Appears in the proceedings of *Computer Architectures for Machine Perception, 1993* (CAMP ‘93).

A Massively Parallel Road Follower

Todd M. Jochem
tjochem@ri.cmu.edu
The Robotics Institute
Carnegie Mellon University

Shumeet Baluja
baluja@cs.cmu.edu
School of Computer Science
Carnegie Mellon University

Abstract

In recent years, significant progress has been made towards achieving autonomous roadway navigation using video images. However, none of the systems developed take full advantage of all the information in the 512x512 pixel, 30 frame/second color image sequence. This can be attributed to the large amount of data which is present in the color video image stream (22.5 Mbytes/sec.) as well as the limited amount of computing resources available to the systems. We have increased the available computing power to the system by using a data parallel computer. The system presented in this paper uses substantially larger frames and processes them at faster rates than other color road following systems. This is achievable through the use of algorithms specifically designed for a fine-grained parallel machine as opposed to ones ported from existing systems to parallel architectures. The algorithms presented here were tested on 4K and 16K processor MasPar MP-1 and on 4K, 8K, and 16K processor MasPar MP-2 parallel machines and were used to drive Carnegie Mellon’s testbed vehicle, the Navlab I, on paved roads near campus.

1.0 Introduction

In the past few years, the systems designed for autonomous roadway navigation using video images have relied on a wide variety of different techniques [Thorpe, 1991]. The techniques range from neural network architectures that learn the appropriate road features required for driving [Pomerleau, 1991], to systems which find road lines and edges based on predefined road models [Kluge, 1992]. Many of these systems have been able to drive vehicles at high speeds, in traffic, and on real highways. However, none of these systems have been able to use all of the information present in 512x512 pixel, 30 frame/second color input image stream. This can be attributed to the limited computing power available to the systems in the face of the sheer magnitude of the data presented to them (22.5 Mbytes/sec.).

Several methods can be used to handle these large sets of data which are present in road following tasks. One method is to use gray-scale images instead of color images. Gray-scale images reduce the amount of data which must be processed by two-thirds. However, even at this lower data rate, processing every pixel is still extremely difficult, and subsampling [Crisman, 1990] and windowing techniques [Turk, 1988] are frequently used to try to achieve frame rate processing on the incoming color data. Another method is to preprocess the color image, projecting the 3D color data into a single value at each pixel [Turk, 1988]. This method, like windowing, has the desired effect of reducing the amount of data which must be processed by the system. Defining regions of interest and only processing the pixels within them is another possibility and has been explored by [Kluge, 1992] and [Dickmanns, 1992]. A final method, and the one we chose to use, is to increase the available computing power by using a massively parallel, single instruction, multiple data (SIMD) computer. The system presented here uses substantially larger frames and processes them at faster rates than other color road following systems. The improved rates are achieved through the use of algorithms specifically designed for a fine-grained parallel machine. Although higher resolution does not directly lead to increased driving accuracy, systems which can handle these resolutions at faster rates have a clear advantage in other domains were fine features are more important. It is our hope that the rudimentary algorithms presented in this paper can be extend to other such vision tasks.

2.0 Overview of System

As is the case in nearly all vision system, a number of assumptions have been made about the environment which simplify the problem. Two assumptions which our system makes are the defining road model and the relationship be-

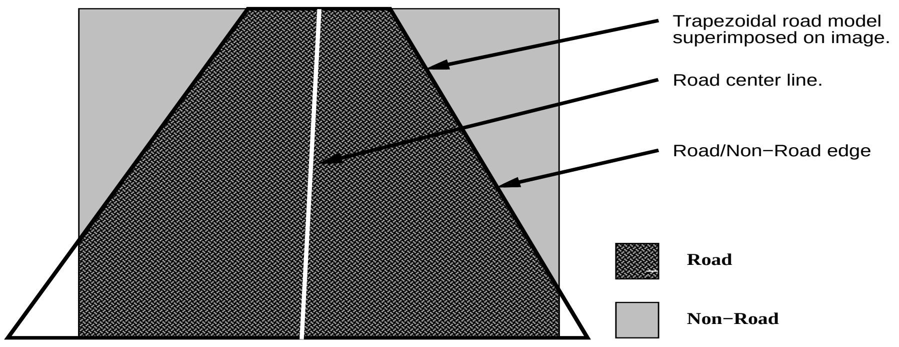


Figure 1. Trapezoidal road model.

tween the colors in each scene and their placement within the color space.

The first assumption is that of a trapezoidal road model. In this model, the road is described in the image as a region bounded by four edges. Two of these edges are defined as the location where road and non-road meet while the other two are the top and bottom of the image itself. These four edges constitute the trapezoid. The road center line is defined to be the geometric line which bisects the trapezoid vertically. For a graphic representation of this model see Figure 1. In addition, our system assumes that the road remains a constant width through all of the input images. This constraint means that although the trapezoid of the road may be skewed to the left or right, the top and bottom edges remain a constant length.

Color clusters play an important role in our system. A color cluster is defined as the group of pixels in the 3D (red, green, blue) color space which are closest to a particular central, or mean, pixel value. In our representation, closeness is defined by the sum of squared difference between a pixel's red, green, and blue values and a central pixel's red, green, and blue values.

An assumption, based on this concept, is that the colors in the image can be adequately represented, or classified, by a small (between 5 and 30) number of clusters. In our system, the ability of a set of clusters to classify an image is measured by how well the mean values of the clusters of the set can reconstruct the original input image. This will be discussed further in Section 3.3.

A third assumption is that any particular cluster can only correctly classify pixels in the road or the non-road portions of the image; it cannot correctly classify pixels in both. Because the road and non-road pixels of typical road images, when plotted in color space, form largely disjoint sets, and because of the nature of the algorithm which partitions color space into clusters, this has proven to be a reasonable assumption. See Figure 3.

The system we have developed is an iterative, three step procedure in which every pixel in the input image is first classified by a color cluster, then labeled as road or non-road, and finally used to find the center line of the road. The world coordinates of this line are passed to the vehicle controller which guides the vehicle on the specified path. This process is supplemented by an initialization phase which occurs only once on the first image the system processes. A high level view of the system architecture is shown in Figure 2.

This system is pixel driven; thus all computation can be done very efficiently on a massively parallel processor array. Instead of mapping existing road following algorithms to the processor array, we integrated algorithms which could exploit the tightly coupled processor array and take advantage of the limited capabilities of the individual processors. The resulting system is fully parallelized, and performs comparably to the state-of-the-art.

Our system is composed of three main algorithms, all of which are parallelized on the processor array. The three parts are a **clustering algorithm**, a **combining algorithm**, and a **road finding algorithm**. The clustering algorithm is an iterative procedure which uses a parallel competitive learning implementation of the isodata clustering algorithm [Ball, 1967] to find the mean red, green, and blue values for each color cluster in the first input image. This will be described in greater detail in the next section.

The combining algorithm uses a single perceptron which is trained to correctly determine whether a pixel in the input image is a road pixel or a non-road pixel. This is accomplished by using information derived from classification of the pixel using the means developed in the clustering algorithm. The clustering and combining network representations will be described later.

Finally, after the road has been cohesively segmented from the input image, a technique which extracts the center line of the road (or any safe path) from the input image is needed. In our system, a parallelized Hough transform is applied in which the topology of the processor array is used to determine the location of the center line of the road.

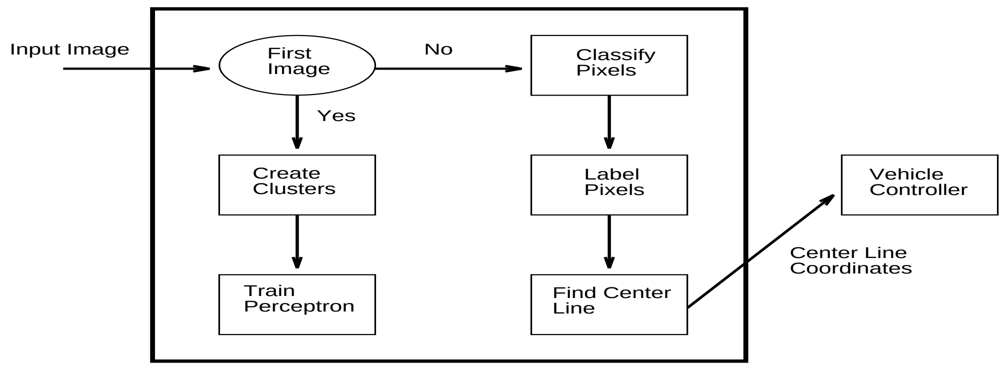


Figure 2. Block diagram of system architecture. Steps located inside the large black box are computed in parallel on the processor array.

3.0 The Clustering Algorithm

If every pixel in the input road image is plotted in a three dimensional space with red, green, and blue being the axes (the color space), clustering of pixels occurs. These clusters often correspond to regions in the image. For example, in an image of a road through a grassy field, two clusters would be expected in the color space, one cluster for the road and one cluster for the grass. See Figure 3. If the mean color value of each cluster were computed, the cluster centered around the road pixels would be gray while the one centered around the grass would be green. By assigning all pixels in the input image which are 'closest' to the road mean as 'road' and all those that are 'closest' to the grass mean as 'non-road,' it is possible to segment the image so that the road can be easily discriminated from the grass. The clustering algorithm implemented for this system is more formally known as **competitive learning**, and is described with more mathematical rigor in the next section. The implementation of this learning algorithm in our system is detailed in Section 3.2.

3.1 Competitive Learning

Competitive learning is an unsupervised connectionist learning technique which attempts to classify unlabeled training data into significant clusters or classes in a feature space. In this framework, many units compete to become active, or turn on, when presented with input training data. The single unit which wins is allowed to 'learn.' Because this technique is unsupervised, it requires no teaching signal to tell the units what to learn. The units find relevant features based on the correlation present in the unlabeled training data.

A typical competitive learning network is composed of several output units, O_i, which are fully connected to each input through a connection weight. To find the activation of the unit, the input activations, ξ_j, are multiplied by the corresponding connection weights, $w_{i,j}$, and summed to produce an output activation. Mathematically,

$$O_i = \sum_j w_{i,j}\xi_j = w_i\bar{\xi}$$

Note that $w_i\bar{\xi}$ is the vector notation representation of the sum. The output unit which has the largest activation is declared the winner. If weights for each unit are normalized, then the winning unit, denoted with by $i*$, can be defined as the unit whose inputs most closely match its weights. This can be expressed as:

$$\left|w_{i^*} - \bar{\xi}\right| \le \left|w_i - \bar{\xi}\right|$$

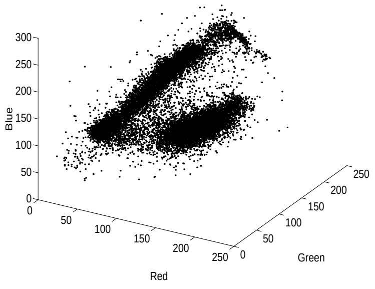
Because the weights in each unit start out as random values, a method is needed to train them. The intuitive idea is to learn weights which will allow the unit to correctly classify some portion of the feature space. The technique that is typically used, and the one used in our system, is to move the weights of the winning unit directly towards the input pattern. This method is known as the **standard competitive learning rule**. This rule can be expressed as:

$$\Delta w_{i^*,j} = \eta\,(\xi_j^k - w_{i^*,j})$$

where η is the learning rate and k is the training pattern [Hertz, 1991]. We have adapted the standard competitive learning rule to the parallel paradigm and have developed the **parallelized standard competitive learning rule**. Mathematically speaking, this rule can be expressed as:

$$\Delta w_{i^*,j} = \frac{\eta}{u}\sum_{k=0}^{u}(\xi_j^k - w_{i^*,j})$$

where u represents the total number of training patterns classified by unit i. An important feature to notice about the parallelized standard competitive learning rule is that the contribution of each training pattern to the overall weight change can be computed independently. This means that the contribution can be calculated concurrently for many training patterns at once. It is this type of algorithm which can take advantage of a fine-grained parallel machine.



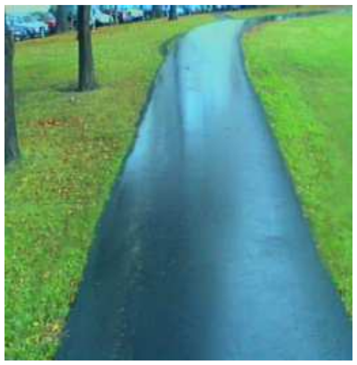


Figure 3. Color space for a typical road scene.

3.2 Pixel Clustering

Our system typically uses five color clusters. These clusters are global structures which any of the processors can access. We have also experimented with up to 30 clusters; however, we have found that using only five provides a good balance between effectiveness and computational load. The first step of the algorithm is to randomly initialize each color cluster's mean red, green, and blue values to a number between 0 and 255. Next, the first image of the color video stream is mapped, using a two dimensional hierarchical mapping scheme, shown in Figure 4., onto the processor array. With each processor containing a small block of the image, the clustering algorithm can be started. The clustering algorithm is a two stage iterative process consisting of clustering and evaluation steps. *In this algorithm, clustering is done with respect to a pixel's location in color space,* ***not*** *it in regard to whether the pixel represents road or non-road*.

For every pixel on each processor, the cluster to which it is closest is determined. 'Closeness' is computed as the sum of squared difference between the current pixel's red, green, and blue values and the mean red, green, and blue values of the cluster to which it is being compared. The cluster for which the sum of squared difference is smallest is the 'closest' cluster and we say that the cluster classifies that pixel. Each processor has five winning cluster counters, one corresponding to each cluster. The value of each winning cluster counter represents the number of pixels which are classified by the respective cluster. Because the clustering algorithm is based on competitive learning, the red, green, and blue mean values of each cluster are represented as weights on input connections to units in the competitive learning network and each unit represents a color space cluster. (From this point, the term cluster will be used to identify the corresponding unit in the competitive learning network.) Learning takes place by adjusting these weights to more accurately reflect the actual mean red, green, and blue values of the cluster in color space. This is accomplished by adjusting the cluster weights toward the mean value of all pixels which the cluster has classified. To do this we must compute the difference between the current pixel value and the cluster mean pixel value and then adjust the cluster mean by some proportion of the computed difference. This is essentially the parallelized standard competitive learning rule.

In our system, there can be many pixels on a processor that are classified by each cluster. For every pixel that a cluster classifies, the difference described above is computed and added to a local (processor based) difference accumulator for the cluster. In addition, the local winning cluster accumulator that corresponds to the cluster is incremented. The difference accumulator consists of three sums, each corresponding to the red, green and blue differences. Once every pixel has been classified and its difference computed and stored, the values from all local winning cluster counters and difference accumulators are summed using a global integer addition operator and stored in global accumulators.

Now, the appropriate adjustments to the cluster weights are calculated. This is done by dividing each weight's difference value stored in the global difference accumulator for a particular cluster by the number of pixels that were classified by the cluster, given by the value in the global winning cluster counter. This value is essentially the average difference between this cluster's current weights and all pixels that it classified.

The only problem left to overcome is handling clusters which did not classify any pixels. This is a common occurrence in the first clustering iteration, as the weights are randomly initialized and can be set to values in the remote corners of the color space. To rectify this, instead of finding the cluster which is closest to a pixel value, we find the cluster which is closest to the one that did not classify any pixels. The sum of the squared difference between the weight (mean) values of these two clusters' is computed as described above. This difference is used to adjust the cluster which classified no pixels. This intuitive heuristic brings the remote clusters to more promising locations, and

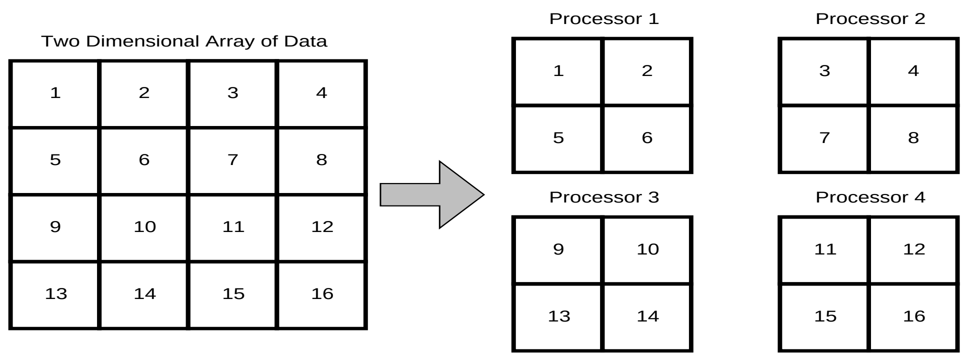


Figure 4. Two dimensional hierarchical mapping takes a 2D array of data like an image and maps it onto the processor array in a block-like fashion. (Image reproduced from MasPar MPDDL manual.)

because the difference is computed before cluster weights are updated, two clusters cannot have the exact same location. Although the method is very simple, it works well in our domain. At this point, all cluster weights are updated using the computed adjustments. Upon completion, the evaluation stage of the algorithm is entered.

Although the algorithm described in this paper uses five clusters, we have evaluated system performance using up to 30 clusters [Jochem, 1993, 2]. As is expected, as the number of clusters increases, the classification becomes more exact, and the final error in reconstruction is reduced. (Image reconstruction is discussed in the upcoming paragraphs.) The trade-off associated with a lower reconstruction error (i.e. better classification) is the time involved in processing more clusters. This will be discussed in further detail in Section 7.0.

3.3 Evaluation

In order to determine when the clustering process should stop (i.e. when the cluster weights have converged) a cluster evaluation process must be employed. We have developed a technique which allows the system to perform self evaluation. This technique is based on the ability of the cluster weights to accurately reproduce, or reconstruct, the input image. The self evaluation process begins by classifying every pixel in the input image according to the most recently updated cluster weights. As in the clustering stage, classification is done locally on each processor on the distributed input image using the previously described closeness measure. Once the closest cluster has been found, the pixel is assigned the red, green, and blue mean values of this cluster. (Remember, the mean values are described by the cluster's weights.) Next, the squared difference between the pixel's assigned value, given by the closest cluster mean, and its actual value, given by the pixel's value in the original input image, is computed and stored in local accumulators. After the squared differences have been calculated and stored for all pixels on each processor, a global average squared difference value is computed using a global floating point addition operator and knowledge of the input image size. If the change between the current average squared difference and the previous clustering iteration's average squared difference for each of the red, green, and blue bands is sufficiently small, clustering ends and the combining algorithm described in the next section begins. If the average squared differences are not yet sufficiently small, the clustering cycle described in the preceding paragraphs repeats using the current cluster weights. The evolution of the reconstructed image as competitive units learn cluster means is shown in Figure 5.

More rigorously, the evaluation process looks at the rate of change, or slope, of average squared difference values. When this slope becomes lower than a user supplied threshold for all three color bands, a minima is said to have been reached, and any further attempts at clustering could yield only minimal improvement. By using the rate of change of the average squared difference, the system can always converge to a locally (in the color space) good solution, regardless of how many clusters are available for it to use. It has been our experience that approximately 20 cluster-evaluate iterations are necessary for proper clustering of the initial input image.

In the current implementation of our system, the clustering algorithm is run on only the initial image of the color video stream. The correct cluster means may change in subsequent images. Nonetheless, through our experimentation on paved roads near campus, we have found that the change is too small to cause system performance degradation. This differs from results by [Crisman, 1990] and is potentially due to the greater number of pixels which are used to form the color clusters and the robustness of our combining algorithm. We have not yet looked deeply at this difference and it remains open for future study.

4.0 The Combining Algorithm

In order to determine where the road is located in the input

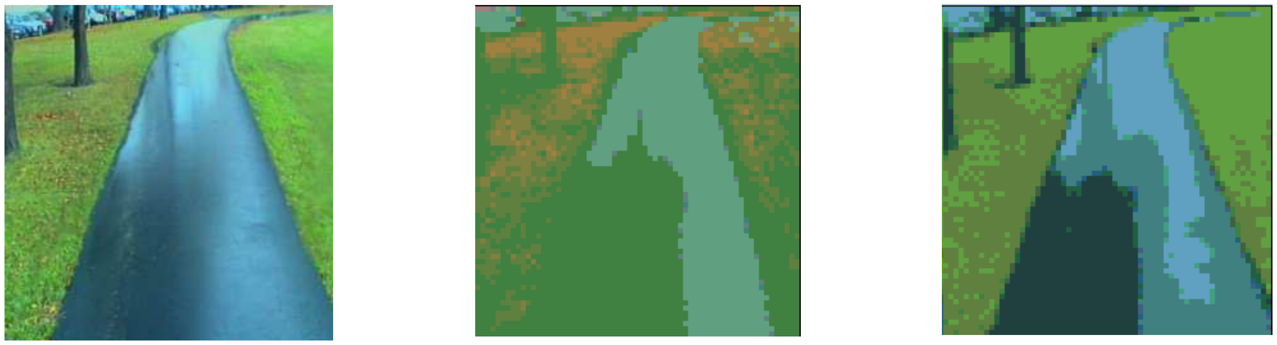


Figure 5. Evolution of reconstructed image as competitive units learn. As the clusters move in the color space, they more accurately represent the actual colors of the pixels. Shown above: original image, reconstructed image after 1 and 12 iterations.

image, it is necessary to separate the road from its surroundings. Although it is easy to assume that a single cluster will always classify the road, as the road is almost uniformly gray, and the rest of the clusters will classify non-road, this is not the case in practice. For example, consider the case of a shadow across the road. In this circumstance, two or more clusters converge to areas in the color space that corresponded to the brightly lit road, the shadowed road, and possibly to the minute color variations in the transition between light and shadow. Since different clusters classify the roadway in this example, a technique must be found to unify this information. No matter if a road pixel is in sunlight or shade, it should be correctly classified as part of the road. As with the clustering algorithm, the combining algorithm unifies the different clusters of the same type in a distributed manner by using every pixel in the input image. Another feature of this algorithm, similar to the clustering algorithm, is its self evaluatory nature.

At the heart of the combining algorithm is a single, global perceptron with a hyperbolic tangent activation function. A perceptron is a connectionist building block used to form Artificial Neural Networks (ANN). Artificial neural networks were chosen as the learning tool because of their ability to generalize. See [Jochem, 1993, 2] for a more complete description of Artificial Neural Networks.

4.1 The Combining Perceptron

The combining perceptron in our system is trained, by modifying its input weights, to differentiate road pixels from non-road pixels using information derived from knowledge of which clusters correctly and incorrectly classify a pixel in the input image. The output of the perceptron can be thought of as the 'certainty' of a particular pixel having a characteristic road color. The perceptron has one input weight for each cluster as well as a bias weight which is always set to 1.0. (Having a bias weight is a standard technique used in perceptron training.) It is important to note that the same global perceptron is used for all pixels on every processor.

Alternatively, probabilistic methods which take into account cluster size and location could be used to differentiate between road and non-road clusters. This approach has been explored by [Crisman, 1990] and will be discussed in greater detail in Section 8.0.

In order to bootstrap the system, the road region in the initial image is outlined by the user. From this original image, we can determine which colors should be classified as road and which as non-road. The outline also determines the trapezoidal road model. Because we are using a two dimensional mapping, the left and right edges of the outlined region can be superimposed onto the processor array. Processors which lie on these edges are marked, and ones which are between the left and right edges are said to contain road pixels while those outside the edges contain non-road pixels. This interaction only occurs once. The algorithm automatically proceeds to classify each pixel in the input image using the clusters previously formed. After a pixel has been classified, if it is located on a road processor (i.e. within the user defined road region), the perceptron is given a target activation value of 1.0. If it is not, the perceptron is given a target activation value of -1.0.

The input activation to each connection weight of the perceptron is determined by the following simple rule: if the connection weight is the one associated with the cluster that correctly classified the pixel, it is given an input activation value of 1.0, otherwise it is given an input activation value of -1.0. See Figure 6. Next, each input activation is multiplied by its corresponding connection weight. These products are summed together and this value is passed through the perceptron's activation function which yields its output activation. This process is known as forward propagation. The difference, or error, between the output activation and the target activation is computed. This error, along with the input activation to each connection, is used in the Least Mean Square learning rule [Widrow, 1960], so that each connection's weight adjustment can be computed. These adjustments are stored in a local accumulator. After all pix-

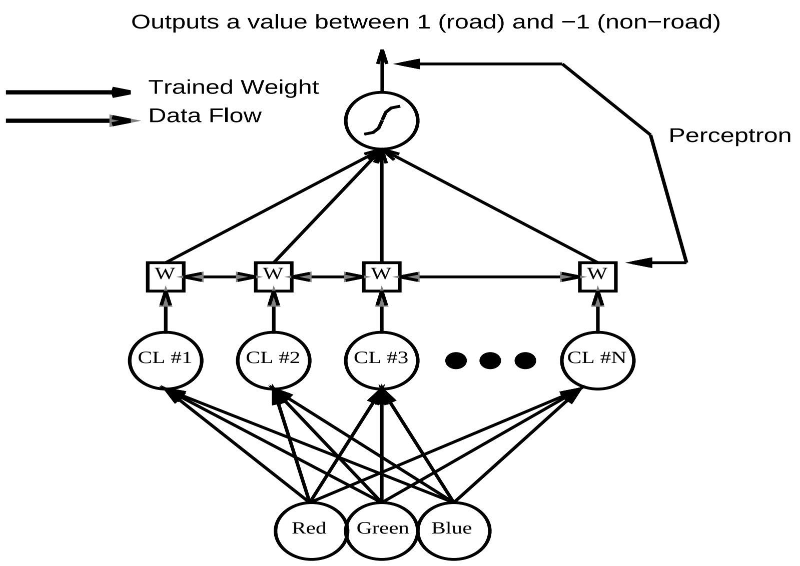


Figure 6. System Architecture. Each pixel is classified in parallel by competitive learning units. The winning unit has its output activation set to 1.0 while all others are set to -1.0. A perceptron classifies this combination of inputs as either road or non-road.

els have been processed, the local connection adjustments are summed using the global floating point addition operator and the average adjustment for each connection weight is calculated by dividing the sums by the number of pixels in the image. Each connection weight is then modified by its computed average adjustment value.

As with the clustering algorithm, a method has been developed which allows the system to determine when learning should be stopped. Using this method, the output activation of the perceptron along with the difference between the target and actual output activation is calculated and stored, as before, for every pixel in the input image. After all pixels have been processed, the average pixel error is computed and compared to the average error from the previous perceptron training iteration. If this difference is sufficiently small, learning is halted. If it is not, another iteration of perceptron training is commenced using the new perceptron connection weights. See Figure 7.

After perceptron training is completed, when pixels which are part of the road are shown to the system, they will be classified uniformly as road, regardless of which individual cluster correctly classified them. In our case, the output of the perceptron for a road pixel is near 1.0 while the output for a non-road pixel is close to -1.0.

5.0 The Road Finding Algorithm

In order for the system to successfully navigate the vehicle, it must find the correct path to follow. We will assume that a safe path is down the middle of the road. The technique that our system employs is a parallelized version of a Hough transform. See [Jochem, 1993, 2] for an introduction to Hough transforms. Hough transforms and related techniques are not new to the field of Parallel Computing, but because our work crosses traditional boundaries, (those between Parallel Computing, Artificial Neural Networks, and Robot Navigation) a detailed description of our algorithm, and how it relates to the domain of autonomous roadway navigation, is warranted. We use the topology of the processor array to simulate the parameters of the Hough space and determine the location of the center line of the road by searching for the most likely combination of all possible parameters. The parameters which we are trying to find are the intersection column of the center line and the top row of the image and the angle of the center line with respect to the top row.

The first step of the road finding algorithm is to assign *each processor* in the array a probability of being road. This is done by averaging the perceptron output activation values of all pixels within a particular processor, as defined by the two dimensional hierarchical mapping scheme, and scaling them to a value between 0 and 255. Because the output of the perceptron can be thought of as the ‘certainty’ of a pixel belonging to the road, the average perceptron output for a processor is related to how many road pixels the processor contains. A higher average value indicates more road pixels while a lower value indicates the processor contains many non-road pixels. Processors which lie on road edges are expected to have intermediate values while ones within the edges are expected to have high average values. In this formulation, 0 represents non-road and 255 represents road.

Because the road edges were marked in the combining algorithm and because we use a two dimensional hierarchical mapping, it is easy to compute how wide, in terms of processors, the road is at each row in the processor array. See Figure 8. We will assume that the width of the physical road

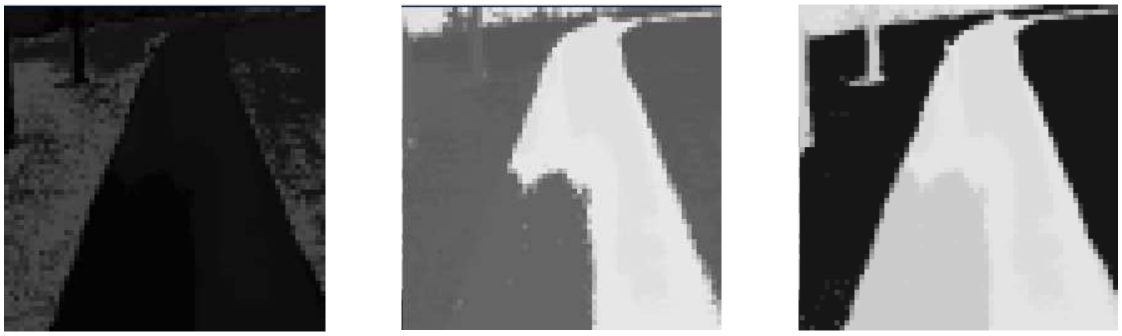


Figure 7. As the perceptron learns, its output better classifies road and non-road. The white pixels represent road, while the black represent off-road. Images shown perceptron output after 1, 3, and 6 training iterations.

is constant and that perspective effects cause the same amount of foreshortening in all images that the system evaluates. More precisely, once we determine that the road is 4 processors wide in the first row, 6 processors wide in the second, 9 in the third, etc., these road width values for each row are held constant for all future images.

The next step is to center a contiguous, horizontal, summation convolution kernel on every processor in the array. (A summation convolution is defined as one in which all values marked by the kernel are added together. The kernel specifies how many processors are to be included in the summation.) The width of this convolution kernel is given by the road width for the row in which the processor is located as determined in the previous paragraph. The convolution kernel is implemented using local, synchronized, neighbor-to-neighbor communications functions and can therefore be computed very quickly. Because the convolution kernel is larger for rows lower in the processor array, not all processors shift their data at the same time. This slight load imbalance has proven to be insignificant. By using this summation convolution kernel, we have essentially computed the likelihood that the road center line passes through each processor. Processors which are near the actual road center line have larger convolution values than ones which are offset from the actual center because the convolution kernel does not extend onto non-road processors. The convolution kernel for processors which are offset from the actual road center will include processors which have low probability of being road because the underlying pixels have been classified as non-road. The convolution process is shown in Figure 8.

Finding the road's center line can be thought of as finding the line through the processor array which passes through processors which contain the largest aggregate center line likelihoods. This is equivalent to finding the maximum accumulator value in a classical Hough transform where the intersection row and intersection angle are the axes of the Hough accumulator. To parallelize this Hough transform, we can shift certain rows of the processor array in a predetermined manner and then sum vertically up each column of the array. The shifting gives us the different intersection angles and finding the maximum vertical summation allows us to determine the intersection of the center line with the top row of the image.

The shifting pattern is designed so that after the first shift, the row center line likelihood in the bottom half of the processor array will be one column to the east from its original position. After the second shift, the bottom third will be two columns to the east, the middle third will be one column to the east and the top third will still have not moved. This pattern can continue until the value of the middle processor in the bottom row has been shifted to the east edge of the processor array. This will happen after 32 shifts in either direction.

Shifting to the west is done concurrently in a similar manner. This shifting represents examining the different center line intersection angles in the Hough accumulator. Because of the way the values are shifted, vertical binary addition up the columns of the processor array can be used to compute the center line intersection and angle parameters. (In this context, binary addition does not refer to binary number addition, but rather to summing using a binary tree of processors.) By finding the maximum vertical summation across all east and west shifts, the precise intersection and angle can be found very efficiently. See Figure 9. Once the intersection column and angle of the center line have been found, the geometric center line of the road can be computed. This geometric description of the center line is passed to the vehicle controller for transformation into the global reference frame and is used to guide the vehicle. This algorithm, while perhaps the hardest to understand, takes full advantage of the tightly coupled processor array.

6.0 Runtime Processing

Once all clusters and the perceptron have been trained, the system is ready for use either in simulation or to physically control the vehicle. Input images are received by the pro-

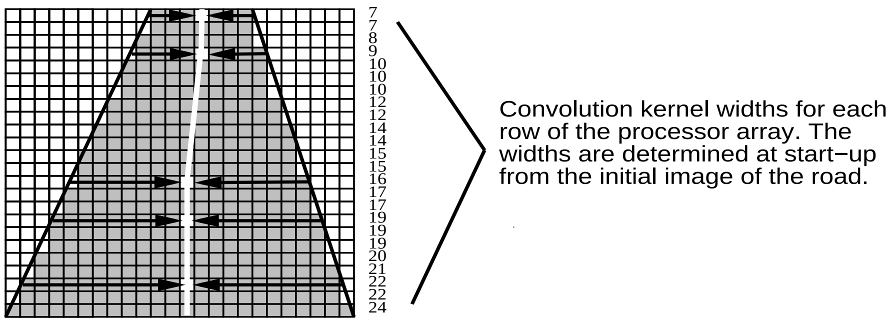

7
7
8
9
10
10
10
12
12
14
14
15
15
16
17
17
19
19
19
20
21
22
22
24
Convolution kernel widths for each row of the processor array. The widths are determined at start−up from the initial image of the road.

Figure 8. Convolution kernels for different rows centered on the actual center line of the road superimposed on the processor array. Arrows indicate the direction of travel of the convolution summation.

cessor array and pixel classification occurs using the precomputed cluster means. The perceptron uses this classification to determine if the pixel is part of the road. The perceptron's output is supplied to the road finding algorithm which determines the center line location. The center line location is transformed onto the ground plane and is passed to the vehicle controller which guides the vehicle. (At this point, because learning is complete, the iterative forms of the clustering and combining algorithms are no longer necessary.)

7.0 Results

The results obtained, both in simulation and driving the vehicle, were very encouraging. The simulation results in terms of color frames processed per second are summarized in Table 1. These results represent average actual processing speed, in terms of frames processed per second, measured when running our system on the specified machine with the set image size. As the number of pixels increased, the frames processed per second decreased in a roughly linear fashion. The slight irregularity is due to the convolution kernel in the road finding algorithm. It is also interesting to note that the 16K MasPar machines did significantly worse than the 4K machines as the image size decreased. Again, this is due to the convolution kernel in the road finding algorithm. We believe that this is not a load imbalance problem but rather a side effect of having a physically larger array topology. Because of the extra width of the processor array, more local neighbor-to-neighbor communication operations are required to compute the summation convolution. As the image size was decreased, these operations came to dominate the processing time.

We are very pleased with the fact that **frame rate computation** was achieved on a 4K MasPar MP-1 machine for the 256x256 pixel image case and on a 16K MasPar MP-2 for the 512x512 pixel image case. Although our ultimate goal is to develop algorithms which can process 512x512 color image streams at frame rate on our 4K MasPar MP-1, we are excited about the results of our initial experiments. Clearly, fine-grained SIMD machines can provide a benefit in real world vision tasks such as autonomous navigation.

On test runs of our testbed vehicle, the Navlab I, the system has been able to successfully navigate the vehicle on a paved road using 128x128 images. At present, the major bottleneck is the slow digitization and communication hardware which is supplying the MasPar with images. This bottleneck limits the image size to 128x128 and the cycle time to approximately 2.5 Hz. Even so, the system has consistently driven the vehicle in a robust manner over the entire 600 meter path. For comparison, other systems that have been developed are ALVINN, which processes 30x32 images at 10 Hz (4 times slower) and SCARF, which processed 60x64 images on a 10 cell Warp supercomputer at 1/3 Hz (32 times slower) [Crisman, 1990][Crisman, 1991]. These figures include image digitization time and slowdowns are computed based on pixels processed per second. A better comparison of processing speed may actually be the time processing the image, not including image acquisition. As an example of this, consider the SCARF system which was also implemented on a parallel machine. For this system running on the Warp machine, a 60x64 images could be processed in one second [Crisman, 1991]. Comparing this to our system (running on a 4K MP-1) processing 64x64 images at 69.4 Hz yields a 74 times speed improvement. For a 1024x1024 image, a 846 times speedup is achievable. On a 16K MP-2, for 128x128 images, processing is 158 times faster and for the 1024x1024 case on this machine, a speedup of over 5000 can be realized. Speedups are computed based on pixels processed per second.

8.0 Other Related Systems

Our system compares very well with any road following system in use today in terms of processing speed. Three competitive high performance systems are the ALVINN and MANIAC systems [Pomerleau, 1991][Jochem, 1993,

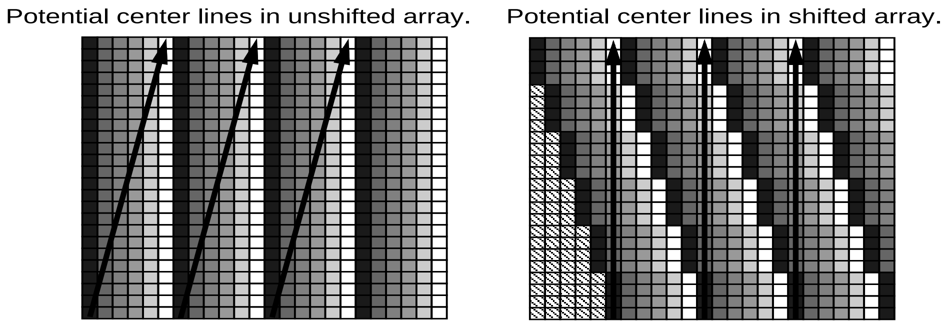

Potential center lines in unshifted array.
Potential center lines in shifted array.

Figure 9. After shifting, center lines which were originally angled, are aligned vertically. Summation up the columns of the processor array can then determine the most likely intersection angle and column.

1], developed at Carnegie Mellon University, and the system designed by the VaMoRs group in Munich, Germany. Like ours, the ALVINN and MANIAC systems are neurally based. However, both use reduced resolution (30x32) preprocessed color images. They cycles at about 10 frames per second, using SPARC based processors, and ALVINN has driven for over 90 miles at speeds of up to 55 m.p.h. (88 km/hr). A system which is founded in very different techniques is the one being developed by the VaMoRs group [Dickmanns, 1992]. This system, which uses strong road and vehicle models to predict where road features will appear in upcoming images, processes monochrome images on custom hardware at speeds near frame rate, but uses a windowing system to track the features so that not every pixel processed. This system has been able to successfully drive on the German Autobahn at speeds around 80km/hr.

Various other systems have been developed that use color data and include the VITS system developed at Martin Marietta [Turk, 1988] and SCARF developed by [Crisman, 1990] at Carnegie Mellon University. The VITS system used a linear combination of the red and blue color bands to segment the road from non-road. The segmented image was thresholded and back-projected onto the ground plane. The vehicle was steered toward the center of this back-projected region. This system was able to drive Martin Marietta's testbed vehicle, the ALV, at speeds up to 20 km/hr on straight, single lane roads found at the test site.

Perhaps most closely related to our system is SCARF. Like our system, SCARF uses color images to develop clusters in color space. These images are greatly reduced; typically 30x32 or 60x64 pixels. The techniques for developing these clusters, however, are very different from the one implemented in our system. SCARF uses Bayesian probability theory and a Gaussian distribution assumption to create and use color clusters to segment incoming color images. The clusters are recreated for every input image by using a strong road model and known road and non-road data as seed points for the new cluster means. The computational load of SCARF is higher than that of our system because of the Bayesian and Gaussian computations and, partially because of this load, the system was only able to drive at speeds of less than 10 m.p.h. The most closely related part of our system and SCARF is the Hough transform based road finding algorithm. Because the Hough algorithm maps very well to finding single lane roads, it was adapted to the parallel framework of our system. Although the implementation of the Hough algorithm is different, the main concept remains the same. Finally, it should be noted that the SCARF's objectives were not limited to autonomous road following; other tasks it performed include intersection detection and per-image updating of the color cluster means. SCARF needed to be smarter because it took a significant amount of time to process a single image.

9.0 Conclusions

Our system has shown that fine-grained SIMD machines can be effectively and efficiently used in real world vision tasks to achieve robust performance. By designing algorithms for specific parallel architectures, the resulting system can achieve higher performance than if the algorithms were simply ported to the parallel machine from a serial implementation. The system described in this paper uses substantially larger frames and processes them at faster rates than other color road following systems. The algorithms presented here were tested on 4K and 16K processor MasPar MP-1 and on 4K, 8K, and 16K processor MasPar MP-2 parallel machines and were used to drive Carnegie Mellon's testbed vehicle, the Navlab I, on paved roads near campus. The goal that our work has strived toward is processing 512x512 pixel color images at frame rate in an autonomous road following system. Our results have shown that in this domain, with the appropriate mapping of algorithm to architecture, frame rate processing is achievable. Most importantly, our results work in the real world and not simply in simulation.

Number of Processors	Size of Color Image				
	1024x1024	512x512	256x256	128x128	64x64
4K MP-1	3.1	11.0	30.6	55.4	69.4
4K MP-2	8.2	25.7	55.4	77.8	86.5
8K MP-2	12.4	25.4	34.3	37.5	na
16K MP-1	9.2	20.0	28.3	31.6	na
16K MP-2	18.5	30.0	35.5	37.2	na

Table 1. System performance in color frames processed per second.

10.0 Acknowledgments

We would like to thank Dr. Edward Delp at Purdue University for allowing us access to their 16K MasPar MP-1. Their machine was purchased in part though NSF award number CDA-9015696. Performance tests on the MP-2's were conducted for us by Philip Ho of MasPar Computer. This work was partially supported by a ARPA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland and by ARPA, under contracts "Perception for Outdoor Navigation" and "Unmanned Ground Vehicle System."

Bibliography

[Ball, 1967] Ball, G. H. and Hall, D. J., "A Clustering Technique for Summarizing Multivariate Data," *Behavioral Science*, Vol. 12, pp. 153-155, March 1967.

[Ballard, 1982] Ballard, D. H. and Brown, C. M. "Computer Vision," Prentice-Hall, 1982. pp. 123-124.

[Crisman, 1990] Crisman, J. D. "Color Vision for the Detection of Unstructured Roads and Intersections," Ph.D. Thesis, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA, May, 1990.

[Crisman, 1991] Crisman, J. D. and Webb, J. "The Warp Machine on Navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 5, pp. 451-465, May 1991.

[Dickmanns, 1992] Dickmanns, E. D. and Mysliwetz, B.D. "Recursive 3-D Road and Relative Ego-State Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, pp. 199-213, May 1992.

[Hertz, 1991] Hertz, J., Krogh, A., and Palmer, R.G. "Introduction to the Theory of Neural Computation," Addison-Wesley, 1991. pp. 217-221.

[Jochem, 1993, 1] Jochem, T.M., Pomerleau, D.A., Thorpe, C.E. "MANIAC: A Next Generation Neurally Based Autonomous Road Follower," In *Proceedings of IAS-3*, Feb. 1993, Pittsburgh, PA, USA.

[Jochem, 1993, 2] Jochem, T.M., and Baluja, S. "Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following," CMU-RI-TR-93-10, Carnegie Mellon University, May, 1993.

[Kluge, 1992] Kluge, K. and Thorpe, C.E. "Representation and Recovery of Road Geometry in YARF." In *Proceedings of the Intelligent Vehicles '92 Symposium*, June, 1992.

[Pomerleau, 1991] Pomerleau, D.A. "Efficient Training of Artificial Neural Networks for Autonomous Navigation," *Neural Computation 3:1*, Terrence Sejnowski (Ed).

[Rumelhart, 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning Internal Representations by Error Propagation" in Rumelhart, D. E. and McClelland, J. L. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press.

[Thorpe, 1990] Thorpe, C.E., editor, "Vision and Navigation: The Carnegie Mellon Navlab," Kluwer Academic Publishers.

[Turk, 1988] Turk, M. A., Morgenthaler, D. G., Gremban, K. D., and Marra, M. "VITS - A Vision System for Autonomous Land Vehicle Navigation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, May 1988.

[Widrow, 1960] Widrow, B. and Hoff, M.E. "Adaptive Cir-- cuit Switching," In *1960 IRE WESCON Convention Record*, part 4, pp. 96-104. New York: IRE.