gNNCaD:

gary’s Neural Network Classifier and Driver

An application for a NN based robotic
driving platform with multiple
articulations based on vision

gary stein

22 Thursday 2004

Advisor:
Dr. Michael Georgiopoulos
# Table of Contents

1. Contributions ........................................................................................................3
2. Introduction ............................................................................................................3
3. Background ...........................................................................................................4
4. Robotic Platform ....................................................................................................5
5. ARTMAP ...............................................................................................................6
6. Vision System .......................................................................................................11
7. Driving System ....................................................................................................14
8. Software Platform ...............................................................................................15
9. Experiments .........................................................................................................17
10. Conclusions .......................................................................................................21
Contributions:

The major contribution of this project is an interesting framework for machine learning research. The robotic platform itself is an interesting test case from an academic standpoint. It has enough sensors and camera systems to relay information about its current situation. The flexibility of the system, both literally and figuratively, allows for movements not typically attainable and allows the machine learning algorithm to develop a unique system of driving. The software developed for the system has been generalized enough to allow abstraction for the learning algorithm from the bare metal. Each of the different IO devices (text, serial, cameras, images) has been abstracted to allow for free use by the program. Since the software design separates the Machine Learning section, other versions can be plugged in to compare.

Introduction:

This project had many goals, which mostly coincided with the contributions. The situations existed that the University of Central Florida (UCF) was offering a class in machine learning and the Association of Unmanned Vehicle Systems International (AUVSI) has an Intelligent Ground Vehicle Competition (IGVC). Between these facts, the idea came about to try and apply current machine learning algorithms to a physical platform for experimentation. A majority of the previously viewed experiments used artificial data to test the abilities of a specific neural network. By using real data, the different neural networks can be tested on the effects of clustering and other situations that might happen.
In addition, it allows for new information to be learned in the process of creating a vehicle using neural networks and computer vision. To personally gain experience about doing experiments and research papers in general.

**Background:**

The inspiration for the project came from several platforms and experiments done in the pass at UCF. The UCF Robotics Laboratory’s first project was Knightrous is 2002. It suffered greatly through time constraints and manpower issues. It was a special topics class of approximately 40 students who were interested in robotics. The platform was a pre-made personal mobility scooter that was interfaced to computer components. The platform, although similar to a car frame, did not allow for the mobility and flexibly to allow for proper control. In 2003, Knightrous was reworked and reformed into the Black Knight, which managed to finish in the top 10 in every category. However, it was time to take on a new framework versus the one previously used.

The IGVC competition consists of two parts. The first part is the autonomous challenge. It consists of a road course of painted lines on grass representing a road to follow. In the course there are several obstacles in the course that must be avoided using sensors including buckets, cones, and potholes. There are several non-obstacles that must be overcome such as ramps and a sand pit. The vehicle that traverses the longest on the course in least amount of time without hitting any of the obstacles wins. The Navigation Challenge consists of GPS coordinates that are spread over an area about a football field in size with buckets and cones located throughout. The main obstacle in this competition is time. The vehicle that reaches the most GPS waypoints in the shortest amount of time wins.
The main point of the competition is to promote the research and the creation of autonomous vehicles from the academic world. To give the students that worked on the projects experience in the field for the future. The various obstacle courses would represent a situation which autonomous vehicles would be needed, both in the military and commercial sectors.

**Robotic Platform:**

The RDB3K platform has 4 “legs”, however, each leg has a direct motor to wheel system attached that provides a driving force in a singular direction of movement. Through the chain system, the drive motor can determine the speed of that leg. On top of the drive wheel, there is also a directional motor that can pivot the tire on its vertical axis to allow the drive wheel to face in any direction, 360 degrees. It allows for each leg to pick a direction to drive toward or away from. The combination of those two motors allows each leg to be omni-directional since it can point and move in any direction. The theory is to allow each leg subsystem to act almost independently of the rest of the vehicle. From each of the leg’s point of view, the camera system, the sonar, and the motors are all belong to that single leg. Because of its independent freedom, the leg can make decision about what direction it wishes to travel. However, since each vehicle shares the central hub a voting system needs to be done to correct for disagreements of the various legs that would cause a tug-o-war. In addition, it allows the entire vehicle to change shape based upon the positioning of the legs to better deal with the given situation.

The sensors on the platform provide interfaces with the real world to determine information about the current status of the platform. The Garmin GPS provides WAAS
enabled positioning to obtain less than 3 meters of accuracy in the worst-case scenario.
The vision system runs off of 4 D-Link USB Webcams. Each is positioned at the top of
each leg at the “knee” joint providing relative camera information versus the position of
the leg. The 12 Ultrasonic range finders are separated into groups of 3 and attached in a
45 degree separation pattern on the “shin” of the leg. The leg positioning is done using
gears and a potentiometer to obtain a split analog voltage that can feed back the angle of
the leg from the “hip” joint. And finally, each leg directional motor is accompanied by a
500 clicks per revolution quadrature encoder to measure the positioning of the wheels
and the current direction of travel. The serial based Gravis Stinger joystick also provides
data from the user for either training purposes or driving the vehicle around in manual
mode.

**ARTMAP:**

The Adaptive Resonance Theory with mapping (ARTMAP) form of neural
network developed originally by Carpenter, Grossberg, and Reynolds in 1991 was a
competitive form of network that eliminates bi-directional weights. The model used by
the ARTMAP has three layers. A presentation layer (F1) that consists of the input for the
system usually modified for use with the neural network. This modification usually
exists as normalizing each input value between 0 and 1 to be properly handled by most
algorithms. The next layer is the hidden layer (F2a) with consists of nodes that store
information about that particular mapping cluster. The final layer is the output layer
(F2b) that contains the mapped output for a particular node. There are bottom up
connections from the input layer to the hidden layer that determine the weight of how
much an input maps to that node. And there are connections from the hidden layer to the
output layer to determine which output that hidden node maps to. The input layer size is determined by the dimensionality of the input data and the output layer size is determined by the number of outputs. The only variable layer is the hidden layer whose size is determined by the data that is presented to it.

The generalized ARTMAP can be separated into two distinct stages, when performing offline training. The first stage is to present a known pair of Input and Output. This will modify the internal nodes in some fashion whether is creates new nodes or modifies the value of existing nodes. Once all nodes are presented, the nodes are then presented again. If all those nodes from the first stage remain the same, then the ARTMAP has learned the situation. If the values have changed then another epoch is needed to be presented until all have stabilized. However, not all ARTMAPs are known to completely stabilize. During the performance stage, the outputs are unknown for the input that is being presented. The top valued node is returned as the answer based on the previous training data. When testing ARTMAPs, the returned value is compared to the actual to determine the degree of performance. Performance for ARTMAPs is factor of how many nodes used to represent data, the amount of data that was missed, and the number of epochs used to learn the data. In practice, the ARTMAP with the fewest nodes that missed the least data is considered the best. The number of epochs is mainly a factor of time and mostly irrelevant if offline training.

The training phase for the generalized ARTMAP has certain steps that need to be done in order to properly implement the method. When the input/output pair is presented to the ARTMAP the Rho value, vigilance, needs to be calculated based on the type of ARTMAP used. Each of the different ARTMAP schemes has a different way of
calculating this value, however, each retains this as a necessary step. Then those values
are compared to the parameter bar-Rho, that is determined by the user, those that are not
greater than bar-Rho are discarded in this pass. Then the bottom up values are calculated
and sorted for the remaining nodes. The bottom up weights also vary from each
ARTMAP however each has a means to obtain this value. The top node is selected based
on the bottom up value which represents the hidden layer node that most represents the
input presented. If the output is the same, then the proper information is already known
and the input is combined with the old node to create a new node that may or may not
have the same values. If the outputs do not match, then bar-Rho is increased to the Rho
of the hidden node that just failed. Those nodes that are not above this new bar-Rho are
discarded and the new Best node is selected. The process is repeated until the output
matches or the sorted list reaches the end. When the list reaches the end, a new node is
created from the designated input and output pair. The performance phase is very similar
to the training phase except that the output is no longer known. Simply the bottom up
values are calculated and then used in the sorting process and the Rho can ignored.
Effectively, the performance stage sets the initial bar-Rho to zero and selects from the top
of the sorted list. From that node, the output that it maps to is chosen and returned as the
output for that particular input. Because the training data was fully presented previously,
the hidden layer nodes should already contain all the necessary information needed and
contain all of the proper mappings for input data. Therefore, that output of the
performance stage should be correct for any reasonable input.

The specific calculations for each of the different forms of ARTMAP are typically
very different from each other, however, they share certain properties. Most systems use
some form of geometry to represent the grouping around sets of inputs. The value returned for Rho should always be within the range between 0 and 1. This allows for the bar-Rho value to represent the vigilance on a scale from 0 to 100% to make logical sense in context. The bottom up values that are calculated are not required to be between 0 and 1, but however are chosen to be small positive real numbers for use for sorting purposes.

The Fuzzy ARTMAP (FAM) uses magnitude of vectors and the Fuzzy Min operator to do its calculations. The magnitude of the vector is simply the sum of the vectors parts and the fuzzy min is the smallest of each vector location between two vectors.

\[
\rho = \frac{|Ir \land W_j|}{|Ir|} \quad B = \frac{|Ir \land W_j|}{(\alpha + |W_j|)}
\]

Where \(Ir\) is equal to the input vector consisting of the input as the first half and the second half of 1 minus the normalized input value and \(W_j\) is equal to the vector that represents the hidden node. The alpha parameter can be changed and varies the bottom up value in relation to the magnitude of the hidden node. Modifications done to FAM that are not part of the original algorithm were the deletion of the uncommitted node and initial trimming of nodes whose Rho that were not above the bar-Rho value. Both of these changes were made to make FAM more similar to the later developed ARTMAPs that contain these changes. Logically this change does not make any difference to the hidden nodes or the ARTMAP in general. Since the uncommitted is always last picked, it can be removed and identified that if all nodes have been evaluated then the uncommitted would have been chosen but just catch that exception rather than have a physical node. Additionally, the trimming of nodes based on the bar-Rho value does not
affect the system since they would have been ignored in the check to see if the Rho value was greater than the bar-Rho later on.

GAM uses standard deviations and natural logarithms to do calculations. In addition, GAM needs to keep track of the number of times that node was selected.

\[
\rho = \exp\left(-\frac{1}{2} \sum \left( \frac{x - \mu_i}{\sigma_i} \right)^2 \right)
\]

\[
B = \exp\left(-\frac{1}{2} \sum \left( \frac{x - \mu_i}{\sigma_i} \right)^2 \right) N * -\ln \left( \prod \sigma_i \right)
\]

Where \( x \) is equal to the input, \( \mu \) is the mean contained by the hidden node, \( \sigma \) is the standard deviation contained by the hidden node, and \( N \) is the number of times that node was selected to add information. The parameter that can be changed in GAM is the gamma value that represents the initial standard deviation for each dimension when new nodes are added. Changes versus the original GAM include the change to the negative natural log of the product of the sigma’s versus the original divide by the product of the sigma’s. This lead to smaller and more maintainable bottom up values than the original since the sigma’s were very small numbers. In addition, the natural log of the product can be broken into the sum of the natural log of each number. None of these changes affect the final outcome because the natural log is a bijective function were there is a pure one to one mapping of input to output and for every input less than 1 the natural log of the smaller of two numbers will be strictly smaller than the natural log of the larger.
Vision System:

The vision system consists of several parts that are abstracted with respect to the calling functions. But the main two phases of the vision system are the camera capture and the image pre-processing stage. The camera capture is done through the Video 4 Linux (v4l) interface in Linux. V4l is a system of headers to standardize the API for capturing from video devices. It is able to abstract the physical source of the data (USB, firewire, capture card) to allow one standard way to capture data and interpret into a computer readable format. More specifically, one layer below the API call interaction is the ov518 kernel driver that converts the compressed USB camera data into an Y’CbCr Image stream. That is then copied into an array, and then presented to the preprocessor.

The Y’CbCr color space is a non-visible delta space that engineers have used to save on bandwidth and stay compatible with black and white. The format was first devised to maintain compatibility to the old black and white televisions. This is done by transmitting the Luma value (Y’) as the first part of the image, which can be directly converted to greyscale. Following that section is the CbCr values, which are weighted delta values with respect to blue and red. Since the human eye is more perceptive to changes in brightness versus changes in color, the resolution of the delta values are reduced by one half. This therefore reduces the entire bandwidth by half. Since computers do not deal in the Y’CbCr space but typically in the 24-bit RGB space, the red, green, and blue spectrum need to be extracted from the original data. However, the transformation is done by a linear matrix multiply and is not a one to one operation.

\[
\begin{align*}
R &= Y + 1.4022 \times (Cb - 128) \\
G &= Y - 0.7145 \times (Cb - 128) - 0.3456 \times (Cr - 128) \\
B &= Y + 1.7771 \times (Cr - 128)
\end{align*}
\]
The values of $Y$ can range from 0 to 255 to approximate the grey value. $Cb$ and $Cr$ can also range from 0 to 255, where looking at the math, it can be seen that those values less than 128 equate to negative delta values and above are positive values. Experimentally truncating the output to integers and using all values of $Y'$, $Cb$, and $Cr$: only 25.33% of the representable colors in RGB can be recovered. The RGB space is required for purposes of display and manipulation of the data on a computer properly.

The image then must be broken up into subsections based on a size parameter that must be an integer divisor of both the width and height of the image to produce small blocks. The reason for the division is to take advantage of the similarity of localized data to help with classification. When the size parameter is very low then number of inputs presented to the NN increases, but the amount of different data the blocks can overlap is minimal. However when the size parameter is high, the number of inputs is smaller and the amount of overlapping data is greater. A happy median needs to be made between those two different advantages, less overlap and less nodes. Decreasing the block size a factor of 2 increases the number of inputs a factor of 4. Additionally, the block size and overlap really cannot be larger than the smallest object being classified since that object’s data could be lost in the surrounding data. Typically, the input captured by the cameras is 320x240 pixels, experimentally the block size of 8x8 seemed to work well with respect to both overlap and number of inputs. For a typical image at block size 8, the number of inputs is 1200.

The sub-blocks then need to be normalized versus the total image so they can be presented equally with blocks from other images. The theory behind the way the blocks are normalized consists of the idea that any information should be accurately estimated
by using its mean and standard deviation. This process reduces the block into a simple set of numbers than can then be presented to the ARTMAP. The mean and standard deviation of the entire image are calculated in order to properly normalize the sub blocks. Each of the sub blocks then has its own mean and standard deviation taken.

$$\mu = \frac{\sum x}{n} \quad \sigma = \sqrt{\frac{(x - \mu)^2}{n - 1}}$$

Then the entire images input pattern is then pressed on each block to normalize its value between 0 and 1 for all values. The mean for each color channel is divided by 255 to evenly distribute the average color. The standard deviation of the block for each color is divided by four times the standard deviation of the entire image. This produces a 6 dimensional string of data for the standard deviation and average of each color channel. This data is now ready to be presented to the ARTMAP as an input.

For the ARTMAP, the input comes in as an array between 0 and 1 and a bit-mask created by the oracle of output labels that match the input blocks. The aforementioned normalization process calculates the inputs from stored images during the training phase and from the camera during the performance phase. For the training phase a good set of test images are selected that should represent all of the objects that will be classified. The images will be processed to provide the input of the input/output pair. The output section is created by the human oracle. The Oracle manually reviews the test images and classifies each sub-block to its pertaining output. Those are stored in a bit-mask file representing the outputs, which are then read by the program as the output section of the input/output pair. Those two pieces of data, input and output, then are presented to the ARTMAP for the training phase until stabilized. Then those values of the hidden nodes
and their connections to the output layer are saved to disk for later processing. During the performance phase, the hidden node values are read back in and only the inputs can be calculated from the incoming camera images. The output of the performance phase should return the same output labels as presented originally in the training section.

Driving System:

ARTMAP is also used to make decisions about where to drive based on the inputs. The driving system receives a series of inputs from different sensors that are generally abstracted and combined together. The output of the previous vision system needs to be placed in the local map. First the classified camera data needs to be transformed from the image projection to the flat surface. Then that image is rotated into the local map with relation to the leg angles at the hip. The ultrasonic range finders are then added as additional projected obstacles into the local map. The GPS simply is used as a measure of how much the vehicle wishes to go in one direction or another for the navigation challenge only, and contributes two more dimensions. In the training section, the joystick provides the output section of the input/output pair for training.

The projection of the image on the ground plane is based upon the camera iris for what the cone of vision the camera has. The height affects the distance and size of the projection on the ground but not the shape. The angle versus normal affects both the shape and size of the projected image.

\[
I = \frac{(H \tan (\alpha) + Y \tan (\frac{\beta}{2}))}{(1 - Y \tan (\frac{\beta}{2}) \tan (\alpha))}
\]
\[
J = \frac{(X \cdot H \cdot \tan \left( \frac{\beta}{2} \right) \sqrt{1 + (Y \cdot \tan \left( \frac{\beta}{2} \right))^2})}{(\cos(\alpha) - Y \cdot \tan \left( \frac{\beta}{2} \right) \cdot \sin(\alpha)) \sqrt{1 - (Y \cdot \tan \left( \frac{\beta}{2} \right))^2}}
\]

Where beta is equal to the Iris angle, alpha is equal to the angle versus normal, X and Y are the coordinates of the non-projected image with the center at the origin, and H is equal to the height of the camera off of the ground. Once the image is projected, the image is then translated to a fixed point equal to the normal leg placement. This process is done by simply adding the nominal leg position to the projected position. And finally, the image is rotated based on the leg angle into position by a pivot around that point equal to the current leg angle. The current leg angle is already tracked by the potentiometer on the leg and taking the offset versus the initial position gives the rotation angle.

**Software Platform:**

The software platform has been generalized to operate on run time without prior knowledge of the functional system blocks that are going to be running in the program. The abstraction from the classifier from the IO allows different file types to be used or different ARTMAPS to be used based on settings interpreted automatically through virtual functions. Although for different functional blocks to work, an interface must be written to convert it to a standard type. The general pipeline defined by the program is IO-In, Pre-Processor, Classifier, Post-Processor, and IO-Out. Following that chain of events produces an abstract system that can be built on using inheritance. The IO-In functions will relate to the serial ports, images, text files, cameras, and joysticks. The
Pre-Processor translates those IO data blocks into useable information by converting color space and breaking up images or convert the GPS into position information. The Pre-Processor produces the Input blocks for the Classifier. The Classifier becomes the various ARTMAPS either FAM or GAM that process the data and produces Output blocks for the Post-Processor. The Post-Processor does the image rotation and converts the Output into motor commands for the IO-Out. The IO-Out does the physical communication to the output devices like the screen or the motor micro-control units.

By breaking the process into smaller blocks, the program can be done in a pipeline and reduces the overhead involved with waiting for input. In addition, it lends itself well to parallelism because different cameras will be capturing at different times, and instead of processing one at a time, multiple images can be done while at different stages of the pipeline. In addition, the program is constructed abstract in C++. Meaning that base functions are made and the additional functions bolted on will either use the functions of the parent or if overloaded will use their own functions using virtual members. In this way a base ARTMAP class can be constructed that contains all of the functions and base steps for training and performance contained within it as the parent. The children FAM and GAM then extend off that class simply adding their distinct vigilance and bottom up calculations along with their particular parameters. Without changing anything else in the system, the ARTMAP can be changed out to test different architectures against each other. Using the idea of polymorphism, a pointer also can be made to the base class and during runtime, the assignment which ARTMAP to use can be done, and the system will call the correct internal functions accordingly.
By using this system of plug and play, once the base classes are built, it should be easy to add additional ARTMAPs, sensors, or even different frames. The architecture is not specific to the one robotic frame but could be modified using the same abstract classes to be used on other frames.

**Experiments:**

The experiments were to Compare FAM versus GAM for different input types. Two different data sets: little data that has been properly classified without overlap sets and an overlap set that has many overlaps and many more input/output pairs of classified data. The little data was to test how well the ARTMAP could expand to find other cases including overlap only knowing a small amount of “perfect” information about some classified image. Presented to the system were 204 input/output pairs for the entire test image of 1200 inputs total. The overlapped test case was used to test how well the system would work with overlapping data and imperfect input data from the oracle. Included in the case were input blocks that contained at least two outputs but were specifically given only one output. The overlapped test set consisted of 523 input/output pairs including bad information from the oracle.

The two ARTMAPs FAM and GAM both had parameters to choose from to try and optimize how well they worked. They shared the parameter bar-Rho, the baseline vigilance, that could vary between 0 and 1 to determine how close each of the hidden nodes truly had to be to the input in order to even be considered. FAM had the parameter alpha that could be used in order to adjust how the size of the hidden node magnitude would change the system. GAM had the parameter gamma that would adjust the initial standard deviation used by the system and in turn affect the initial size of the point node
in collecting other nodes around it in its cluster. For the experiment, the parameters were varied for each system over bar-Rho and its own parameter. Bar-Rho was varies between 0.00 and 0.99 and the other parameter was varied between 0.00 and 0.20. Since each of the ARTMAPs is a deterministic process, only one run of each parameter set was run on both data sets. Those sets were then averaged over the different bar-Rho values to produce a set of the average over all personal parameter settings for that value bar-Rho. Those values were then graphed for the different optimizing parameters: number of epochs, percent missed, and number of nodes. In addition, the values for percent missed versus the number of nodes were graphed to determine a relationship.
By comparison of the graphs for the number of epochs, it can be seen that the more overlapped data caused both architectures to perform more epochs and therefore more time to process the input information, but in both cases they eventually were able to converge. FAM was rather steady with the number of passes required to process both sets. The little set only required the minimum of 2 passes while the overlap required 6. GAM on the other hand seemed to increase to a much higher point for both cases and was almost strictly higher than FAM for every Rho value. This could relate to the fact that GAM will not necessarily converge.

Both algorithms did well in the percent missed category for classifying the image. As expected, both algorithms did much better with classifying blocks with the little set since it was much more perfect of information to categorize on. FAM’s shortcomings with overlapped data can be seen with respect to the overlapped case. It performed relatively poorly missing approximately 22% of the data for low values of Rho, but increasing Rho decreased the number of misclassified points. However, GAM performed on par or better with the overlapped set as FAM did with the little set. FAM was unable to perform better than 2.5% until high values of Rho. GAM performed perfectly for all Rho values in the little case and continuously became better, however during this phase it also became much slower. All around GAM had better performance with respect to percent missed, however, took longer to process.

The reason for GAM’s increase in time can be seen in the following graph with respect to the number of nodes. FAM managed to stay primarily flat, only raising for high values of Rho and was almost strictly less than GAM in nodes for each Rho value. The little test case produced less nodes, as expected, because less new nodes had to be
made because of match tracking and overlap. The increased number of nodes increased the operating time and memory usage.

To get a comparison of how the number of nodes relates to the percent missed, a scatter plot was formed. By the shape of the scatter plot for both FAM and GAM appear to develop an inverse relationship between the two values. The more nodes produces better results by means of less missed. This can be expected because the more points produced on the same data would break down the clustering data into more precise regions. However, it can be seen that because of this relationship, the better the classification is going to need more nodes and more memory.

![Graph](image)

The generalized averaging 2 dimensional graph is good for determining a good personal parameter value for use with the selected ARTMAP. However, if one wants to pick and choose the best of both parameters for the selected data, a 3 dimensional mesh must be constructed and the lowest point will produce the best results. However, to be factored in is also the number of nodes that determines the amount of memory used and the amount of time for training. FAM appears to work best with alpha values between .05 and .15, it appears to make the smallest percent missed and a large Rho value. However, it can also be seen that for the same value, if the Rho is too small, it will can
cause odd things to happen. Parameter selection is vital because it seems with an improper Rho value would throw off the program drastically. With GAM, the worst case scenario is still pretty good with only 14% missed. GAM appears to work different than FAM in that the smaller the gamma and the higher the Rho the better the results appear to be. But to optimize for memory a lesser value of Rho should probably be chosen versus selecting the lowest and highest value for gamma and Rho respectively.

To summarize the results of all the previous graphs, FAM and GAM both functioned well to classify images. FAM worked quicker with fewer nodes and less training time plus took fewer epochs to stabilize but did not work as well with overlapping sets and parameter selection vastly changed the output of the system. GAM classified much better with less missed blocks plus could have worse input data sets and still properly classify but it took more time, more epochs and more nodes to classify.

**Conclusions:**

The ARTMAP selection seems to be the difficult option because it is based on need. How accurate the results need to be versus time constraints and the state of the input data. GAM produces better results with dirty data but requires more time and memory. If real-time results are required then the FAM should be chosen, yet it does not produce as good results. The abstract programming model allows for plugging in of different systems. The ARTMAP method is similar and the same methods can be used through virtual functions. In addition, the same model allows for pipelining and parallelism by being able to break the problem into much smaller sections. The final conclusion relates to the project as a whole, it was a large project and time constraints did not allow for experimentation with the driving system. It will have to be future research.
Works Cited

Carpenter, Gail A, et all. Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps

Georgiopoulos, Michael. Discussions on ARTMAP

Linux OVCam Drivers: Reference and FAQ


Video4Linux Kernel API Reference v0.1:19990430

Xilinx Color Space Conversion: CCIR 601 Standard