, computing a correction factor for each color separately. Due to the use of recent frames, the sudden illumination change gets averaged out in the correction factors. These correction factors are then applied to the background image when it gets updated. Segments of the image are then found to be part of the foreground after subtracting the background, and can thus be tracked.

They find features located in the foreground sections only. They then compare these features to features on the background images, discarding any matches. These features are tracked through subsequent frames, as well. However, there are usually multiple features in each object that is being tracked, so the features must be grouped together. First, features are grouped into spherical clusters, using such properties as position, trajectory, and previous grouping. Then, the spherical clusters are grouped into ellipsoid objects based on their 3D trajectories in world coordinates. This two-phase grouping method works for vehicles, bicycles, and pedestrians.

Kanhere and Birchfield [9] describe a method to track vehicles at relatively low camera angles using features. At low angles, different problems arise from those faced by using a high-angled camera, such as occlusion and drastic changes in object size. At this angle, a homography calculation is not enough to specify the feature locations in world coordinates, so another method is proposed.

First, the camera is calibrated using user-defined marks. Both sides of the road are marked, as well as a line crossing the direction of traffic. From these three markings, a full perspective mapping from 3D to 2D can be created. Next, the background of the image is found offline by averaging image frames over a set amount of time. By subtracting the background image from the current image, a foreground mask can be created. Feature points are found using the Kanade-Lucas-Tomasi feature tracker. This returns features on the background, in vehicle shadows, and on moving vehicles. However, the foreground mask can be used as a filter. Any features not in the foreground mask or that are within a certain distance from edges shared with the background are discarded,

leaving only features that are on the vehicles.

Using a plumb line projection, the world coordinates of each feature can be found. Then, each feature is classified as either a stable feature or unstable feature. Stable features are low to the ground and are on the front of a given vehicle. Finally, features are grouped together to form vehicles. Stable features are grouped together. The basis for their grouping is that features on the front of a vehicle should be on the same plane perpendicular to the direction of travel. If the stable features extend sideways more than the width of a lane, however, the extra features are split off into a second grouping.

Each of the unstable features is then attached to a group of stable features using a score based on plumb line projections and similarity of motion, where unstable features are more likely to be attached to larger groupings (larger vehicles). Classification of feature groups as either trucks or cars is then performed. Vehicles with a larger number of unstable features over time are more likely to be a truck, as trucks are larger, longer, and taller. Classification is performed on that basis.

Chapter 3

Research Design and Methods

This chapter discusses our algorithm for the segmentation of traffic images using graph cuts. First, we give a brief overview of graph cuts and the terminology involved. Next, we look at a normal graph cut as well as two different variants, in order to discover what the best type of graph cut would be to use for segmenting traffic images. We then discuss practical issues that arise when implementing the max-flow/min-cut algorithm. The final type of graph cut that we examine is the multi-way cut, which is required to segment images into more than two labels. The multi-way cut would be useful for including shadow detection in future additions to our work. After all that, we discuss the implementation of our algorithm for traffic detection as well as the derivation of all parameters used.

3.1 Graphs

A graph G is a set of vertices V (alternatively referred to as nodes) and a set of edges E that connect them together. This will be denoted $G = (V, E)$. For our purposes, let s and t be two of the vertices in V, called the source and the sink, respectively. These two vertices are also known as the terminals. Let M be the remaining vertices. Then, $V = M \cup \{s, t\}$. Each vertex in M is connected to all neighboring vertices in M by a bi-directional edge called an n-link (neighbor link). Each vertex in M is also connected to each of the terminals by uni-directional edges called t-links (terminal links). One set of t-links goes from the source vertex to all vertices in M, and one set of t-links goes from all vertices in M to the sink. In a weighted graph, each edge has a non-negative

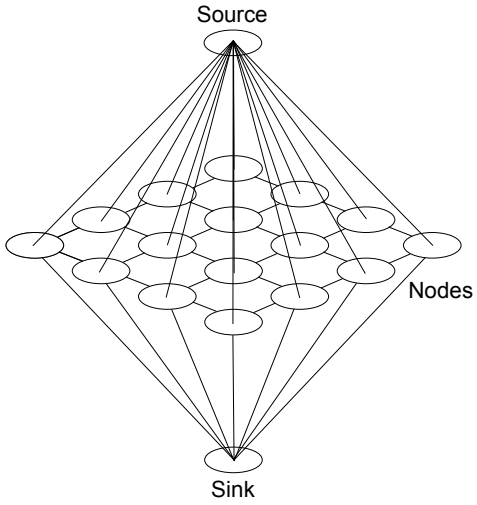


Figure 3.1: An example of a 4x4 graph.

weight, also called a capacity.

To segment an image, each pixel in the image has its own vertex. These vertices form the group of vertices M. Each vertex in M is connected to each neighboring vertex by an n-link. The neighborhood of a vertex is based on the neighborhood of that vertex's corresponding pixel. Each vertex in M can be connected to either four neighbors or eight neighbors, depending on the connectedness of the graph. Each vertex in M is then connected to both the source and sink vertices via t-links. An example of a two dimensional image graph of this type can be seen in Figure 3.1.

Three dimensional graphs of this type can be created as well. The main lattice of vertices would be three dimensional and would use either six neighbors or twenty-six neighbors for each vertex depending on the connectedness of the graph. Each vertex would then have an edge connecting it to the source and sink.

Each edge in E has a capacity and at least one direction. N-links are bi-directional, while t-links are not. The capacity of an edge is based on the measurement used to create the graph. The capacity of an edge is the amount of flow that can go through that edge. For bi-directional edges, flow can go in either direction. For uni-directional edges, flow can only go in one direction. An edge is considered to be saturated if the amount of flow is equal to the capacity.

3.2 Cuts and Flows

A path from a vertex u to a vertex u' is defined as a sequence of vertices $(v_0, ..., v_k)$ such that $v_0 = u$, $v_k = u'$, and each edge $(v_{i-1}, v_i) \in E$. The length of a path is simply the number of vertices in the path. An s-t cut, hereafter referred to simply as a cut, is a set of edges C, that if removed, will leave each vertex in M with a path to either the source or the sink, but not both. A minimum cut is simply a cut with the minimum possible cost. That is, the sum of all capacities of the removed edges is the minimum possible to achieve the goal of separating the set of vertices M as mentioned above.

Since each edge has a capacity, flow can be driven from the source to the sink. The maximum flow through a graph is simply the maximum possible amount of flow that can be forced from the source to the sink. The max-flow min-cut theorem states that the maximum flow through a graph is equal to the minimum capacity of a cut. The cost of a cut is the sum of the capacities of all the edges contained in the cut.

Thus, if the maximum flow through a graph is found, a certain number of edges will be saturated. If these saturated edges are removed from the graph, a minimum cut has been found.

3.3 Graph Cuts

Graph cuts are simply a minimum cut on a given graph. This cut segments images into two regions. Because the minimum cut is equivalent to the maximum flow, many graph cut algorithms actually compute the maximum flow instead of the minimum cut.

One way to do this is to use the augmenting paths algorithm [7]. An augmenting path is simply a path, in this case from the source to the sink, that increases the total flow through the graph when flow runs through it. The residual capacity of an edge is the capacity minus the flow running through it. All edges along any augmenting path must have a positive residual capacity.

Let us use an example involving water running from a source, through a network of pipes (edges), to a drain (a sink). We want to increase the amount of water running through those pipes until there is a bottleneck somewhere that stops us from increasing the amount of water. We have to search through the system of pipes looking for a path that goes all the way from the source to the sink where all pipes in the path have some space left to push more water through (positive residual capacity). If we cannot find a path from the source to the sink that has a positive residual capacity

through all the pipes in the path, that means that there is a bottleneck, and we cannot increase the flow any more. If the flow cannot be increased, that means that the maximum flow has been found.

When creating the graph, both bi-directional edges and uni-directional edges are created using two edges. For a bi-directional edge, each edge simply has the same capacity. Thus, flow can go in either direction. For a uni-directional edge, one edge has a positive capacity, and the other edge has a capacity of zero. This way, flow can only go in one direction. However, an edge with capacity zero and negative flow has a positive residual capacity. This means that running flow through an augmenting path can lessen the flow through a uni-directional edge.

To go back to our water-in-pipes example, a bi-directional pipe means that water can be pushed in either direction through that pipe. If the flow of water is going in one direction, that means that there is a positive flow in that direction. For the same pipe, there is a negative flow in the other direction. A uni-directional pipe means that flow can be pushed through in one direction, as the other direction has zero capacity. If flow is pushed through in the direction that flow is allowed, that means that there is a negative flow in the other direction. A negative flow with zero capacity still causes a positive residual capacity. This means that after water is pushed through a uni-directional pipe, paths can then be found going the opposite direction in the pipe if those paths are augmenting and will thus increase overall flow. Since the capacity in the opposite direction is zero, this means that flow in the correct direction through the uni-directional pipe will decrease. It does not mean that flow will actually go in the opposite direction. Decreasing flow through a uni-directional pipe allows for the algorithm to continually find better paths, even if it already saturated some pipes with water.

A search for an augmenting path is run on the graph. When an augmenting path is found, the following steps take place.

1. The minimum residual capacity along the path is found.
2. Flow equal to the minimum residual capacity along the network is added to every edge in the path.
3. Flow equal to the minimum residual capacity along the network is subtracted from the reverse of every edge in the path.

To again go back to our pipes example, the steps of the above list occur when a series of pipes from the source to the sink has been found where each pipe still has room for more water to

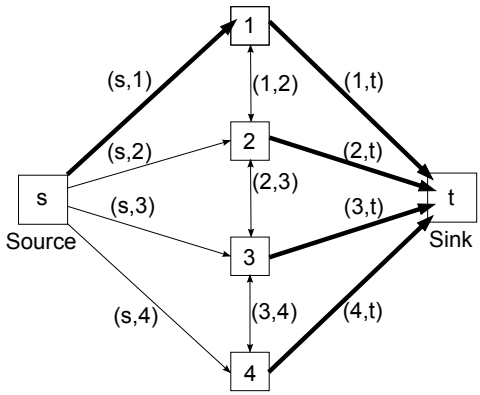


Figure 3.2: A simple example of finding an augmenting path.

be pushed through it. Step 1 means that the series of pipes should be examined until one is found that has the minimum possible flow that can still be added to it. Step 2 means that now that the bottleneck has been found, we push enough water through the pipe to fill the bottleneck. Since the pipes are all modeled as bi-directional pipes, flow in one direction is negative flow in the other direction. Step 3 says that since we push flow through in one direction, we must subtract the same amount of flow in the opposite direction to keep balance in the network. Now that the bottleneck pipe is full, another augmenting path must be found.

Different versions of the augmenting paths algorithm exist. The difference between them is how the augmenting path is found. If the path is found using a breadth-first search, the algorithm is known as the Edmonds-Karp algorithm. If the path is found using a depth-first search, the algorithm is known as the Ford-Fulkerson algorithm. A comprehensive search must be run over every possible path in the graph, starting with every edge leading from the source. However, there are certain things to be avoided here. If a search is run, starting with all edges leading from the source, it is possible that the search algorithm will search edges that have already been searched fruitlessly.

Figure 3.2 shows a small example of a graph. We use it to illustrate one problem with finding an augmenting path. It shows six vertices: the source (s), the sink (t), and vertices 1-4. The arrows indicate edges, with uni-directional edges marked at one end and bi-directional edges marked at both ends. The edges are marked as (v_i, v_j), where v_i and v_j are the vertices that a given edge links together. Let the bold arrows be edges that have been saturated with flow. If a search for an augmenting path is run on this graph, it may start at the source vertex s and examine edge $(s, 1)$. Edge $(s, 1)$ is already saturated, so the search moves on to edge $(s, 2)$. Edge $(s, 2)$ is

unsaturated, so vertex 2 is added to the path. The algorithm then examines edge $(1, 2)$. Edge $(1, 2)$ is also unsaturated, so vertex 1 is added to the path. However, edge $(1, t)$ is saturated, so the algorithm has nowhere to go. Vertex 1 is removed from the path and the algorithm continues searching. The algorithm finds that there is no augmenting path starting with the smaller path $(s, 2)$. Vertex 2 is removed from the path and vertex 3 is added, since edge $(s, 3)$ is unsaturated. A totally exhaustive search could then examine edge $(2, 3)$, find that it is unsaturated, and add vertex 2 to the path. However, the algorithm has already searched all possible paths leading from vertex 2, with no augmenting paths found. Researching the paths leading from vertex 2 wastes time, and should be avoided. This can be avoided by marking each vertex as "checked" as the algorithm searches past and around that vertex. If the algorithm then comes across a vertex that has already been checked, it does not need to waste time researching all the edges connected to that vertex, because all those edges have already been searched.

Another way of speeding up the max-flow algorithm is to start by finding minimum length augmenting paths and then increasing the path size. In Figure 3.2 and all the graphs we deal with in this paper, the minimum path length from the source vertex to the sink vertex is 3. Let us examine Figure 3.2 and assume that all edges are unsaturated. If the search starts by finding all paths of minimum length, it starts by finding the paths $(s, 1, t)$, $(s, 2, t)$, $(s, 3, t)$, and $(s, 4, t)$. When the search algorithm finds these paths, the flow through them is then augmented. Each path of length 3 has two edges connecting its vertices. One of the edges in each pair is saturated, immediately cutting down on the possible paths that can be found afterward. Then, all paths of length 4 are found, followed by those of length 5, 6, etc. This cuts down on the time spent finding the maximum flow immensely.

Another difficulty in implementing this algorithm is actually taking the maximum flow of a graph and converting that to a minimum cut. Essentially, any edges that are totally saturated with flow are removed from the graph to create a minimum cut. That means that any vertex still connected to the source or the sink via an unsaturated path is considered a source or sink vertex, respectively, and is labeled accordingly. The vertices may not be directly connected to the source or the sink, so another search may need to be run. However, if the augmenting paths software is written as described above, with each vertex "checked" as the augmenting paths algorithm searches past it, the final state of the graph when an augmenting path cannot be found can be used to label the vertices. The augmenting paths algorithm was looking for an augmenting path from the source

to the sink. Therefore, the final state of the graph will have any vertices that are connected to the source via an unsaturated path "checked," because the algorithm went over them. If the vertex is "checked," mark it as a source vertex. There is a special case in which all edges connected to a vertex are saturated, leaving no unsaturated paths to either the source or the sink. In this case, the vertex will not be "checked," and will be considered a sink vertex. This is somewhat arbitrary, since vertices which fall into this case could technically be either source or sink vertices.

Pseudo-code for using the graph cut algorithm in binary image segmentation is as follows:

1. Choose the weighting scheme to be used.
2. Loop through the image. For each applicable pixel:
   (a) Add a node to the graph.
   (b) Calculate the weights for the t-links.
   (c) Calculate the weights for the n-links.
   (d) Add edges for each t-link and n-link with capacity equal to the previously calculated weights.
3. Calculate the min-cut/max-flow of the graph.
4. Use the results as needed.

Graph cuts are discussed in detail in [4, 5, 6].

3.4 Alternate Techniques

The standard graph cut algorithm can be used in many applications. It segments the entire image all at once. However, in some applications, such as large three dimensional volumes, a faster algorithm is needed.

3.4.1 Banded Graph Cut

The banded graph cut [13] is one such faster algorithm, when run on a set of data that warrants its use. The banded graph cut is based on the idea of segmenting multiple smaller versions of the image. Instead of running a graph cut on the entire full-size image, the segmentation is run on

(a) Downsampled image

(b) Initial segmentation

(c) Initial segmentation border

(d) Expanded border

(e) Upsampled border

(f) Larger image

(g) Upsampled segmentation

(h) Final cut

Figure 3.3: Steps in a banded graph cut.

a relatively small area, comparatively. This allows for significant speed improvements on the actual segmentation.

Figure 3.3 shows a series of images illustrating many of the steps in a banded graph cut, where each image is at the correct relative scale compared to the others. In a banded graph cut, we set a number of extra levels. Then, we downsample our starting image that number of times. For instance, if we use 2 extra levels, we then have a series of three differently sized images. Figure 3.3a shows a sample image downsampled twice that is 1/16 the original image size. Then, we use a graph cut, as discussed in section 3.3, to segment the smallest image. The results of this segmentation can be seen in Figure 3.3b, where white pixels are one label, and black pixels are another label.

We then find the border of this small segmentation, as can be seen in Figure 3.3c. The border is expanded (Figure 3.3d) and then upsampled (Figure 3.3e). This enlarged border is the "band" that is referred to by "banded" graph cuts. We then make use of the next largest version of our original image, which is 1/4 the original image size. This image can be seen in Figure 3.3f. This is a downsampled version of the original image, not an upsampled version of the smaller image shown in Figure 3.3a, meaning that it has more detail than the image in Figure 3.3a.

Another graph cut is run on this larger image in Figure 3.3f, where the only parts of the image that are included in the graph are pixels that have a corresponding white pixel in Figure 3.3e. The only parts that are re-segmented are those that are in the "band." The initial segmentation, from Figure 3.3b, is upsampled. The results of that operation can be seen in Figure 3.3g. The graph cut results in the upscaled segmentation (Figure 3.3g) are now merged with the results from performing a graph cut on our larger image (Figure 3.3f) in the areas inside the band (Figure 3.3e). The graph cut on the band pixels is considered to be more reliable, due to being run on an image with more detail. Therefore, the results from this graph cut overwrite the previous results during the merge. The only changes to the segmentation image of Figure 3.3g that occur during the merge should happen around the boundaries between white and black pixels. The final segmentation with the merged information can be seen in Figure 3.3h. It can be seen that the segmentation border is much smoother in Figure 3.3h than in Figure 3.3g.

The process then loops back to the step in Figure 3.3c, except with the larger segmentation that we just obtained. This process repeats until the final segmentation (the step illustrated in Figure 3.3h) is full size.

Pseudo-code for the banded graph cuts algorithm is as follows:

1. Downsample and store the images for the chosen number of extra levels.
2. For the bottom level image:
    (a) Run a full graph cut.
3. For each level of image:
    (a) Using the resulting binary segmented image, find the border of the segmentation.
    (b) Upsample both the segmented image and the segmentation border.
    (c) Dilate the segmentation border.

(d) Run a graph cut on the expanded segmentation border.

(e) Using the results of the graph cut, modify the upsampled binary segmented image. The only pixels affected should be inside the band.

4. At this point, the segmented binary image should be full resolution.

3.4.2 Banded Graph Cut with Laplacian Augmentation

One of the problems with with the banded graph cut is that when images are downsampled, detail is lost, especially in long narrow areas. One way to get around that is to augment the band using data from a Laplacian pyramid [15].

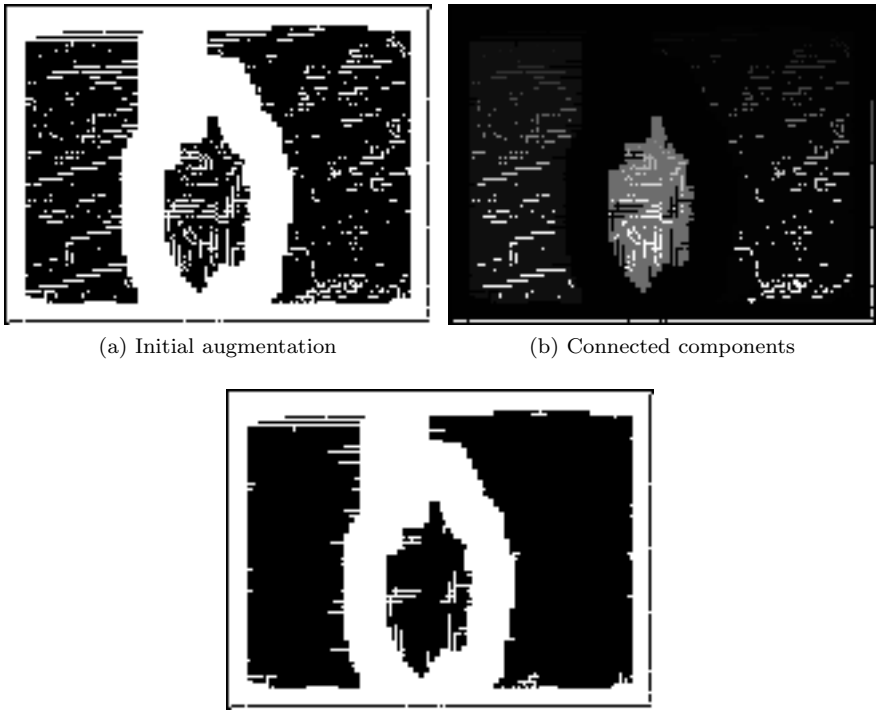
This algorithm is exactly the same as the banded graph cut above, except it augments the band that is to be re-segmented after each upscaling operation. The augmentation of the banded graph cut makes use of the image details that are lost during a downsampling operation. For each k^{th} level of image I^k, a second image S^k can be found using the following operations

$$S^k = I^k - U(D(I^k)) \tag{3.1}$$

where U and D in Equation 3.1 are upsample and downsample operations, respectively. This operation of downsampling, upsampling, and subtracting from the original serves to save all the information lost during the downsample operation.

Figure 3.4 shows a series of steps that the augmented banded graph cut performs that the normal banded graph cut does not. The steps in this image would occur right after the step shown in Figure 3.3e and will actually replace that result with another. The only change to the banded graph cut algorithm is a slight change to Figure 3.3e.

This image S^k is used to modify the band Q^k (Figure 3.3e) by using a threshold α. Any pixel in S^k whose absolute value is over the threshold α is added into the set of band pixels Q^k to create the augmented band Q^{k*}. The augmented band Q^{k*} can be seen in Figure 3.4a. A connected components algorithm is run on the augmented Q^{k*}, as seen in Figure 3.4b. Any pixels in the augmented Q^{k*} that are not in the same connected component as the original Q^k are then removed from Q^{k*}. If we examine Figure 3.4b, each different shade of gray is a different unconnected piece of the image. The large black band in the middle of the image and around the edges are all the



(a) Initial augmentation

(b) Connected components

(c) Merged augmentation

Figure 3.4: Extra steps for the Laplacian augmentation.

areas in the same connected component as the original band Q^k. Any white pixels in Figure 3.4a that are not also in the large black connected component in Figure 3.4b are removed from Figure 3.4a, resulting in Figure 3.4c. This result is then substituted for the image in Figure 3.3e. Thus, any detail over a certain threshold that was removed during the downsampling operation, that is connected to the main band, will also get segmented. This serves to check for narrow structures obtruding from the main segmentation area that would have been lost in the original banded graph cut.

3.4.3 Multi-way Graph Cut

The graph cut algorithm by itself can only be used to do binary segmentation. To use three or more labels, the algorithm must be modified. There are many ways to do a multi-way cut, but the algorithm described here is the $\alpha - \beta$ swap algorithm.

In this algorithm, the graph cut algorithm is run multiple times, once for each pair of labels. However, only pixels with one of the current pair of labels α and β get segmented, while all other pixels are ignored. Some pixels labeled with α will swap to being labeled with β and vice versa. All other labels stay the same.

For instance, if there are three labels 0, 1, and 2, a graph cut will be performed for the pairs $(0, 1)$, $(0, 2)$, and $(1, 2)$. By starting with an arbitrary labeling, all the possible swaps are covered.

This entire process can be repeated multiple times to ensure that the current labeling has the best fit.

Pseudo-code for the multi-way graph cut is as follows:

1. Arbitrarily label all pixels in the image.
2. Loop through the number of desired cycles for each cut.
    (a) Loop through each pair of labels.
        i. Add a node to the graph for each pixel with one of the current pair of labels.
        ii. Set up edges for each node.
        iii. Compute the maximum flow of the graph.
        iv. Use the segmentation to relabel all pixels with labels that changed.

Our implementation of the multi-way cut is based on the α-β swap algorithm discussed by Boykov et al. in [6].

3.5 Traffic Segmentation Using Graph Cuts

To segment traffic from its surroundings in a video sequence, we perform a graph cut on each frame separately. If we can find a background image for the section of road we are observing, then we can base the weights used for the t-links for the graph cut off of the background image. We will discuss two separate sets of t-link weights, one using exponentials and one using grayscale value differences.

We first compute the background image using the average of the image frames in our image sequence. While we compute the background image, we also compute the variance of each pixel. The variance will be used in the t-link weights, and this is the best place to do the calculations. The variance σ^2 is computed for each pixel in the image sequence using the equation

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{3.2}$$

where N is the number of images in the sequence, x_i is the grayscale value of the pixel in the current image i, and μ is the average grayscale value defined by the equation

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.3}$$

where N, i, and x_i are as above. The values of μ from Equation 3.3 over all pixels in the image form the background image.

Next, we perform a binary graph cut on each frame separately using the weights explained in sections 3.5.1 and 3.5.2, where the pixel is only added to the graph if its variance is above a certain threshold. Any pixel with a variance near zero is almost certainly part of the background. By not including it in the graph, we set those pixels as background pixels, removing any possible noise that could occur if the graph cut algorithm decided that they were foreground pixels. This also solves the problem where some of the weighting schemes we use divide a number by the variance. If the variance is zero, this operation would fail. Finally, we perform minor morphological operations on the graph cut results to clean them up.

3.5.1 T-link Weights

3.5.1.1 Exponentials

Given a pixel in the background image with mean grayscale value μ and variance σ^2 and a pixel in the current image with the grayscale value x, we can model the background with a non-normalized Gaussian with the following equation.

$$e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

Letting α standard deviations be the cutoff, then if

$$\frac{|x-\mu|}{\sigma} \leq \alpha \tag{3.5}$$

then the current pixel is considered to be a background pixel. If not, then it is considered to be a foreground pixel. Now, we turn this model for the background into t-link weights for the graph cuts.

Let

$$f_1(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} + \beta \tag{3.6}$$

be the background model, where $\beta > 0$, and let

$$f_2(x) = 1 - e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.7}$$

be the "foreground" model. We do not have an exact foreground model, so these weightings are more like the similarity and dissimilarity to the background model. For the model to work, it is required that the two models cross at α standard deviations away from the mean:

$$f_1(\mu + \alpha\sigma) = f_2(\mu + \alpha\sigma) \tag{3.8}$$

$$e^{-\frac{(\mu+\alpha\sigma-\mu)^2}{2\sigma^2}} + \beta = 1 - e^{-\frac{(\mu+\alpha\sigma-\mu)^2}{2\sigma^2}} \tag{3.9}$$

$$e^{-\frac{\alpha^2}{2}} + \beta = 1 - e^{-\frac{\alpha^2}{2}} \tag{3.10}$$

$$\beta = 1 - 2e^{-\frac{\alpha^2}{2}} \tag{3.11}$$

For $\alpha = 1.2$, $\beta = 0.03$ by substitution. For values of $\alpha \geq 1.2$, we will have $\beta > 0$. Also, Equations

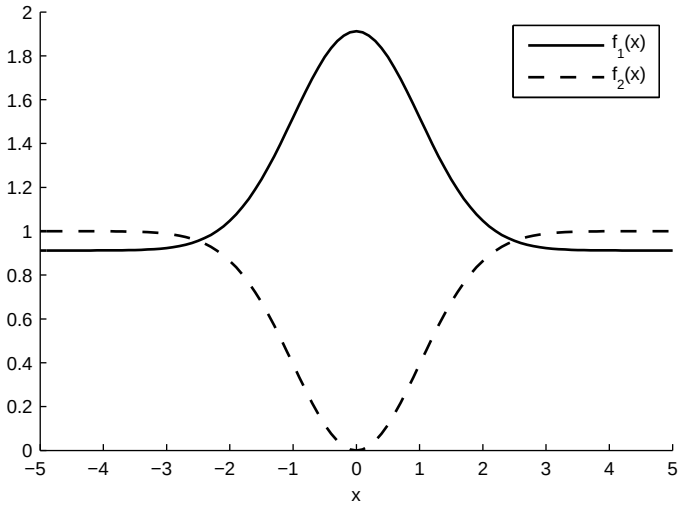


Figure 3.5: A plot of Equations 3.6 and 3.7.

3.6 and 3.7 will be multiplied by a factor of 255 in the implementation, to keep the weights from all being between 0 and 2. The weights are then rounded to integer values.

Figure 3.5 shows example plots of Equations 3.6 and 3.7, with $\mu = 0$, $\sigma^2 = 1$, and $\alpha = 2.5$. β is then calculated to be 0.9121. It can be seen from these plots that pixels similar to the background pixel in question will have a high weighting using Equation 3.6, while pixels different from the background pixel will have a high weighting using Equation 3.7. The two plots cross at α, which is 2.5 standard deviations in this case. Thus pixel values at exactly 2.5 standard deviations from the background pixel value have equal weights from both equations and could be foreground or background pixels.

We will refer to this model as the exponential model.

3.5.1.2 Absolute Differences

Instead of using exponentials as we did previously, here we make use of the absolute difference. Let

$$f_3(x) = K - |x - \mu| + L \tag{3.12}$$

be the background model, and let

$$f_4(x) = |x - \mu| + L \tag{3.13}$$

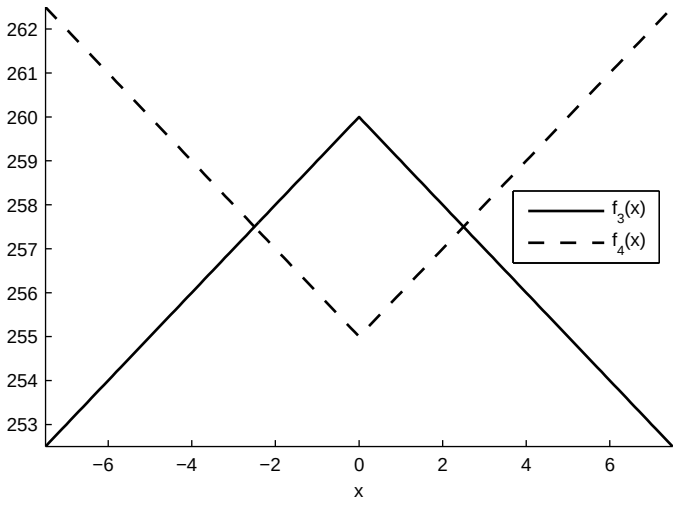


Figure 3.6: A plot of Equations 3.12 and 3.13.

be the "foreground" model, where $L = \max(|x-\mu|) = 255$ is the maximum possible distance between x and μ to ensure that Equation 3.12 remains non-negative. $K > 0$ is the relative shift between the two functions to ensure that they cross at α standard deviations:

$$f_3(\mu + \alpha\sigma) = f_4(\mu + \alpha\sigma) \tag{3.14}$$

$$K - |\mu + \alpha\sigma - \mu| + L = |\mu + \alpha\sigma - \mu| + L \tag{3.15}$$

$$K - |\alpha\sigma| + L = |\alpha\sigma| + L \tag{3.16}$$

$$K = 2|\alpha\sigma| = 2\alpha\sigma \tag{3.17}$$

A reasonable value for α is 1.2 standard deviations. In the actual implementation, Equation 3.12 is implemented as $f_3(x) = \max(0, K - |x - \mu| + L)$, to ensure all values of f_3 remain non-negative.

Figure 3.6 shows example plots of Equations 3.12 and 3.13, with $\mu = 0$, $L = 255$, $\alpha = 2.5$, and $\sigma = 1$. K is then calculated to be 5. Again, pixels with a value similar to the background pixel will have a high weighting using Equation 3.12, while pixels different from the background pixel will have a high weighting using Equation 3.13. The two plots cross at $\alpha = 2.5$ standard deviations. Therefore, pixels with values at 2.5 standard deviations from the mean have equal weights from both equations and could be either foreground or background pixels.

We will refer to this model as the absolute difference model.

3.5.1.3 Simple Grayscale Differences

Another simple model is used on various images initially, before testing starts on vehicle segmentation. This model is used to segment images based on grayscale values. For a binary graph cut, foreground and background grayscale values are needed. For a multi-way cut, more grayscale values are needed based on how many segmentations are desired. Let these grayscale values be referred to as C_1, C_2, etc.

Given a pixel with the grayscale value x, the t-link weights are then

$$f_5(x) = |x - C_m| \tag{3.18}$$

$$f_6(x) = |x - C_n| \tag{3.19}$$

where m and n are indices on the grayscale values such that $m \neq n$. We will refer to this model as the simple grayscale model.

3.5.2 N-link Weights

N-link weights are the weights on the edges between neighboring nodes. In this case, we are using a 4-connected graph. That is, each pixel node in the graph is connected to those above, below, to the left, and to the right of it. They are not interconnected diagonally, which would make the graph 8-connected. We use two different sets of n-link weights.

3.5.2.1 Distance Weights

For our first set of weights for the n-links, we simply use the Euclidean distance between the neighboring pixel locations,

$$f_7 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.20}$$

where the pixel locations are denoted by the coordinates (x_1, y_1) and (x_2, y_2). For a 4-connected graph, all the weights end up as a constant 1. We will refer to this model as the distance model.

3.5.2.2 Smoothing Weights

For our second set of weights, we use the similarity of grayscale pixel values to smooth the graph cut results. We weight the edges between two neighboring nodes using

$$f_8 = (J - |x - y|)\gamma \tag{3.21}$$

where x and y are the grayscale values of the two neighboring pixels, $J = \max(|x - y|) = 255$ is the maximum possible difference between the two pixels, and γ is a modifier (generally from 0 to 1) on the weight value that determines how much smoothing takes place. How much γ affects performance is dependent upon the number of standard deviations used for α in the t-link weighting systems discussed previously. By using the similarity of the grayscale pixel values in the weighting scheme, pixels that are very similar have a high weight on the edge between them. This means that those pixels are unlikely to be separated by the graph cut algorithm. We will refer to this model as the smoothing model.

3.6 Background Subtraction

To compare our traffic detection with another algorithm that has been used in traffic detection, we implement background subtraction. The background is again computed by averaging the image frames in the image sequence we are using. Then, for every frame in our image sequence, we take the absolute difference between the current frame and the background. Any pixel values with an absolute difference above a certain threshold are considered to be part of a vehicle. Minor morphological operations are then performed to clean up the results.

3.7 Implementation Details

All code for this thesis was developed using Microsoft Visual Studio 2008 on Windows 7. The Blepo Computer Vision Library [2] is used to handle image processing tasks. All graph cuts, except for one instance to be noted in chapter 4, utilize the max-flow/min-cut software provided by Kolmogorov [12], which is based upon [5].

Chapter 4

Results

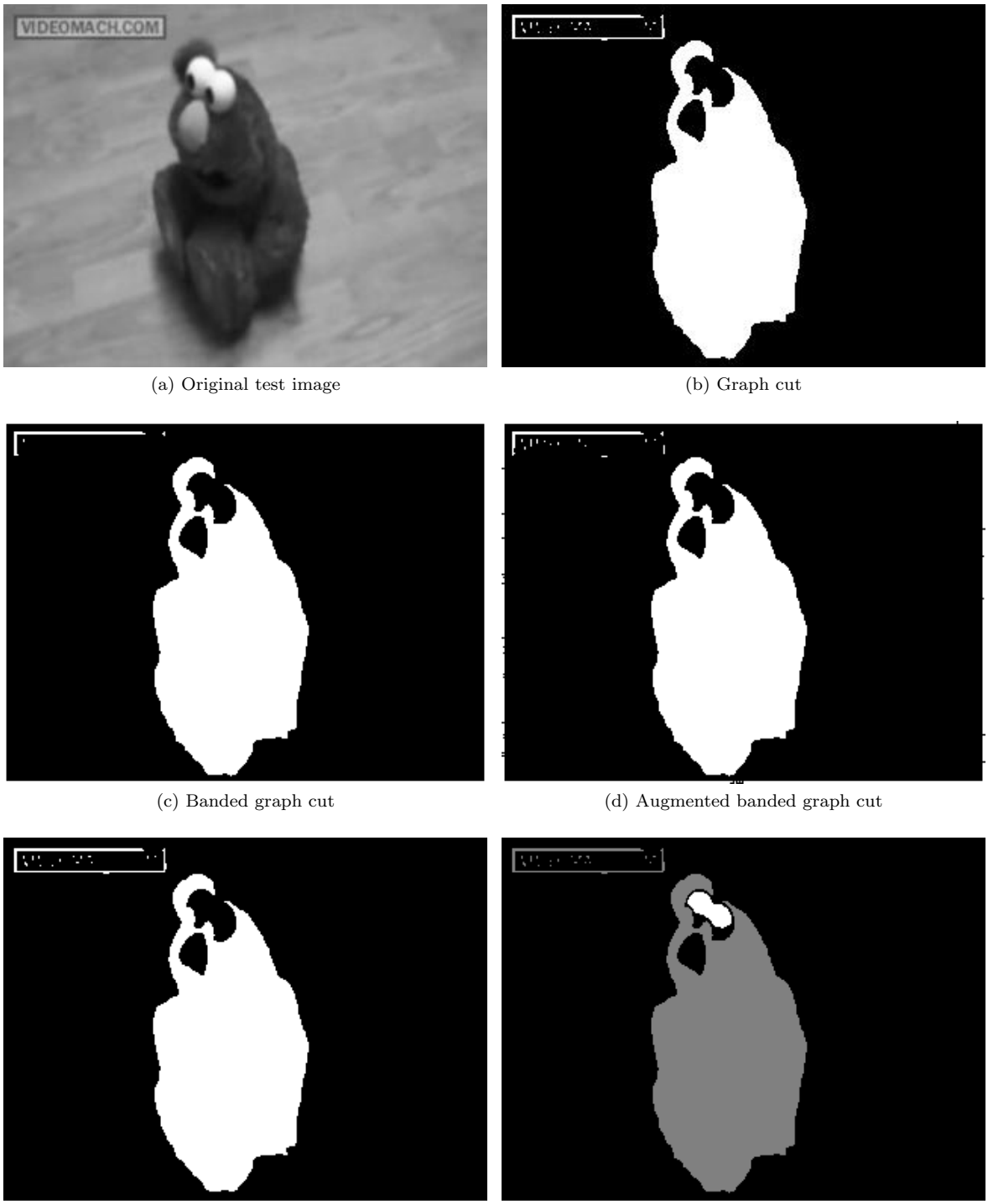
In section 4.1, we compare and contrast different types of graph cuts and how they perform compared to each other. In section 4.2, we discuss the results we obtained using graph cuts to segment traffic images taken from video sequences.

4.1 Different Types of Cuts

In this section, we first discuss the results of graph cuts. We then discuss how the different algorithms compare to each other with respect to run-time.

4.1.1 Image Results

Figure 4.1 shows the various types of graph cuts discussed in this thesis. Figure 4.1a shows the base image that we used for testing. All of these cuts were performed using the simple grayscale model described in section 3.5.1.3 for the t-link weights and the distance model of section 3.5.2.1 for the n-link weights. The foreground grayscale value was chosen from among the fur pixels and the background grayscale value was chosen from among the floor pixels. Figure 4.1b shows the results using a basic graph cut. The max flow calculations were performed over the entire image at full scale, so this result image is the most detailed. The white part of the image is considered to be the foreground of the image by the graph cut algorithm, while the black part of the image is considered to be the background. The graph cut successfully segments the foreground from the background, but there are still areas where errors are made. For instance, part of the label in the top left-hand



(a) Original test image

(b) Graph cut

(c) Banded graph cut

(d) Augmented banded graph cut

(e) Our graph cut implementation

(f) Multi-way cut

Figure 4.1: Different types of graph cuts.

corner of the image shows up as being part of the foreground of the image. Also, most of the eyes (aside from the pupils) and the nose both show up as being a part of the background of the image. Using the simple grayscale model, the graph cut is essentially looking at whether a given pixel is closer to the foreground grayscale value or the background grayscale value. Parts of the label are dark enough to be fur, and parts of the eyes and nose are light enough to be considered to be part of the floor. This is why the segmentation is not perfect.

Figure 4.1c shows the banded graph cut with two extra levels (the number of times the image was originally downsampled). This image has a bit less detail than the graph cut, most of which can be noticed in the upper left-hand corner of the images. The reason that it has less detail has to do with the downsampling that is performed at the beginning of the algorithm. When the image is downsampled, detail is lost. When we downsample twice, enough detail in the label is lost on the bottom and right-hand sides of the label that they do not register as part of the foreground. When the image is then upsampled and resegmented, the resegmenting of the band around the foreground segmentation does not re-add that detail. Thus it is missing on the final result image.

Figure 4.1d shows the results for a banded graph cut with the Laplacian augmentation using two extra levels. This image has a bit more detail than the banded graph cut, as it is a slightly augmented version. The augmentation is specifically to add back in some of the lost detail. As in the banded graph cut, the image is downsampled several times. However, when the images are segmented and up-sampled, the band around the segmentation and some extra pixels get resegmented. These extra pixels are added to the area to be segmented in areas in which a lot of detail was lost. This means that some detail that was lost by downsampling is added back in. However, it is still less detailed than the basic graph cut, which is using the full scale image. This is why the segmentation of the label area in Figure 4.1d is in between the segmentations in Figures 4.1b and 4.1c as far as how well the segmentation worked.

Figure 4.1e is the only result in Figure 4.1 that was not implemented using Kolmogorov's code [12]. This result image was created using our own implementation of a max-flow/min-cut algorithm. As such, the results are very similar, if not identical, to the results seen in Figure 4.1b. Any discrepancies would depend on if we used different methods to break ties in deciding whether a pixel should be a source pixel or a sink pixel. For example, if all the flows in and out of a pixel node were totally saturated, cutting off the pixel from both the source and the sink, a choice has to be made which terminal to assign the pixel node to. If our method defaults to setting a pixel node as

a source pixel and Kolmogorov's method defaults to setting pixel nodes as sink pixels, that would make the output different on the relatively few pixel nodes that had ties that needed to be broken.

Figure 4.1f shows a multi-way cut on the same image. Instead of a binary cut, it cuts the image into three different segments for the background, the fur, and the eyes. A third grayscale value was thus taken from among the pixels in the eyes to use as a model. Again, because of the simple grayscale model being used, the segmentation is basically deciding which grayscale value any given pixel is closest to. Therefore, compared to Figure 4.1b, the areas that were segmented into the foreground are still segmented as fur. However, some areas which were previously part of the background are now segmented into the eyes, since they are closer in value to the eye grayscale value chosen then the floor grayscale value. Some areas of the eyes are still put in the same segment as the floor, though. This is because those parts (mainly the lower shadowed areas), are a gray shade very close to the shade of the floor. Again, the segmentation is not perfect, but it is acceptable as a test image.

4.1.2 Timing Results

The time each type of graph cut takes to run is also interesting. Some cuts may not be as accurate as others, but might have a faster run-time that makes them more useful. At the same time, they may be more complicated to implement. Table 4.1 shows timing data for all the types of graph cuts discussed, from the basic graph cut all the way to the multi-way cut. All cuts in this section were performed on the same 320x240 pixel image, and can be seen in Figure 4.1.

The graph cut, which is the most accurate type of cut, is actually the second fastest to run out of the three types of binary graph cut implemented using Kolmogorov's code. Because the graph cut operates on each pixel node in the image, building the graph takes longer. The build time is over twice the time required to build the graphs for the banded graph cut and augmented banded graph cut. The max-flow/min-cut computations also take slightly longer. The final step is to take the results from the graph cut and apply them to a result image for viewing. We refer to this as "pasting."

The banded graph cut, which is the least accurate of the cuts tested, is the quickest type of cut to run. The key here is the downsampling and upsampling that the method uses. The first segmentation is done on an image one sixteenth of the size of the the full image (for two extra levels used in the computations), so it is much faster. That segmentation is upsampled, and then

the border around the segmentation (a band, if you will) is resegmented. This second segmentation is on an image one quarter of the size of the full image, and it is not performed on the entirety of that image. The segmentation of the quarter-size image is obtained and upscaled again. Then a band around this segmentation is resegmented. This is a segmentation on the full-size image, but the segmentation again only covers a band around the previous segmentation. These three segmentations, because the three graphs likely have fewer nodes than 320x240 pixel nodes total, take less time to perform than the basic graph cut takes to do one segmentation. The time taken to paste the results onto an image is very similar, even though three separate result images have to be filled. This is because the initial results are pasted on an image sixteen times smaller than the full size image. Those results are then upsampled. When new segmentations are performed, the upsampled results are only changed in small bands around the previous results. The total changes made on three result images are similar in scale to the changes made to the single result image for the basic graph cut. However, the banded graph cut has other steps that the basic graph cut does not have. For instance, images must be upsampled and downsampled. The calculation of the band location needs to be performed for each segmentation after the first one. All of these steps are included in the "other calculations" field in Table 4.1. Even with these extra steps, the banded graph cut is faster than the basic graph cut.

The banded graph cut with the Laplacian augmentation is mostly similar in timing to the banded graph cut. However, the band around each of the upsampled segmentations needs to be augmented with data from the Laplacian pyramid. This extra step takes some extra time that the other algorithms do not have to take. This is why the augmented banded graph cut takes longer to run than the banded graph cut. With the extra time taken, the augmented banded graph cut also adds more accuracy, taking it closer to the accuracy of the basic graph cut than the banded graph cut. All the extra steps in this algorithm actually make the augmented banded graph cut slower than the basic graph cut, even though it has lower accuracy. On larger images and 3D datasets, the banded graph cut and the augmented banded graph cut should be even faster compared to the basic graph cut. This is because the extra computations for the banded graph cut and the Laplacian pyramid are linear complexity in the number of nodes. The graph cut computations are not, and take much longer in comparison as the graph size increases.

Our implementation of the maximum flow algorithm for computing graph cuts performs poorly compared to the implementation provided by Kolmogorov. While building the graph is

	Graph cut	Banded graph cut	Augmented banded graph cut	Graph cut (our implementation)	Multi-way cut
Time to build the graph	16	5	6	7	35
Time to calculate max-flow/min-cut	3	2	2	4221	19
Time to paste results onto an image	2	3	2	1	9
Laplacian augmentation	0	0	8	0	0
Other calculations	0	6	5	0	0
Total	21	16	23	4229	63

Table 4.1: Computing times for various parts of the cuts in milliseconds (ms).

faster, the total time for the computations is much, much slower. Computing the maximum flow takes almost four and a half seconds, which is about a thousand times slower than the algorithm provided by Kolmogorov. Finding out whether each pixel node is still attached to either the source or sink node requires a similar amount of time compared to Kolmogorov's implementation. While the segmentation results are similar to those given by Kolmogorov's code, the timing data makes this version less useful for segmenting traffic images.

The final type of cut implemented was the multi-way cut. Unlike the other cuts, this approach segments images into three or more parts. Essentially, a binary cut is looped through the possible combinations. So, for instance, labels 0, 1, and 2, would be paired up into $(0, 1)$, $(0, 2)$, and $(1, 2)$. A binary cut (in this case the normal graph cut) is run using each of those pairs of labelings. Each of these sets of cuts are also run multiple times, in case any changes made in one cut affected the changes made in the previous/following cuts. By the time this series of cuts finishes, successive cuts should have minimal affect on the final cut. However, after the first segmentation ($(0, 1)$ in our implementation) each successive cut is not performed on the entire image. The successive cuts are only performed on the parts of the image that have one of the current pair of labels. That is, a cut using the labels $(1, 2)$ would not affect any pixel nodes in the image labeled 0. Because the entire

image is not cut each time, the times required for building and segmenting the multi-way cut are not a straight multiple of the times required for the basic graph cut.

4.2 Traffic Detection

Now we apply our methodology to segmenting vehicles in traffic images by running a graph cut on each frame, using both our exponential model and our absolute difference model for the t-links. We also test using both the distance and smoothing n-link weight models.

4.2.1 Distance N-link Weight Model Results

Figure 4.2 shows a series of result images captured from a single frame of our first set of test footage. Figure 4.2a shows the current frame of the footage that the graph cut was run on. Several vehicles can be seen driving down a stretch of highway. Figure 4.2b shows the background image of that test video. The background image was computed by averaging the image frames over the entire length of footage. This is a plausible result for the background image, as the vehicles are totally removed. At the same time, the variance of each pixel in the camera frame was computed over the whole length of video footage as well. The variance can be seen in Figure 4.2c, where highly varying pixels in the video footage are very bright. The only pixels in the image that vary very much are on the two stretches of road in the image. One goes straight through the middle of the image, and the other can be seen in the upper right-hand corner. Any pixel that is close to black in the variance image is assumed to be a background pixel and is not included in the segmentation.

Figures 4.2d and 4.2e show the results of segmenting the image using the exponential weight model and the absolute difference weight model for the t-link weights, respectively. The distance weight model is used for the n-link weights for all the results in this section. Foreground pixels are white, and background pixels are black. The segmentation of the vehicles into the foreground works well, but there are areas around the windows where background noise exists in the segmentation. The results using the exponential and absolute difference models are essentially the same.

Using graph cuts to segment images tends to work better the closer the vehicles are to the camera and the bottom right-hand corner of the screen. That section of the camera frame has the greatest amount of detail as well as the best angle. The third vehicle from the bottom right-hand side is partially segmented into the foreground, but only one of what appears to be two vehicles in

the top left-hand corner of the frame is picked up by the segmentation. This method of segmentation seems to work best in areas where the grayscale values are similar. For instance, the entire hood of the second vehicle is segmented in one section, as is most of the side panel. However, background noise appears in between the two where there is a shift in grayscale values.

Figures 4.3 and 4.4 show segmentations performed on other pieces of video footage taken on the same stretch of highway but at slightly different times of day. The cars are casting shadows differently in each series of images, and the general lighting conditions are different as well. The segmentation of vehicles in Figure 4.3 works well, segmenting most of the two vehicles close to the camera into the foreground. There is a lot of background noise in the area between the wheels of the van, however. Four of the vehicles in the distance appear to be picked up by the segmentation, as well.

The segmentations in Figure 4.4 are very good. Vehicles very close to the camera are segmented into almost a single solid block, aside from small areas. Many of the vehicles in the distance appear in the segmentation as well. These three videos are all at a resolution of 352x240.

Figure 4.5 is the worst test case we use. The camera appears to be either dirty or have some sort of a cover on it that gives the entire image an odd gray tint. This video also has the most traffic in it. The moving truck in the bottom right-hand corner has the best segmentation. Several of the trucks in the top right-hand corner have reasonable segmentations as well. The biggest problem, however, is that many of the cars in the middle of the frame are almost entirely segmented into the background. This video was taken at a resolution of 320x240.

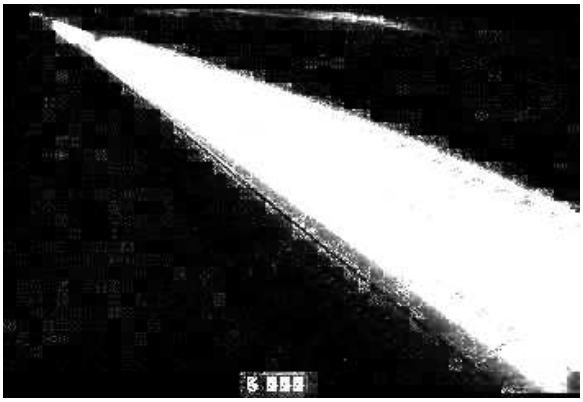
4.2.1.1 Comparison with Background Subtraction

In this section, we compare our method of segmenting traffic images with another commonly used method to evaluate our method's effectiveness. Figure 4.6 shows a comparison between our method and background subtraction using the first series of footage. Only one type of cut (absolute difference) is shown in the comparison figure because the results of both methods are very similar. Figures 4.6 and 4.7 both use the distance weight model for the n-link weights. The results are actually quite similar for our method and background subtraction. In Figure 4.6, the segmentation using graph cuts is slightly better, with more of the second and third vehicles from the right segmented into the foreground.

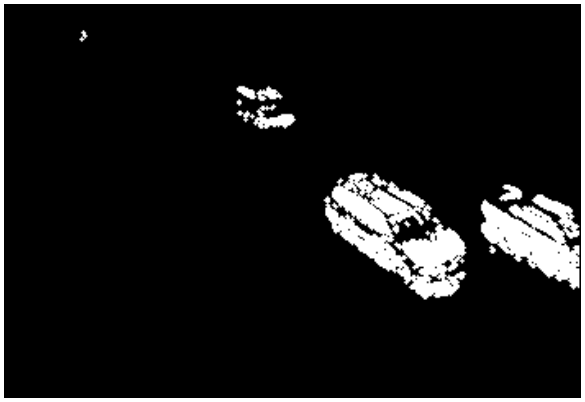
In Figure 4.7, the segmentation using background subtraction is slightly better, as more of

(a) Base image

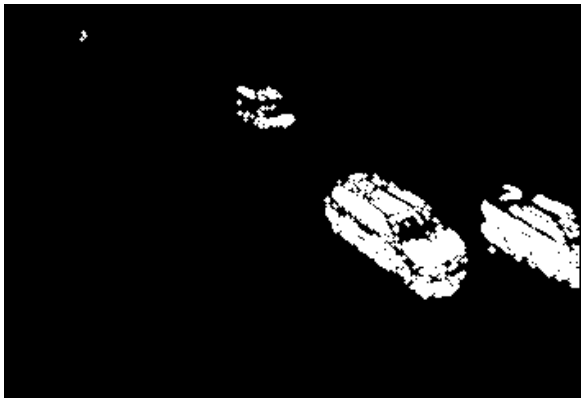
(b) Background



(c) Variance

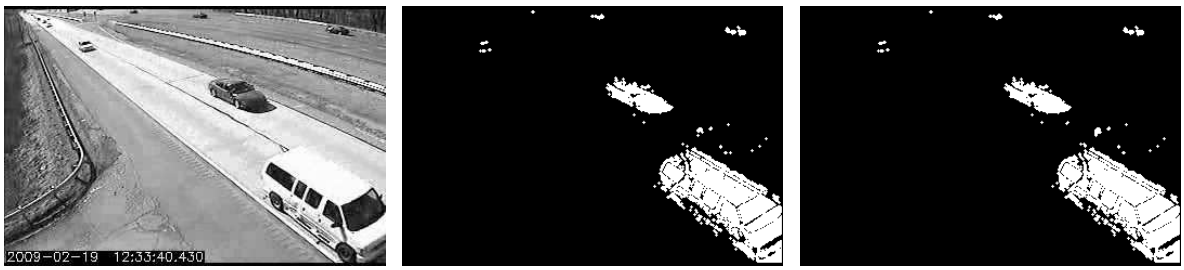


(d) A cut using the exponential model



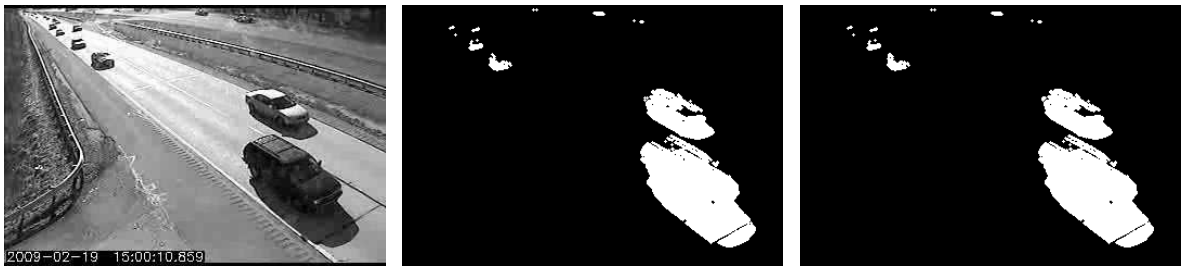
(e) A cut using the absolute difference model

Figure 4.2: Results from the first series of footage (distance n-link weight model).



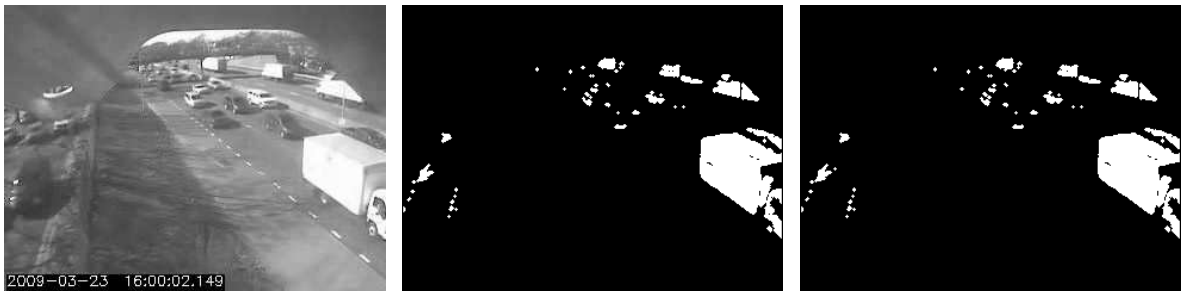
(a) Base image (b) A cut using the exponential model (c) A cut using the absolute difference model

Figure 4.3: Results from the second series of footage (distance n-link weight model).



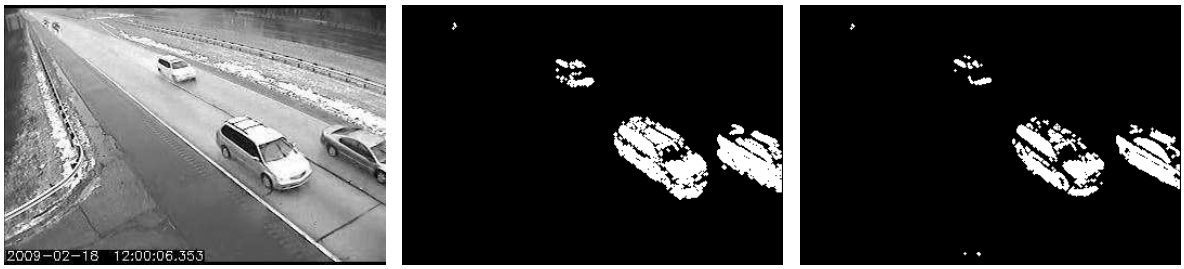
(a) Base image (b) A cut using the exponential model (c) A cut using the absolute difference model

Figure 4.4: Results from the third series of footage (distance n-link weight model).



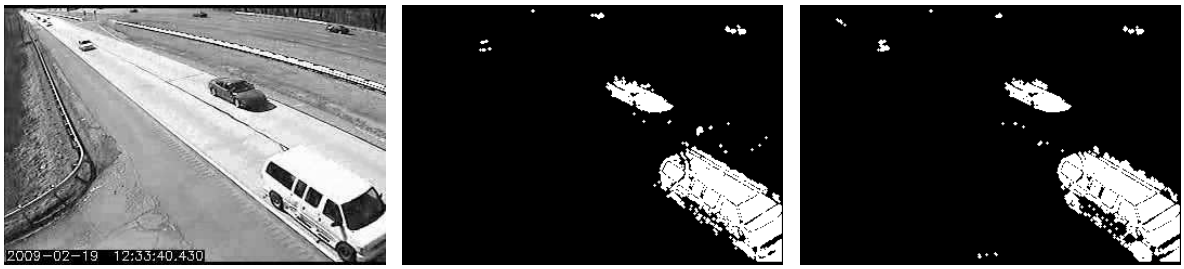
(a) Base image (b) A cut using the exponential model (c) A cut using the absolute difference model

Figure 4.5: Results from the fourth series of footage (distance n-link weight model).



(a) Base image (b) Absolute difference cut (c) Background subtraction

Figure 4.6: A comparison of background subtraction versus graph cuts using our first series of footage (distance n-link weight model).



(a) Base image (b) Absolute difference cut (c) Background subtraction

Figure 4.7: A comparison of background subtraction versus graph cuts using our second series of footage (distance n-link weight model).

the van is segmented into the foreground. This is possibly due to the threshold on the variance that we use in our graph cuts method. The variance near the edge of the road tends to be lower, and if it is below our threshold, parts of vehicles near the road edge may not be included in the graph cut at all. Background subtraction also picks up the string of vehicles in the upper left-hand corner of the screen, where graph cuts do not.

4.2.2 Smoothing N-link Weight Model Results

Figure 4.8 shows a series of result images captured from a single frame of our first set of test footage. All results in this section use our smoothing n-link weight model. Figure 4.8a shows the current frame of the video that the graph cut segmentation is operating on. Figures 4.8b and 4.8c show the graph cut segmentation results using the exponential weight model and the absolute difference weight model for the t-links, respectively. Due to the use of the smoothing n-link weight model, the vehicles are more solidly segmented. The large van is segmented into one solid piece in

(a) Base image (b) A cut using the exponential model (c) A cut using the absolute difference model

Figure 4.8: Results from the first series of footage (smoothing n-link weight model).

both segmentations, with the absolute difference cut performing slightly better, as there is no "hole" in the van segmentation. The absolute difference cut also manages to keep the car on the far right segmented into one piece, where the exponential cut does not. However, the absolute difference cut tends to miss the smaller pieces, such as the bumper area of the third vehicle and the small vehicle in the far distance.

Figure 4.9 shows a set of results from our second series of test footage. These results are excellent, especially those of the absolute difference cut. Both of the vehicles in the bottom right-hand corner are segmented into one piece, as is the vehicle in the top right-hand corner. The absolute difference cut misses some of the smaller vehicles in the far distance, but does not have the hole in the van that the exponential cut does.

Figure 4.10 shows some results from our third series of footage. These results are also excellent. Most of the vehicles in the segmentation are in one large piece, with little noise. There is a minor problem in front of the vehicle in the bottom right-hand corner of the image, where the crack on the ground is a similar shade of gray to the vehicle shadow. This causes there to be a line of background noise running across the segmentations. The exponential cut also has a hole in the segmentation of the second vehicle in line, where the absolute difference cut does not. However, the absolute difference cut also attaches the two closest vehicles together, which is a problem. Many of the vehicles in the background are picked up in both segmentations.

Figure 4.11 shows the results from our fourth series of footage. This piece of footage is still problematic, as most of the vehicles in the middle of the frame are still missed entirely. However, most of the vehicles that are found by the graph cut are in larger segments compared to the results in Figure 4.5. Much of the noise from Figure 4.5 is missing as well.

(a) Base image

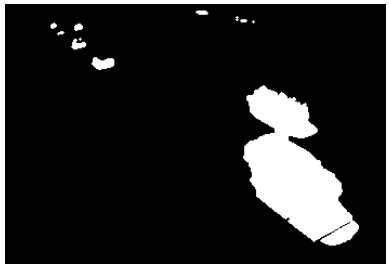
(b) A cut using the exponential model

(c) A cut using the absolute difference model

Figure 4.9: Results from the second series of footage (smoothing n-link weight model).

(a) Base image

(b) A cut using the exponential model



(c) A cut using the absolute difference model

Figure 4.10: Results from the third series of footage (smoothing n-link weight model).

It is possible that some of the differences between the results for the exponential cut and the absolute difference cut can be resolved by modifying the γ variable from Equation 3.21 separately in each cut.

4.2.2.1 Comparison with Background Subtraction

In this section, we compare our results using the smoothing n-link weight model to results found using background subtraction. Figure 4.12 shows a comparison using an image from our first series of test footage. We show the absolute difference cut because it seemed to have fewer holes in the segmentation. Compared to background subtraction, our method works really well. The close vehicles on the right-hand side of the image frame are extremely solid using our method, where they are very fragmented using background subtraction. Background subtraction does pick up smaller objects, however, such as one of the vehicles in the far distance and the changing time at the bottom of the frame.

Figure 4.13 shows a similar comparison using an image from our second series of footage.

(a) Base image

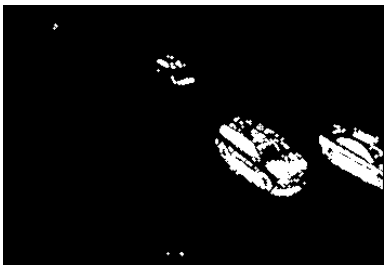
(b) A cut using the exponential model

(c) A cut using the absolute difference model

Figure 4.11: Results from the fourth series of footage (smoothing n-link weight model).

(a) Base image

(b) Absolute difference cut



(c) Background subtraction

Figure 4.12: A comparison of background subtraction versus graph cuts using our first series of footage (smoothing n-link weight model).

Again, the graph cut segmentation performs very well comparatively. Each of the large vehicles close to the camera are segmented into one large piece. Background subtraction leaves a large amount of noise in each vehicle segmentation. However, it is worth noting that using the smoothing model for n-link weights tends to lose objects, such as the more distant vehicles, into the background.

4.2.3 Timing Comparison

Background subtraction takes between 1 and 2 milliseconds per frame to perform, making it faster than the 5 to 15 milliseconds per frame required for performing a graph cut. Background subtraction is also much simpler to implement than graph cuts.

(a) Base image (b) Absolute difference cut (c) Background subtraction

Figure 4.13: A comparison of background subtraction versus graph cuts using our second series of footage (smoothing n-link weight model).

Chapter 5

Conclusions and Discussion

In this thesis, we investigate the use of graph cuts in segmenting traffic images. We compare the results of various types of graph cuts to determine which type would be best for our purposes. We propose a method for segmenting traffic images using graph cuts, using two different weighting systems. The results of these two weighting systems are compared to see if either is drastically better than the other. Finally, our method of using graph cuts to segment traffic images is compared to background subtraction, which is another commonly used method.

Out of all three types of graph cuts implemented, we decide to use the most basic form for traffic image segmentation. The basic graph cut is the most accurate of all the different graph cuts we tested, even though it is not the fastest. The fastest graph cut is the banded graph cut. However, the banded graph cut is only faster by roughly 25%. For the size of image we are using for testing, all three types of graph cut take close to the same amount of time. We use the most accurate type: the basic graph cut.

Our method for segmenting traffic images works rather well with good images where traffic was sparse. It detects most vehicles, while ignoring most random noise. However, it does have problems when the image quality is poor. Our method also has trouble with roadways where there is a lot of traffic. Some vehicles are detected, but others are ignored entirely. Both of our t-link weight models work similarly when using the distance n-link weight model. Very few differences are noticeable to the naked eye. When using the smoothing n-link weight model, the results are similar to each other. However, the absolute difference t-link weights provide segmentations with fewer holes, which is preferable.

The results from our traffic image segmentation method when using the distance n-link weight model are comparable to those obtained using background subtraction. Background subtraction works slightly better for some images, but worse for others.

However, when using our smoothing weight model, we are able to segment vehicles into single sections, instead of the multiple pieces divided by background information that background subtraction often leaves. Because there is less background noise in each segmentation, our graph cut method performs better than background subtraction. Combining the absolute difference t-link weight model with the smoothing n-link weight model provided the best performance. However, that performance comes at a cost: graph cuts are slower than background subtraction.

There are several things that could be added to this project in future work. Using multi-way cuts, shadow segmentation could be added if a weighting system could be devised. This project could also be adapted to use color images rather than grayscale. The absolute difference model could easily be expanded to RGB rather than just grayscale. The exponential model should be expandable to RGB as well, but it would be more difficult.

Bibliography

[1] G. Alessandretti, A. Broggi, and P. Cerri. Vehicle and guard rail detection using radar and vision data fusion. *Intelligent Transportation Systems, IEEE Transactions on*, 8(1):95 –105, March 2007.

[2] S. Birchfield. Blepo computer vision library. `http://www.ces.clemson.edu/~stb/blepo/`.

[3] S. Birchfield. Klt: An implementation of the kanade-lucas-tomasi feature tracker. `http://www.ces.clemson.edu/~stb/klt/`.

[4] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105 –112 vol.1, 2001.

[5] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124 –1137, Sept. 2004.

[6] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222 –1239, Nov 2001.

[7] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, 1990.

[8] J.-W. Hsieh, S.-H. Yu, Y.-S. Chen, and W.-F. Hu. Automatic traffic surveillance system for vehicle tracking and classification. *Intelligent Transportation Systems, IEEE Transactions on*, 7(2):175 – 187, June 2006.

[9] N. Kanhere and S. Birchfield. Real-time incremental segmentation and tracking of vehicles at low camera angles using stable features. *Intelligent Transportation Systems, IEEE Transactions on*, 9(1):148 –160, March 2008.

[10] Y.-K. Ki and D.-Y. Lee. A traffic accident recording and reporting model at intersections. *Intelligent Transportation Systems, IEEE Transactions on*, 8(2):188 –194, June 2007.

[11] Z. Kim. Real time object tracking based on dynamic feature grouping with background subtraction. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8, June 2008.

[12] V. Kolmogorov. Maxflow - software for computing mincut/maxflow in a graph. `http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software/maxflow-v3.01.src.tar.gz`.

[13] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 259 – 265 Vol. 1, Oct. 2005.

[14] N. Saunier and T. Sayed. A feature-based tracking algorithm for vehicles in intersections. In *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, page 59, June 2006.

[15] A. Sinop and L. Grady. Accurate banded graph cut segmentation of thin structures using laplacian pyramids. In Rasmus Larsen, Mads Nielsen, and Jon Sporring, editors, *MICCAI (2)*, volume 4191 of *Lecture Notes in Computer Science*, pages 896–903. Springer, 2006.