A Highly Efficient And Automated Approach To Identifying And Classifying Urban Traffic Images Using Self Organizing Maps and Artificial Intelligence

by

Hisham Mardam Bey

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science

Project Adviser: Dr. Nash'at Mansour

Department of Computer Science
Lebanese American University
February 2006

LEBANESE AMERICAN UNIVERSITY

GRADUATE STUDIES

We hereby approve the project of

Hisham Mardam Bey

Candidate for the *Master of Science* degree*

(signed) [redacted] Nash'at Mansour
(chair)

[redacted] Faisal Abu Khzam

_____

_____

_____

date Feb. 3, 2006

*We also certify that written approval has been obtained for any proprietary material contained therein.

I grant to the LEBANESE AMERICAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or to its students, agents and employees. I further agree that the University may reproduce and provide single copies of the work, in any format other than in or from microforms, to the public for the cost of reproduction.

_____

A Highly Efficient And Automated Approach To Identifying And Classifying Urban Traffic Images Using Self Organizing Maps and Artificial Intelligence

ABSTRACT

by

Hisham Mardam Bey

Identifying moving objects and isolating background noise to concentrate on a particular foreground object is a task that is heavily needed by a lot of applications in this day and time. Such a task is not always an easy one and can prove very tedious if done using the manual approach of assigning human beings to watch and enter data for long working hours. This is both a waste of human resources and valuable company time. Previous work on this subject included subtracting background information using conventional imaging techniques. This paper proposes a highly efficient approach using both optimized software and hardware techniques to achieve the best results in terms of correctness and speed. We do so by employing heavily optimized algorithms which divide the load over the CPU (central processing unit) and the GPU (graphical processing unit) and hence extract maximum performance from both processing units simultaneously. This sort of hybrid method has not been observed in proposed solutions to our problem.

Keywords: Self Organizing Maps, Kohonen Networks, artificial intelligence, foreground identification, urban traffic video.

To my family and Sara.

ACKNOWLEDGEMENTS

I would like to thank my dad for inspiring this idea and allowing it to flourish into this paper, my adviser for pointing me in the right direction when needed, and to all my family, friends, and Sara for bearing with me while I worked on this.

ACKNOWLEDGEMENTS

I would like to thank my dad for inspiring this idea and allowing it to flourish into this paper, my adviser for pointing me in the right direction when needed, and to all my family, friends, and Sara for bearing with me while I worked on this.

Contents

List of Figures

Chapter 1

Introduction

When identifying moving objects, a common technique that can be used is removal of the background and any noise that might be available. In the case of urban traffic, we need to consider some extra factors that might not be present in other similar cases. Our algorithm must account for weather changes, light changes, unexpected shadows, and smaller objects that might abruptly slip into the background area. Conventional methods usually associate a given background image or color and use that to determine what fits into the background and what does not (color spectrum and similarity algorithms). Instead of using the more conventional methods to separate the background and the foreground, we will rely on Kohonen networks [7, 8] (self organizing maps, referred to as SOM from now on) and artificial intelligence techniques to extract and compare images of vehicles. Moving traffic can be very difficult to capture and identify unless we have one of two conditions satisfied. The first condition being that the camera is placed at a far location from the object to avoid motion blur, and the second condition being that the camera is placed at an intersection point where cars need to slow down. The interest of this project is in the latter case as the first case can not apply inside cities but tends to work more on highways. Our best results can be obtained when cars come to a complete stop because of a red traffic light. We do not, however, assume that this is the only scenario as we aim to be able to identify even those cars that move at a slow to normal speeds.

Most of the work that has gone into this task was either involved with common day to day methods for image-object recognition (frame differencing, median filters, Kalman filters) or is done with the aid of specialized hardware. Some work uses neural networks but this field is still young, and very little work has been done using neural networks (mainly for face recognition) and particularly Kohonen networks. We will further discuss previous work in Chapter 3. We state the problem we are trying to solve in Chapter 2. In Chapter 3, we describe Kohonen networks (SOM). In Chapter 4 we discuss our proposed solution. In Chapter 5 we explain how we can match the collected data against a preset database of identification criteria. Following that, in Chapter 6 we present some experimental results for our work. Finally, we discuss any possible future work and conclude our paper and finally present our conclusion in Chapter 7.

Chapter 2

Description of the Problem

Several private companies and governmental agencies [1, 5] conduct traffic studies to deduce traffic patterns, road usability, and vehicle types used on those roads. Conducting such studies is very taxing if done manually, so we need to device an efficient and accurate way to come up with the results. The goal is to be able to count vehicles and individually study each one. Often, classifying vehicles into several categories is also required. Another issue is being able to tell what cars too what turn on an intersection to collect possible usage statistics of this particular intersection for studies that will improve it later in the future. When such studies are currently carried out, there is no completely automated method to reach the desired results. In addition to using cameras, some studies are conducted using ground loops which can point out where cars are going. Such studies usually have around 90% accuracy, but can only identify vehicle paths. Problems faced when attempting to collect information about urban traffic mainly arise in two stages. The first stage is collecting the information on the road itself in a format that will allow some automation later on during the process. The second is mainly during the information processing. Common methods to doing this are using a combination of both manual and computerized work. Computers might attempt to do some image processing and the rest of the tasks are carried out manually, or, manual preprocessing is done, and the resulting data is fed to computers. We wish to automate the entire process in a very efficient and low cost way. Automating this task requires that we decide on three main criteria: camera

placement, vehicle characteristics, and matching the collected results to the preset characteristics (image recognition). We will discuss each of the three and mention how we will go about selecting each one and why.

Chapter 3

Related Work

3.1 3-D Object Recognition from 2D Images using Geometric Hashing [10]

In this paper, a general technique for model-based recognition is discussed, called Geometric Hashing. Its purpose is to identify an object in the scene, together with its position and orientation. This technique is based on an intensive preprocessing stage, done off-line, where transformation invariant features of the models are indexed into a hash table. The algorithm stands out for its high inherent parallelism and its ability to deal with occluded scenes. This paper focuses on the use of Geometric Hashing for the case of 3D object recognition from 2D images. An efficient method to represent a 3D model by its 2D projections is proposed. Results are presented of experiments on random data and 3D objects. It has been found that distinguishing between different types of features in a model or scene results in a very efficient implementation of Geometric Hashing using a multi- dimensional hash table.

3.2 Traficon – Video Detection Solutions For Traffic Data Acquisition [11]

Unlike other methods that we will look at, Traficon has dedicated hardware for its image processing (note: this is a commercial system so little information is available in the open). The key factor in a Traficon detection system is the VIP (Video Image Processor), a standard detector board on which several types of detection software can be run. The video signal from the camera monitoring the traffic is used as input for the detection unit. Detection lines are superposed onto the video image. Vehicles crossing these lines are

detected. The VIP analyzes the video images to generate traffic data and alarms. Detector boards are grouped in rack systems. Dedicated boards handle compression of images and transmission of data, alarms and images. Communication interfaces link the Traficon video detection product range with different types of communication networks: direct line, telephone lines, fiber networks and wireless communication. On the host computer at the control center, the WATTS (Wide Area Transport Telematics Server on PC) monitors the video detection systems, handling TCP/IP communication and database storage of data, alarms and images.

3.3 Robust Techniques for Background Subtraction in Urban Traffic Videos [12]

This paper talks about several algorithms that can be used for background subtraction and vehicle recognition in urban traffic videos. The algorithms discussed are frame differencing, medial filters, linear predictive filters, and a non-parametric model. The paper then compares the experimental results and gives comments about each algorithms advantages and disadvantages. The main conclusion that we can draw from this work is that recursive techniques can give better and more accurate results although they have proved to be more calculation intensive. This paper is also provides a summary of many algorithms that have been used for the task we are seeking to accomplish so it serves as a strong guide to what works well and what does not work well.

3.4 Neural Network with MIN/MAX Nodes for Image Recognition and Its Implementation in Programmable Logic Devices [13]

This paper discusses using MIN/MAX NN's for image matching based on a template

approach. The main approach is to use simple lookup tables and class patterns. This is mainly implemented in hardware for fast operation. The software system is split into two main parts, image preprocessing, and the neural network MIN/MAX implementation. This paper offers an interesting approach on how to take a learning algorithm, use it for image recognition, and translate it into a hardware circuit. Unfortunately, the paper does not go into too much detail about how it is implemented, nor does it give in depth results of the work.

3.5 Commercial Solutions

Several commercial solutions for vehicle identification exist on the market. Unfortunately, the level of detail provided for such systems is scarce. I will list some of those systems here:

- TEC Traffic Systems: Vehicle Detection Technology and Traffic Management Systems [19]
- Image Sensing Systems - Autoscope Video Vehicle Detection [16]
- JAMAR Technologies - Traffic Data Collection Equipment, Software and Supplies [17]
- Peek Traffic - Intersection Control, Vehicle Detection and Automated Red Light Enforcement Systems [18]

Chapter 4

Proposed Solution

In order to obtain optimum results, we need to decide on conditions and tweak them to yield values that can be easily used by our algorithm.

4.1 Camera Placement

Deciding where we place our cameras and at what angles they view the roads and vehicles is very crucial. Proper camera placement can ensure accurate results which require less processing time. If the videos captured from our cameras conceal the target vehicles or show them in angles which make identification difficult, the time required to categorize those vehicles will greatly increase and the accuracy of the results might drop. Possible good positions for placing our cameras are horizontal and vertical positions. A camera that has an angle of vision that is horizontal and perpendicular to the road allows for a fairly accurate side-view of the vehicles. We favor side-views because they allow us to get a feel for the vehicles' height and length; two very important and decisive factors in identifying the vehicle. The limitation of having a side-view, however, lies in the fact that it is not very well suited for multi-lane streets. A multi-lane street can have up to three or four cars standing side by side (in parallel). This automatically tells us that we need to deal with rows and columns of vehicles. To combat this limitation, we can place a camera which can give us a top-view of the parallel vehicles. A top-view helps us in identifying the width and length of a vehicle and can also contribute in giving us certain numerical statistics that we might want to collect about the street. Combining both, a side-view and

a top-view together will definitely yield much better results and will allow us to match vehicles more accurately.

4.2 Vehicle Categories

The second step in identifying vehicles is to have a database with vehicle details. Details we care about are typically the vehicle's length, height, width, and sometimes the wheel size. The first three are very important because they are determining factors which will identify the vehicle's category (car, van, bus etc...). The fourth, wheel size, can be uses as a sub-category for further identifying vehicles with similar types. We chose the wheel because it is a circular shape that can easily be identified in the foreground image. We plan to set up this database of vehicles in multiple stages. The first stage will involve taking pictures of vehicles and producing outlines. Those outlines will then be used in our image matching techniques. We will divide vehicles, to start with, into some rough categories and match against them. For further fine tuning, we can ask car manufacturers for the dimensions of their cars and each class and continue refining the database based on the obtained results.

4.3 Image Recognition

Having acquired the images, and set up a reference database, the third step to be able to identify and match those images against the database so we can come up with the results we need. The image recognition process is divided into several stages that will be explained in depth in the following sections. The first stage involves identifying when we should be looking for a vehicle. The second stage is applying our SOM algorithm.

Chapter 5

Image Recognition and Classification

5.1 Kohonen Networks and Self Organizing Maps

Our image processing techniques will rely on using Kohonen Networks (SOMs) [7, 8] and artificial intelligence methods. Unsupervised learning allows the network to find its own energy minima and is therefore more efficient with pattern association. The SOM is an algorithm used to visualize and interpret large high-dimensional data sets. Typical applications are visualization of process states or financial results by representing the central dependencies within the data on the map. The map consists of a regular grid of processing units, "neurons". A model of some multidimensional observation, eventually a vector consisting of features, is associated with each unit. The map attempts to represent all the available observations with optimal accuracy using a restricted set of models. At the same time the models become ordered on the grid so that similar models are close to each other and dissimilar models far from each other. Fitting of the model vectors is usually carried out by a sequential regression process, where $t = 1,2,...$ is the step index: For each sample $\mathbf{x}(t)$, first the winner index c (best match) is identified by the condition:

$$\forall i.\, ||\mathbf{x}(t) - \mathbf{m}_c(t)|| \leq ||\mathbf{x}(t) - \mathbf{m}_i(t)||.$$

After that, all model vectors or a subset of them that belong to nodes centered around node $c = c(\mathrm{x})$ are updated as

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(\mathbf{x}),i}(\mathbf{x}(t) - \mathbf{m}_i(t)).$$

Here $h_{c(x),i}$ is the ``neighborhood function", a decreasing function of the distance between the i^{th} and c^{th} nodes on the map grid. This regression is usually reiterated over the available samples. The network is trained by presenting it with random points. The neuron that has the largest response is reinforced by the learning algorithm. Furthermore, the surrounding neurons are also reinforced (this is explained in much greater depth later). This has the effect of "pulling" and "spreading" the network across the training data. Figure 1 shows how we can map the network onto a grid (and the learning process) while Figure 2 shows how we can map the network to a bitmap.

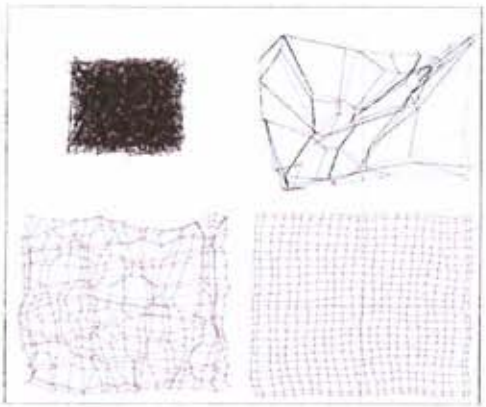

Figure 1: TL: Initial iteration, TR: 100 iterations, BL: 200 iterations, BR: 500 iterations.

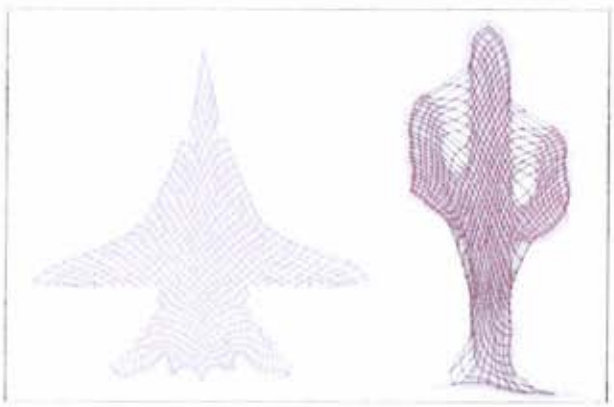

Figure 2: Mapping a Kohonen Network to a bit-mapped image.

5.2 Applying SOMs to a Three Dimensional Vector

The first application to SOMs we conducted was applying them to a 3-dimensional vector. In our case, we chose this vector to represent the red, green, and blue color factors of an image (RGB). We start the process by drawing a grid of tiles, each containing NxN pixels. Each tile is given a random set of three weights between 0.0 and 1.0. According to those weights, we then compute the RGB values for each tile and apply it to each pixel in the tile. The end result looks like this:

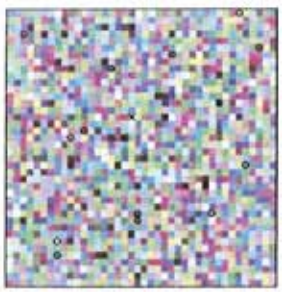

Figure 3: Randomized pixel tiles that will be fed to the SOM

Now that we have a set of randomly colored tiles, we can apply the SOM algorithm using a set of training vectors. We decided to generate a set of 10 training vectors which will be randomly given to the training program. The training set was obtained using the same pixel randomization techniques that allowed us to generate the previous tiles. A single iteration over the training loop will be called an epoch. With ever epoch, the best matching unit is calculated based in the Euclidean distance between the vector weights of the training vector and the current tile being inspected. We do this simply by applying the following:

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2}$$

This will effectively tell us how far or close a certain tile is from the current input. After iterating over the entire number of tiles, the one with the closest distance is selected as the best matching unit. After a BMU is selected, we need to influence all the surrounding tiles using a de-generative influence technique. This will influence each of the neighboring tiles in a certain radius of the BMU and a decreasing manner. Those that are closest to the BMU will get influenced the most, and as we move out on the radius the influence becomes smaller and smaller. The following image illustrates the BMU's neighborhood.

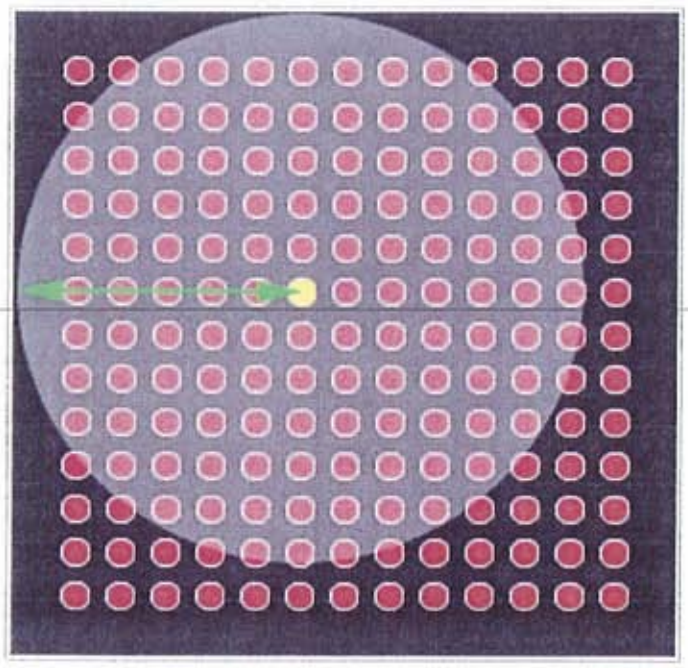

Figure 4: The selected BMU and its corresponding neighborhood.

The radius of the neighborhood is shrunk over time using the following equation:

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1, 2, 3...$$

The letter *sigma* denotes the width of the lattice at time *t0* and the letter *lambda* denotes a time constant. t is the current time-step (iteration of the loop). As we loop and shrink the radius, only those tiles that are within the radius have their weights adjusted. The amount by which we adjust the weights should become smaller as we get farther from the BMU. This is achieved by the following equation:

$$W(t+1) = W(t) + \Theta(t) L(t) (V(t) - W(t))$$

where t represents the time-step and L is a small variable called the learning rate, which decreases with time. L (decay function) and *theta* (amount of influence) can be computed as follows:

$$L(t) = L_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1, 2, 3...$$

$$\Theta(t) = \exp\left(-\frac{dist^2}{2\sigma^2(t)}\right) \qquad t = 1, 2, 3...$$

where *dist* is the distance a node is from the BMU and , is the width of the neighborhood function as calculated earlier. After training the network and updating the tiles visually, this is what we should get:

Figure 5: Trained network after SOM algorithm was applied.

At this point, we can input any vector to the network and it will be able to classify it into the correct location. Having taken the first step in applying Kohonen networks, the next one would be to increase the dimensionality of the vectors and choose proper features (RGB for pixels, other for vehicle images) that will allow us to identify and classify our input data. Figure 6 shows the SOM's pseudocode.

```
For t from 0 to m {
   sample = Get a random sample from training data
   winner = SOM node whose model vector is closest to the sample
   // Scale the winner's model vector toward the sample
  scale(winner, sample, winner)
  //Scale the winner's neighbors toward the sample
   For each node in Neighborhood(winner)   scale(node, sample, winner)
}
scale(node, sample, winner) {
   // Move the node's model vector toward the sample by a factor that depends on a time
// dependent learning rate L and the node's  distance from the winner
   node.model += (sample .  node.model)*(D – distance2D(winner,node))
}
```

Figure 6: SOM algorithm

5.3 Applying SOM's to High Dimensionality Vectors

Applying SOM's to pixels is not a very difficult task because in themselves, pixels have enough data (RGB) that allows us to easily identify and classify them. Our next task is a bit more difficult because when dealing with complete images, we need to decide on the characteristics that will be used. Those characteristics need to be extracted and averaged out. This phase is mainly a preprocessing stage that will then be followed by applying the SOM algorithm we discussed above. To begin, we need to do some image processing on the images so that we can create a feature vector for each image. Two important features in any image are color and texture. Although in our case color is not very important, it should still be taken into consideration because it will become useful at one stage or the other. To extract color information, we tried a number of different color schemes: gray, RGB, HSL, YUV and CIELAB.

- RGB: Red, Green, and Blue. A method of describing the color values and color saturation of an image. RGB is the simplest way to represent each pixel of an image according to much of each color it contains. Typically, RGB values are between 0 and 255.

- HSL: Hue, Saturation, and Lightness. A method of describing any color as a triplet of real values. The hue represents the color or wavelength of the color. It is sometimes called tone and is what most people think of as color. The hue is taken from the standard color wheel and is thus calibrated in degrees about the wheel. Saturation is the depth of the color. It states how gray the color is. It is real valued

parameter from 0.0 to 1.0 with 0.0 indicating full gray and 1.0 representing pure hue. The lightness is how black or white a color is. It also ranges from 0.0 to 1.0 but with 0.0 representing black and 1.0 white. A lightness of 0.5 is pure hue.

- YUV: The YUV model defines a color space in terms of one luminance and two chrominance components. Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components. YUV signals are created from an original RGB (red, green and blue) source. The weighted values of R, G and B are added together to produce a single Y signal, representing the overall brightness. The U signal is then created by subtracting the Y from the blue signal of the original RGB, and then scaling; and V by subtracting the Y from the red, and then scaling by a different factor.

- CIELAB: (Commission Internationale d'Eclairage) CIELAB is the most complete color model used conventionally to describe all the colors visible to the human eye. The three parameters in the model represent the luminance of the color (L, L=0 yields black and L=100 indicates white), its position between red and green (a, negative values indicate green while positive values indicate red) and its position between yellow and blue (b, negative values indicate blue and positive values indicate yellow).

From the aforementioned color description methods, HSL fits our needs the best because it allows us to carefully find information about shading that will make texture recognition easier and more accurate. For textures, we need to determine how much a pixel differs from its neighbors. If the pixel was mostly like its neighbors, then the image has low

texture. If the pixel was very different than its neighbors, then texture is high, and the image was probably noisy. For both color and texture, two measurements are conducted: histogram and area. The histogram is a global operation that distributes the values for the entire image between a continuum of minimum and maximum values. For example, a histogram for gray scale has a minimum value of 0 (black) and a maximum value of 255 (white). Between those values, we divide it up into 16 buckets. Each pixel would then be dropped into one of those buckets, incrementing its count. If the image is generally light, then the buckets closer to 255 would have the highest counts. The bucket counts of the histogram could then be used as a feature vector of length 16. The other measure used area. This basically involves dividing the image up into 3x3 section matrices. Each section has its average color and texture value calculated. This list represents a feature vector of length 9. The figures below show these concepts. Figure 7 shows the histogram and second Figure 8 shows the area concept we discussed.

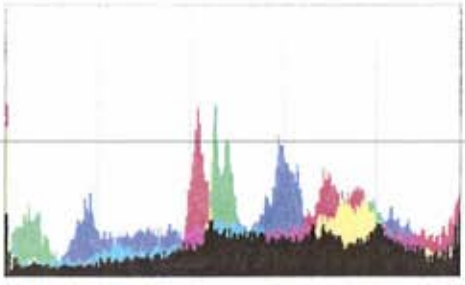

Figure 7: Typical histogram of an image

Figure 8: Area concept distribution used in our algorithm

In order to properly identify the vehicles in the images we are analyzing, we need to assign several other attributes to our feature vectors. Two very important measures to have are the vehicles length and height. A third attribute we can use is wheel size (although this is still being tested at this stage). Applying our algorithm with all of the mentioned attributes allows us to classify vehicles both quickly and accurately after the network is trained. By using our texture calculation idea mentioned above, and by doing a simple background subtraction algorithm [20] on the images, we can get an approximate width and an approximate height of the vehicle in question. The height and width are also added into the feature vector. Now that the feature vector can represent the vehicles, all we need to do is to hook it into our algorithm just like we hooked the feature vector for the RGB pixels. In the next section, we will illustrate some experimental results and show the accuracy of our work.

Chapter 6

Experimental Results

6.1 Experimental Procedure

We conducted our experimental work in two main steps. The first step was to apply the algorithm to an arbitrary set of pixels. Pixels are simple in nature (3-dimensional RGB vectors) and can accurately and easily show us if our algorithm is performing what it was designed to do. After being satisfied with the results we obtained from the first step, we moved on to the next more important step, identifying and classifying complete vehicle images. The next step consists of identifying a single image, and then showing how this image can be classified appropriately with similar images.

6.2 Application to Pixels

When we applied our techniques to pixels, the results were very accurate. When using pixels, we can mainly change two figures, the first being the number of pixels (either per block or in total) and the second being the number of colors. If all of our pixels have a limited range of colors (16, 32, 64...) then the smaller the color range, the faster we can train the network, then the faster we can classify new incoming pixels. In our test runs, we considered both bases, when we have a limited number of colors, and when the color range was open. The following figure (Figure 8) shows the results when using 9 colors only after processing input like that shown in Figure 3.

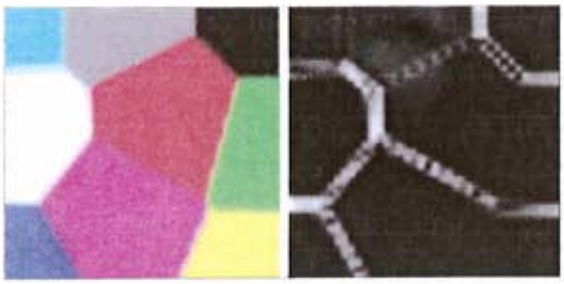

Figure 9: Applying our algorithm to pixels with 9 different colors

From the previous figure, we can see that since we have limited the pixels to only 9 colors, the border lines (showed in the differenced and negatived version of the result, to the right) are very clear and distinct. This will not be observed when we allow the pixels to have any random RGB value. We will observe gradient like behavior in the results. This is illustrated in the following figure.

Figure 10: Applying our algorithm to pixels with completely random colors.

When we randomly initialize the RGB values for every pixel, we have very little similarity in every actual final color for each pixel, and this is where the power of our algorithm shows. Even though the pixels are truly very different, the algorithm manages to group them together extremely well. To the human eye, this means the the greens will be next to each other, and so will the blues, reds, pinks etc. The final outcome, as seen in Figure 9, is that we have central areas with high concentrations of certain colors followed by gradual changes in the color in a giving a gradient, until we start reaching another high

concentration area. The other interesting observation that we can make from Figure 9 is that the segmentation image (the one to the right) is almost impossible to look at. There are no clear segmentations and borders as observed in Figure 8 (this is actually the outcome we expect!). Because our colors fade into one another, it is virtually impossible to define border lines where we say that this is the end of a certain cluster and the beginning of another. This is both good and bad. It is good because it means that we now have centers, and the farther you get from the center, the farther you get from its characteristics. This allows us to define certain thresholds when doing searches or when fitting new pixels (in the general case, new N long vectors). The disadvantage is that it is not as clear cut as the first approach of limited the values for the input vector (limited the RGB values for our pixels). Now in reality, when clustering and identifying certain objects, the problem never involved dealing with an infinite set of possibilities because this would make things infinitely complex and accurate results can not be guaranteed. We did, however, prove that even with completely random sets with infinite possibilities, our algorithm still works very well. The following table shows some numerical figures that tell us more about the number of pixels and the time it takes to identify and position a certain pixel.

Table 1: Results of algorithm applied to pixels.

Pixels / Node	***Number of Nodes***	***256***	***1024***	***4096***	***16384***
4		0.034s	0.091s	0.141s	0.831s
9		0.041s	0.106s	0.182s	0.922s
16		0.053s	0.110s	0.201s	0.974s
25		0.059s	0.131s	0.214s	1.014s

The results in Table 1 do not include the rendering time to draw the results in real-time, they reflect the code calculation time required to perform the algorithm and to identify a pixel and where it belongs. The rendering time is dependent on factors like the graphics subsystem being used and to some extent the video hardware in use.

6.3 Application to Vehicle Images

When dealing with vehicle images, applying the algorithm is not as straight forward as it was on simple pixels. The first distinction we have to make here is whether to use colors or not. When we use colors, we will have an additional feature in our feature vector. For our current scenario, we do not care about classifying and identifying vehicles by color, so we will make sure that we gray scale the images before we use them. As we have previously discussed, our feature vector will be based on histograms, textures, and vehicle information like width, height, and wheel base. To show how our algorithm applies to identifying vehicles, we will illustrate two main concepts: identifying the general outline of the vehicle in an image (which could be a frame in a video) and classifying it. Figure 10 will illustrate how extract the vehicle from its original image. A simple background removal algorithm [12] is applied as illustrated.

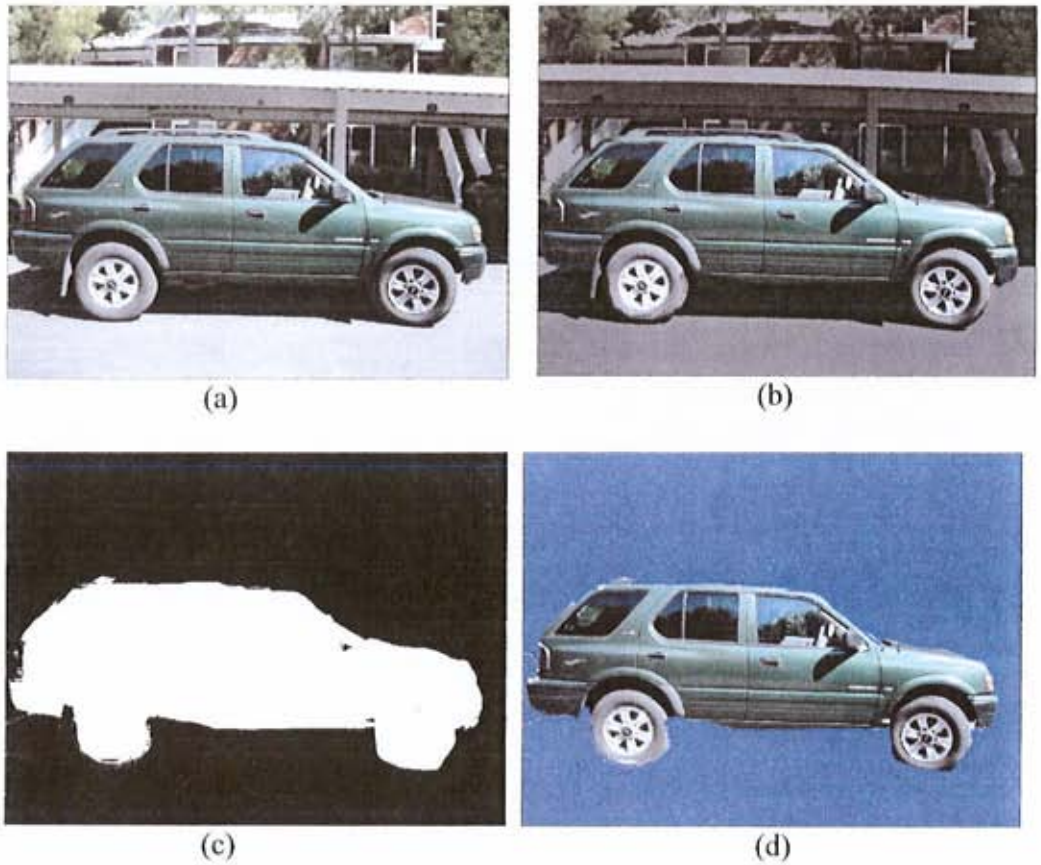

Figure 11 (a) The image without background subtraction applied. (b) The image with the background mask faded out (but not deleted yet). (c) The image with the generated mask, 2 colors. (d) Backgrounds mask subtracted and replaced by a solid color.

After obtaining the raw vehicle image which we want to identify and further classify, we can easily crop out the solid color to obtain an image which we can work with, as shown in Figure 11.

Figure 12: Final preprocessing stages complete, cropped image (non-gray scale).

In its cropped state, we can easily measure the vehicle's width and height, and pass it through our histogram and texture filters in order to fill up our feature vector. The preprocessing stage on a 640x480 pixel image takes 0.121 seconds. This is the time required to take an image through all the stages in Figures 10 and 11 with an additional 0.018 seconds for gray scale transformation. In order to train the network, we provided around 250 sample images and ran the algorithm around 1000 times. The training time is not very important because it is performed when the system is off-line. What we do care about is the average time to identify and categorize an arbitrary image of a vehicle because this might be done in real time with some buffering time (which will give us a temporary delay, hence would allow for longer identification times to be acceptable). Table 2 will illustrate the time required for classifying a 640x480 pixel image (gray scaled) in a trained network with a certain number of nodes.

Table 2: Total nodes per network versus the time it takes to classify an image

Total Nodes	*256*	*1024*	*4096*	*16384*
Time	0.231s	0.408s	0.865s	1.107s

Again, as previously mentioned for Table 1, the results for Table 2 do not include the time required to do any disk I/O or rendering to the graphics subsystem. Disk I/O is highly dependent on the type of disk we are using, and whether we are taking any shortcuts or optimizations (picking up the image data directly from memory before it is written to disk for example, this would half our disk I/O operations). We can make an interesting deduction about the times reported through Table 2, when moving from 256 to 16384 nodes (64 times more), we only have a time change of 0.876 seconds (or 4.792 times more). This shows that even though the number of nodes was greatly increased, the time required to classify the image was not changed a lot!

6.4 Discussion and Comparison of Results

Being able to identify the core image is only the first part of the work the we need to do. The second part if to be able to classify that image and place it with the images that is is closest to. After training the network, we will end up with a node -grid. Each node represents similar images. At the edges of the grid, vehicles will be highly related to each other, and as we move towards the inside (or towards another edge) we will move farther and father from the original set of images we started with. For example, one edge of the grid might represent larger 4x4 cars (Jeeps, Pathfinders, Pajeros etc.) while another edge might represent smaller cars (Renault 5, Mini Cooper, Micra, etc.) and on the line

connected both edges will show us a gradient of values. This gradient will start with large 4x4 cars and end up with smaller cars forming a transition from the first group to the other. Figure 12 will illustrate the visualized grid containing images that have been classified together (4x4 vehicles).

Figure 13: Visual grid showing similar vehicles

By looking at this grid, we see that the results are indeed accurate and vehicles of similar nature have been classified together. In addition to that, our runtime values are small which can only say that our algorithm not only is accurate, but is also fast. Another area of the grid shows a completely different cluster with vehicles different that those in Figure 13. All vehicles in Figure 14 are similar; this further proves the accuracy of our results.

Figure 14: Visual grid showing similar images on a different edge

6.5 Hardware Notes

Both sets of results were benchmarked on an Intel Pentium4-M (Centrino) CPU clocked at 1.86Ghz and took advantage of of level 3 optimizing techniques in the GNU C Compiler (GCC using the -03 optimization technique) along with loop-unrolling compiler optimizations where applicable. The tests were also carried out on an ARM-200Mhz embedded CPU (Sharp Zaurus PDA) but the results have not been documented here. It was apparent that since the Centrino mobile CPU architecture could be used (low amounts of heat dissipation, low power consumption, available wireless network) on site, then that set of tests would be more logical to use.

Chapter 7

Conclusion and Future Work

The ability to identify and classify images is a very important problem with a very wide range of applications. In our paper, we showed that we can use an intelligent process that does not require user input to train, identify, and classify images of vehicles. One of the most important results obtained in this work was the fact that greatly increasing the number of nodes in our network does not exponentially increase the time required to classify vehicles. This is extremely important because it guarantees that the algorithm will work equally well on both small and large datasets. In the future, we would like to develop this algorithm further with more information in each image's feature vector, and we would like to provide an interface that would make training the algorithm and viewing the results much easier. Presently, this is all part of a series of non-visual operations. Turning this project into a fully-featured application will prove very useful to a lot of people with interest in this field.

Bibliography

1. Minnesota Department of Transportation, "Field Test of Monitoring Urban Vehicle Operations Using Non-Intrusive Technologies", 1997, http://www.dot.state.mn.us/guidestat/nitfinal/part1.htm
2. Y. Hofman, Hi-Tech Solutions, "License Plate Recognition - A Tutorial", 2001, http://www.licenseplaterecognition.com/
3. Citilog Video Detection Systems, "Citilog - The Only video Detection System Without Configuration", 2005, http://citilog.fr/en/applications/intersection.php
4. K. Kockelman, R. Shabih, "Effect of Vehicle Type on The Capacity Of Signalized Intersections: The Case Of Light-Duty Trucks", 1999, http://www.ce.utexas.edu/prof/kockelman/public_html/ASCELDTShabih.pdf
5. U.S Department of Transportation, "Traffic Analyssis Toolbox Volume III: Guidelines for Applying Traffic Microsimulation Modeling", 2002, http://www.ops.fhwa.dot.gov/trafficanalysistools/tat_vol3/sect3.htm
6. Machine Vision, CSIRO Manufacturing & Infrastructure Technology, "Safe-T-Cam Highway Vehicle Identification Network", 2000, http://vision.cmit.csiro.au/project/stc/
7. M. Taner, "Kohonen's Self Organizing Maps and their use in Interpretation", 1997, http://www.rocksolidimages.com/pdf/kohonen.pdf
8. H. Ritter, K. Schulten, "Kohonen's Self Organizing Maps: Exploring their Computational Capabilities", 1998, http://www.ks.uiuc.edu/Publications/Papers/PDF/RITT88A/RITT88A.pdf
9. O. Sarzeaud, Y. Stephan, "Data Interpolation Using Kohonen Networks", 2000,

http://www.ec-nantes.fr/ectia/PUBLIS/IJCNN00.pdf

10. D. Gavrila, F. Groen, "3D Object Recognition from 2D Images using Geometric Hashing", , 1992, http://www.gavrila.net/Publications/pr92.pdf
11. Traficon NV, "Traficon - Video Detection Solutions for Traffic Data Acquisition", 2005, http://www.traficon.com
12. S. Cheung, C. Kamath, "Robust techniques for background subtraction in urban traffic video," Video Communications and Image Processing, SPIE Electronic Imaging, San Jose, January 2004, http://www.llnl.gov/casc/sapphire/pubs/UCRL-CONF-200706.pdf
13. R. Holota, "Neural Network with MIN/MAX Nodes for Image Recognition and Its Implementation in Programmable Logic Devices.", In: Proceedings of IEEE International Conference on Computational Cybernetics, Siófok, Hungary, 2003, ISBN 963-7154-18-3.
14. S. Klein, "Traffic Cameras: Protecting Our Streets Or Invading Our Privacy", 2001, http://cseserv.engr.scu.edu/StudentWebPages/SKlein/ResearchPaper.htm
15. I. Oliveira, N. Correia, N. Guimaraes, "Image Processing Techniques for video content extraction", Fourth Delos Workshop for Image Indexing and Retreival, San Miniato", 1997, http://www.ercim.org/publication/ws-proceedings/DELOS4/oliveira.pdf
16. Image Sensing Systems Inc., "Image Sensing Systems - Autoscope Video Vehicle Detection", http://www.imagesensing.com/
17. JAMAR Technologies, Inc, "Traffic Data Collection Equipment, Software and

Supplies", http://www.jamartech.com/

18. Peek Traffic Corporation, "Intersection Control, Vehicle Detection and Automated Red Light Enforcement Systems", http://www.quixtraffic.com/
19. TEC Traffic Systems, "Vehile Detection Technology and Traffic Management Systems", http://www.tectraffic.nl/