Teaching the computer subjective notions of feature connectedness in a visual scene for real time vision.

T. Nathan Mundhenk[a,c,1], Chris Landauer[c], Kirstie Bellman[c], Michael A. Arbib[a,b], Laurent Itti[a,b]

[a]Computer Science Department, Henry Salvatori Computer Science Center, University of Southern California, Los Angeles, CA 90089-0781;
[b]Neuroscience Program, Hedco Neuroscience Building, University of Southern California, Los Angeles, CA 90089-2520;
[c]Aerospace Integration Science Center, Aerospace Corporation, 2350 E. El Segundo Blvd, El Segundo, CA 90245-4691

ABSTRACT

We discus a tool kit for usage in scene understanding where prior information about targets is not necessarily understood. As such, we give it a notion of connectivity such that it can classify features in an image for the purpose of tracking and identification. The tool VFAT (Visual Feature Analysis Tool) is designed to work in real time in an intelligent multi agent room. It is built around a modular design and includes several fast vision processes. The first components discussed are for feature selection using visual saliency and Monte Carlo selection. Then features that have been selected from an image are mixed into useful and more complex features. All the features are then reduced in dimension and contrasted using a combination of Independent Component Analysis and Principle Component Analysis (ICA/PCA). Once this has been done, we classify features using a custom non-parametric classifier (NPclassify) that does not require hard parameters such as class size or number of classes so that VFAT can create classes without stringent priors about class structure. These classes are then generalized using Gaussian regions which allows easier storage of class properties and computation of probability for class matching. To speed up to creation of Gaussian regions we use a system of rotations instead of the traditional Psuedo-inverse method. In addtion to discussing the structure of VFAT we discuss training of the current system which is relatively easy to perform. ICA/PCA is trained by giving VFAT a large number of random images. The ICA/PCA matrix is computed by features extracted by VFAT. The non-parametric classifier NPclasify it trained by presenting it with images of objects having it decide how many objects it thinks it sees. The difference between what it sees and what it is supposed to see in terms of the number of objects is used as the error term and allows VFAT to learn to classify based upon the experimenters subjective idea of good classification.

Keywords: iRoom, Biological, Vision, Tool, Multi Agent, Saliency, Real Time

1. INTRODUCTION

1.1 The iRoom Project

As a primer for this paper we first wish to give an overview of the project for which this work is intended to be a part of in order to provide a basis to understand the design decisions we have chosen. The iRoom project is a distributed surveillance system designed to both provide a practical framework upon which to deploy a working system in the real world, but it is also a platform on which to test the feasibility of biologically inspired algorithms for both vision and cognition. As such, where it is feasible systems are designed around brain operating principles. However since the brain is still far to complex to emulate in real time, more traditional computational approaches are used in places particularly where the processes fall further from the theoretical scope we are most interested in. As a result, the work presented here exhibits both strongly biological elements and weakly biological elements.

[1] mundhenk@usc.edu; University of Southern California, 3641 Watt Way, HNB 10, Los Angeles Ca, 90089-2520

The iRoom is being designed to track humans around an area such as a room or group of rooms and gain basic understanding of their actions. The current system is being implemented to do this using a distributed Beowulf architecture whereby computational resources are shared by individual PC's each of which may control its own camera. Additionally, it is being designed as a method to extract interesting scenes and bring them to the attention of human operators or to the attention of an executive machine, which can integrate scene understanding. As such, the iRoom needs a fast way in which to gather visual information about a scene and gleam understanding from it in order to perform real time scene understanding. Such information may include whether the same person has been seated at a computer all day. This, may be less interesting, depending on how suspicious people act, than if someone new sits down at the same computer. As such, one of the applications of the iRoom project is to find scenes which may be most interesting to monitor and bring such scenes to the attention of a human operator who might otherwise be overloaded watching far to many cameras survey an area.

Another additional feature which we wish to include in the iRoom is an ease of use. Autonomous and robotic systems are in much the same state as PC in the early 1980's. Applications are beginning to appear for home and small business use, but these systems, for the most part, are still to complex for untrained people to deploy and use. Thus, the iRoom should have an intuitive feel in how it works. One ideal scenario might be that the room could be trained in the same way one trains a pet. That is, the iRoom should be able to learn based upon individual subjective criteria of interest. If trying to interpret a scene, the iRoom should have a simple and intuitive system which allows it to learn the trainers subjective interpretations of the scene.

1.2 Modular Systems Design

The iRoom is being comprised of independent standardized computational modules. Each module is treated as an expert in a mixed expert system. That is, each module may have understanding of different aspects of scene analysis, but that their opinion about different aspects of the scene are integrated into a group opinion. For instance, lower level modules may isolate parts of a scene as more interesting based upon the saliency of feature within an image. These areas can then be analyzed by more complex modules which, as a result, can run more efficiently since they will only examine sections of the image which are of highest saliency. Additionally, these higher level modules may communicate to the lower level saliency module in order to bias its working. An analogy might be that when looking for your keys, you might prime your visual system to be more attentive for the color of your keys so they are easier for you to spot.

Another motivation for a modular system is that as new programs are developed that are more effective for the task, they can more easily be integrated into the existing system. As such each module is a resource which can provide different things for other modules. In general this means that as each part of the system is deployed it must be standardized such that what it will provide and what it needs are well understood.

1.3. The Visual Feature Analysis Tool (*VFAT*)

The rest of this paper will mostly be devoted to describing the VFAT module, which has been designed as a middle layer between lower level visual saliency and higher level visual scene analysis. Its primary duties include coalescing features statistically and creation of complex features from simpler features. For instance one thing it would try and do is group all pixels in an image together that are the same color and texture and label them as part of the same object. The statistical values from that class could be computed which would for instance allow a Bayesian comparison to determine if two objects are the same thing. As an example, two objects that are the same color are more likely to be the same object than two objects of different color. Additionally, by linking objects by feature, one can perform tracking on that objects movement.

Once features are grouped, VFAT will give these groups to higher level modules which can use them to compute compound objects or to determine the gist of a scene. Additionally, it can bias lower level saliency to attend to certain objects that might be more interesting to higher level areas. This is possible since it can weight the saliency mechanisms directly with the statistical properties of the group features.

2. VFAT design

2.1 Overview

VFAT is comprised of several smaller modules (figure 1). These modules fall into three different categories primarily based upon the stage of visual processing the module participates in. The first group of modules are the feature selection modules. These are primarily focused on selecting which feature location and type of feature in the image should be selected for further processing. An example of a feature might be the red value on a pixel or the crispness of an edge in some location in an image. Several features measured from one location in an image, which may be a region of several pixels comprise a feature vector for that location. Several feature vectors are a feature set. It should also be noted that a feature channel is a collection of the same type of feature but with different sizes for the region sampled (scales).

The second group of modules are feature mixing modules. These take the features selected by the first group and mix them into more complex features. They also perform dimensionality reduction on the feature vectors at each location selected. That is, for each feature vector, we would like to reduce its size primarily to reduce computational overhead.

The third type of modules are classifiers. These take the features and group them into classes and find the statistical properties of each feature class in an image. Thus, once we have a working feature set, we want to find how they statistically relate to each other for the purpose of tracking and identification.

2.2 Feature Selection Modules

2.2.1 Visual Saliency

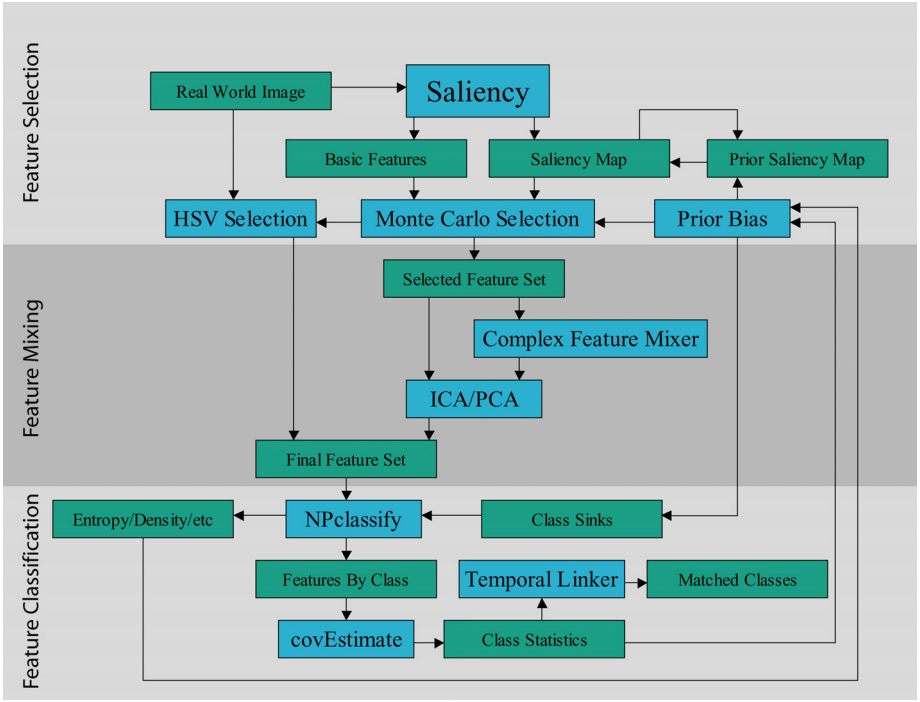

Figure 1: VFAT is a conglomeration of different modules that select features from an image, mix them into more complex features and then tries to classify those features without priors for what kind of features it should be looking for.

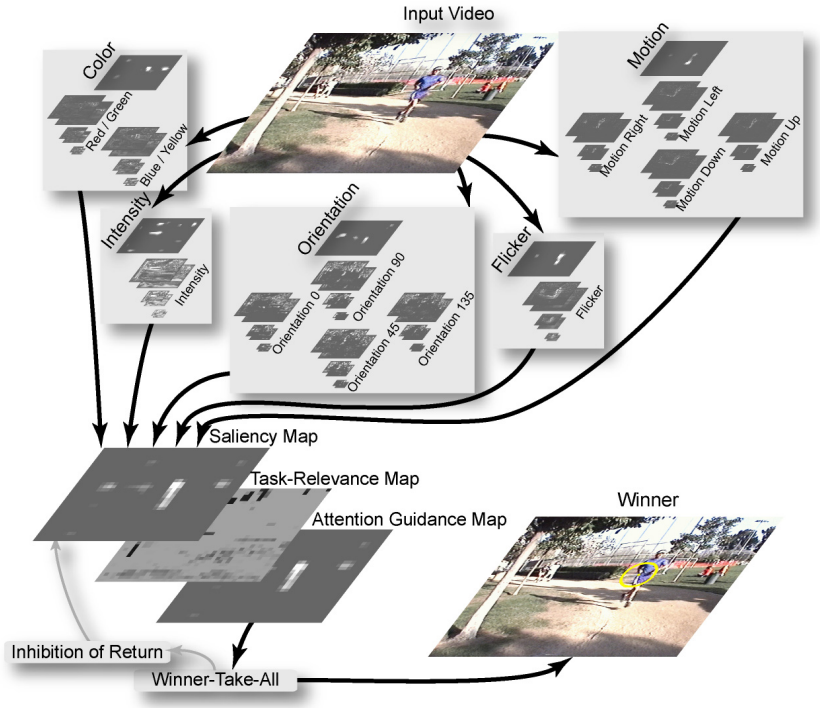

Figure 2: Saliency is comprised of several channels which process an image at a variety of different scales and then combine those results into a saliency map.

The first stage of processing is finding which locations in an image are most salient. This is done using the saliency program created by [2], which works by looking for certain types of uniqueness in an image (figure 2). This simulates the processing in visual cortex that the human brain performs in looking for locations in an image, which are most salient. For instance, a red coke can placed among green foliage would be highly salient since it contrasts green against red. In essence, each pixel in an image can be analyzed and assigned a saliency value. From this a saliency map can be created. The saliency map simply tells us the saliency of each pixel in an image.

2.2.2 Monte Carlo Selection

The saliency map is taken and treated as a statistical map for the purpose of Monte Carlo selection. The currently used method will extract a specified number of features from an image. Highly salient locations in an image have a much higher probability of being selected than regions of low saliency. Additionally, biases from other modules may cause certain locations to be picked over consecutive frames from a video. For instance, if properties of a feature vector indicate it is very useful, then it makes sense to select from that location in the next frame. Thus, the saliency map combines with posterior analysis to select locations in an image which are of greatest interest.

2.3 Mixing Modules

2.3.1 Junction and End Stop Extraction

During the computation of visual saliency, orientation filtered maps are created. These are the responses of the image to Gabor wavelet filters. These indicate edges in the image. Since each filter is tuned to a single preferred orientation, a response from a filter indicates an edge that is pointed in the direction of preference. The response from the filters are stored in individual feature maps. One can think of a feature map as simply an image which is brightest where the filter produced its highest response. Since the feature maps are computed as part of the saliency code, re-using them can be advantageous from an efficiency standpoint. As such we create feature maps to find

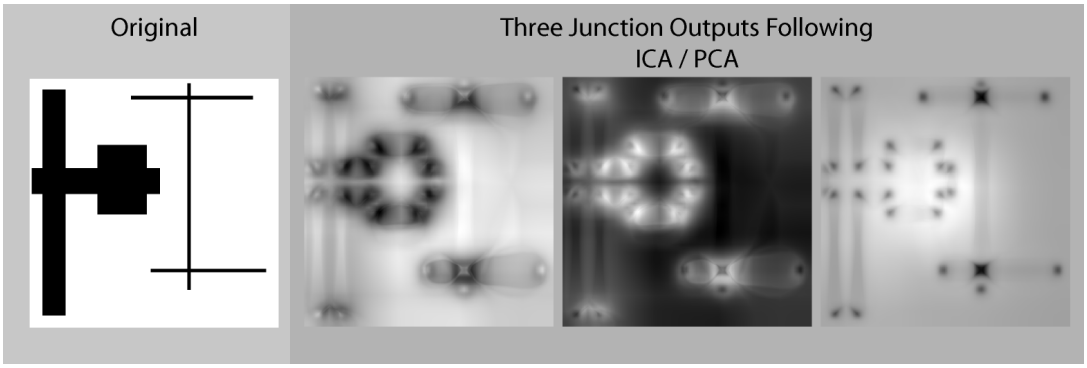

Figure 3: The three images on the right are the results of the complex junction channel after ICA/PCA processing from the original image on the left. As can be seen it does a reasonable job of finding both junctions and end stops.

visual junctions and end-stops in an image by mixing the orientation maps (figure 3). We believe such new complex feature maps tell us about the texture at image locations.

The junction and end stop maps are computed as follows. At some common point on the orientation maps (*x,y*) the filter responses from the orientation filters are combined. Here the response to an orientation in one orientation map is subtracted from an orthogonal map's orientation filter output and divided by a normalizer *n* which is the max value for the numerator. For instance, one orientation map that is selective for 0 degree angles is subtracted from another map selective for 90 degree angles. This yields the *lineyness* of a location in an image since where orthogonal maps overlap is at the junctions of lines.

(1.1) $$c_{xy}^{i} = \frac{\left| p_{xy}^{orth} - p_{xy} \right|}{n}$$

We then compute a term which is the orthogonal filter responses summed. This is nothing more than the sum of the responses in two orthogonal orientation maps.

(1.2) $$c_{xy}^{\gamma i} = \frac{p_{xy}^{orth} + p_{xy}}{n_{\gamma}}$$

We then compute a term for the full output of all the orientation filters normalized. Here $\gamma 1$ would be the result of the 0 and 90 degree filters and $\gamma 2$ would be the result of the 45 and 135 degree filters.

(1.3) $$\alpha_{xy} = \frac{c_{xy}^{\gamma 1} + c_{xy}^{\gamma 2}}{n}$$

The total lineyness a is then computed by combining the results from eq. 1.1

(1.4) $$\beta_{xy} = \frac{\left| c_{xy}^{1} - c_{xy}^{2} \right|}{n}$$

Junctions are then computed by subtracting the lineyness term from the total output of the orientation filters.

(1.5) $$\gamma_{xy} = \alpha_{xy} - \beta_{xy}$$

Since the junction map is computed by adding and subtracting orientation maps which have already been computed during the saliency computation phase, we gain efficiency we wouldn't have had if we were forced to convolve a whole new map by a kernel filter.

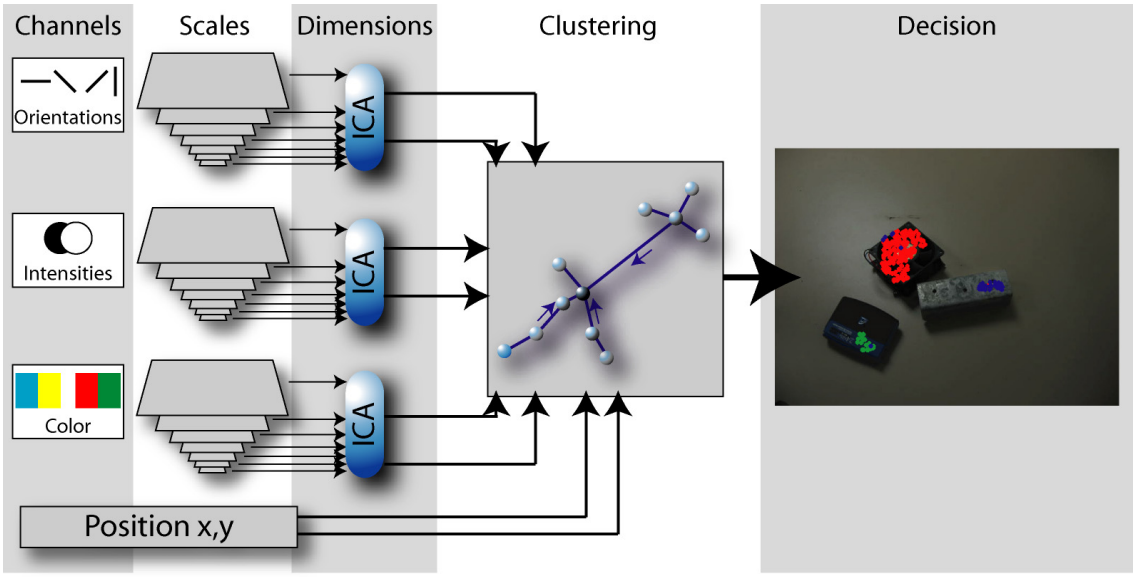

Figure 4: Saliency features are extracted from the image and reduced in dimension using ICA/PCA. These are then classified using NPclassify.

2.3.2 ICA/PCA

We decrease the dimensionality of each feature vector by using a combination of Independent Component Analysis (ICA) and Principle Component Analysis (PCA). This is done using FastICA [1] to create ICA un-mixing matrices offline. The procedure for training this is to extract a large number of features from a large number of random images. We generally use one to two hundred images and 300 points from each image using the Monte Carlo selection processes just described. FastICA first determines the PCA reduction matrix then determines the matrix that maximizes the mutual information using ICA. Unmixing matrices are computed for each type of feature across scales. So as an example, the red-green opponency channel is computed at different scales, usually six, PCA/ICA will produce a reduced set of two opponency maps from the six original scale maps. Using ICA with PCA helps to ensure that we not only reduce the dimension of our data set, but that the information sets are fairly unique. From the current data, we reduce the total number of dimensions with all channels from 72 to 14 which is a substantial efficiency gain especially given the fact that some modules have complexity $O(d^2)$ for d number of dimensions.

2.4 Classification Modules

2.4.1 Classification of Features with NPclassify

Features are initially classified using a custom non-parametric clustering algorithm called NPclassify. The idea behind the design of NPclassify is to create a clustering mechanism which has *soft* parameters that are learned and are used to classify features. We define here soft parameters as values which specify the values of other parameters which are dependant upon the values of the dataset. For instance, if we wanted to determine at which point to cut off a dataset and decided on two standard deviations from the mean, two standard deviations would be a soft parameter since the actual cut off distance depends on the dataset.

NPclassify (figure 4 and 5) [3] works by using a kernel to find the density at every sample point. The currently used kernel does this by computing the inverse of the sum of the Euclidian distance from each point to all other points. After density has been computed the sample points are linked together. This is done by linking each point to the closest point which has a higher density. This creates a path of edges which ascends acyclically along the points to the point in the data set which has the highest density of all. Classes are created by figuring out which links need to be cut. For instance, if a link between two sample points is much longer than most links, it suggests a leap from one

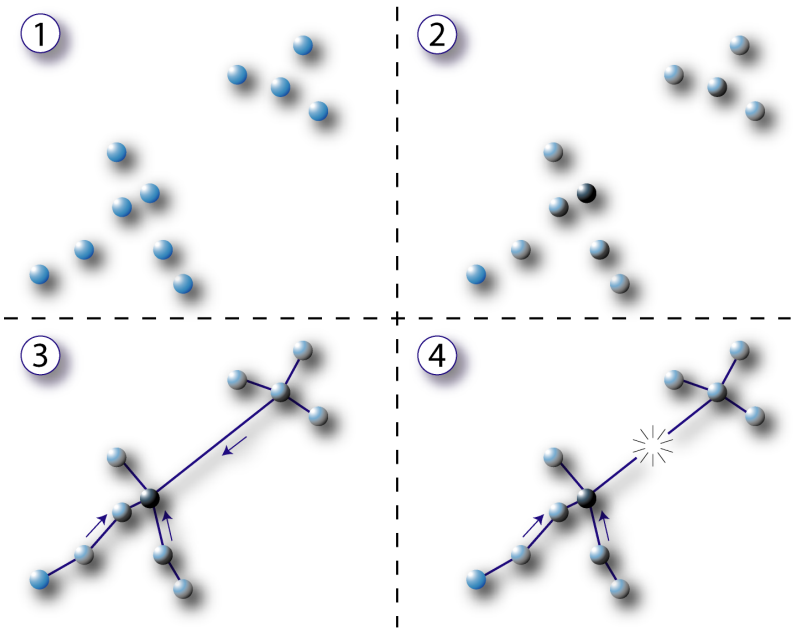

Figure 5: NPclassify works by (1) first taking in a set of points (feature vectors) (2) then each point is assigned a density which is the inverse of the distance to all other points (3) Points are then linked by connecting a point to the nearest point which has a higher density (4) Very long links (edges) are cut if they are for instance statistically longer than most other links. This creates separate classes.

statistical mode to another. This then may be a good place to cut and create two separate classes. Additionally, classes should be separated based upon the number of members the new class will have. After classes have been created, they can be further separated by using interclass statistics.

The advantage to using NPclassify is that we are not required to have a prior number of classes or any prior information about the spatial or sample sizes of each class. Instead, the modal distribution of the dataset combined with learned notions of feature connectedness determine whether a class should be created. So long as there is some general statistical homogeneity between training and testing datasets we should expect good performance for clustering based classification. The training results are discussed later in the section on training results.

2.4.2 Gaussian Generalization and Approximation

In order to store classes for future processing it is important to generalize them. Gaussian ellipsoids are used since their memory usage for any class is $O(d^2)$ for d number of dimensions for a given class. Since d is fairly low for us this is acceptable. Additionally, by using Gaussians we gain the power of Bayesian inference when trying to match feature classes to each other. However, the down side is that computing the eigen values and vectors necessary for Gaussian fitting scales minimally as d^3 for dimensions and s^2 for the number of samples. This is due to the fact that computing such elements using the pseudo inverse method involves matrix inversion and multiplication. In order to avoid such large complexity we have implemented an approximation technique that scales minimally as d^2 for dimensions and s for the number of samples. It works by using orthogonal rotations to center and remove covariance from the data. By recording the processes, we can then compute the probability on data points by translating and transforming them in the same way to align with the data set.

The first step is to center the data about the origin. This is done by computing the mean and then subtracting that number from each data point. Next we compute approximate eigenvectors by trying to find the average vector from the origin to all data points. So for data point k we first compute the ratio between its distance l from the origin along dimensions j and i. This yields the ratio r_{ijk}.

(1.6) $$r_{ijk} = \frac{l_{jk}}{l_{ik}}$$

Next we find the distance u_{ijk} from the origin along dimensions j and i.

(1.7) $$u_{ijk} = \sqrt{l_{ik}^2 - l_{jk}^2}$$

by Summing the ratio of r_{ijk} and u_{ijk} for all k, we obtain a mean ratio that describes the approximated eigenvector along the dimensions i and j.

(1.8) $$m_{ij} = \sum_{k=0}^{k} \frac{r_{ijk}}{u_{ijk}}$$

A normalizer is computed as the sum of all the distances for all samples k.

(1.9) $$n_{ij} = \sum_{k=0}^{k} u_{ijk}$$

next we determine the actual angle of the of the approximated eigenvector along the dimensions i and j.

(1.10) $$\theta_{ij} = \tan^{-1}\left(\frac{m_{ij}}{n_{ij}}\right)$$

Once we have that, we can rotate the data set along that dimension and measure the length of the ellipsoid using a basic sum of squares operation. Thus, we compute ρ_{ik} and ρ_{jk} which is the data set rotated by θ_{ij}. Here ξ is the positions of data point k along the i dimension and ψ is the position of the data point along the j dimension. What we are doing here is rotating covariance out along each dimension so that we can measure the length of the eigenvalue. Thus, we iterate over all data points k and along all dimensions i and along $i+1$ dimensions j summing up σ as we go. We only sum j for $i+1$ since we only need to use one triangle of the eigenvector matrix since it is symmetric along the diagonal.

(1.11) $$i + 1 \le j$$

(1.12) $$\rho_{ik} = \xi \cdot \cos(\theta_{ij}) + \psi \cdot \sin(\theta_{ij})$$

(1.13) $$\rho_{jk} = -\xi \cdot \sin(\theta_{ij}) + \psi \cdot \cos(\theta_{ij})$$

Since the mean is zero because we translated the data set by the mean to the origin, variance for the sum of squares is computed simply as:

(1.14) $$s_{iij} = \sum_{k=0}^{k} \frac{\rho_{ik}{}^2}{n}$$

(1.15) $$s_{jji} = \sum_{k=0}^{k} \frac{\rho_{jk}{}^2}{n}$$

Each sum of squares is used to find the eigenvalue estimate by computing Euclidian distances. That is, by determining the travel distance of each eigenvector during rotation and combining that number with the computed sum of squares we can determine an estimate of the eigenvalue from triangulation. The conditional here is used because σ_{ii} is computed more than once with different values for θ_{ij}. Thus, σ_{ii} is the sum of all the products of θ_{ij} and s_{iij}.

(1.16) $$\sigma_{ii} = \begin{cases} s_{iij} \text{ iff } \sigma_{ii} = 0 \\ \sigma_{ii} + \left(\sqrt{s_{iij}} \cdot \cos(-\theta_{ij})\right)^2 \text{ otherwise} \end{cases}$$

(1.17)
$$\sigma_{jj} = \begin{cases} s_{jji} \text{ iff } \sigma_{jj} = 0 \\ \sigma_{jj} + \left(\sqrt{s_{jji}} \cdot \cos(-\theta_{ij})\right)^2 \text{ otherwise} \end{cases}$$

The end result is a non-standard eigenmatrix which can be used to compute the probability that a point lies in a Gaussian region. We do this by performing the same procedure on any new data point. That is, we take that data point and translate it to align with the eigenmatrix. Probability for the data point is then computed independently along each dimension thus eliminating further matrix multiplication during the probability computation.

2.4.3 Feature Contiguity, Biasing and Memory

Once features have been classified we want to use them to perform various tasks. These include target tracking, target identification and feature biasing. Thus from a measurement of features from time *t*, we would like to know if a collection of features at time *t+1* is the same, and as such either the same object or a member of the same object.

By using Bayesian methods we can link classes of features in one frame of a video to classes in the next frame by tying a class to another which is its closest probabilistic match. Additionally, we use the probability to bias how the non-parametric classifier and saliency work over consecutive frames. For NPclassify we add a sink into the density computation. That is, we create a single point whose location is the mean of a class with the mass of the entire class. Think of this as dropping a small black hole in a galaxy that represents the mass of the other class. By inserting this into the NPclassify computation, we skew the density computation towards the prior statistics in the last iteration. This creates a Kalman filter like effect that smoothes the computation of classes between frames. This is a reasonable action since the change in features from one frame to the next should be somewhat negligible.

3. METHODS AND RESULTS

3.1 Complexity and Speed

One of the primary goals of VFAT is that it should be able to run in real time. This means that each module should run for no more than about 30 ms. Since we are using a Beowulf cluster, we can chain together modules such that even if we have several steps that take 30 ms each, by running them on different machines we can create a vision

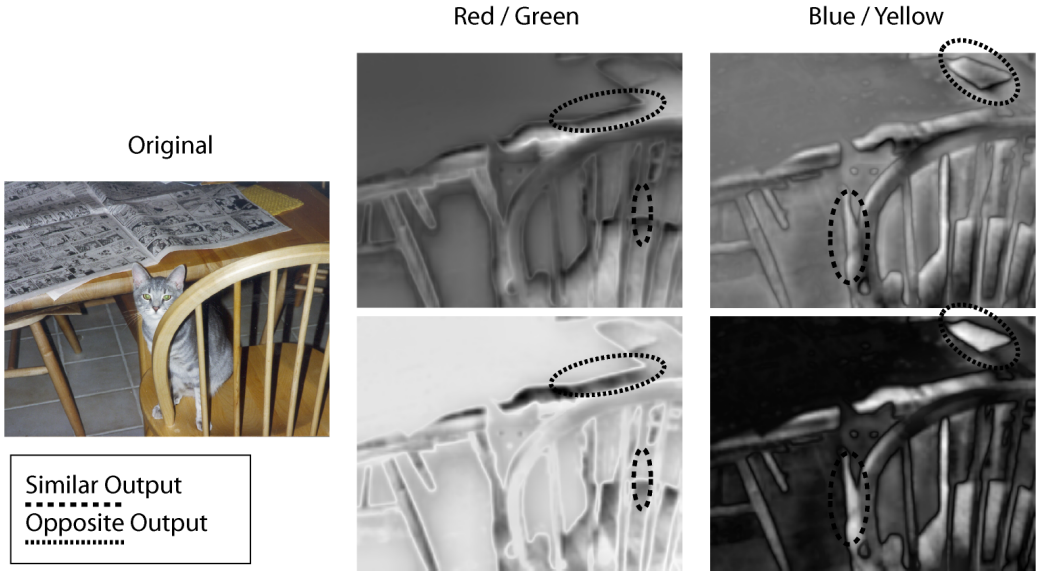

Figure 6: From the original image we see the results of ICA/PCA on the red/green and blue/yellow channels. As can be seen some parts of the outputs are negations of each other which makes sense since ICA maximizes mutual information. However, close examination shows they are not negatives. It is possible that scale information applies as a second input type and prevents obvious negation.

pipeline whereby a module finishes a job and hands the results to another machine in a Beowulf cluster that is running the next process step.

In time trials the modules run within real time speeds. Using a Pentium 4 2.4 GHz Mobile Processor with 1 GB of RAM, each module of VFAT runs at or less than 30 ms. The longest running module is the NPclassify feature classifier. If given only 300 features it runs in 23 ms, for 600 features it tends to take as long as 45 ms. The saliency code is also an important exception. This is because it already runs in real time, but requires a Beowulf cluster to do so. Thus, it has already been optimized in a similar fashion as VFAT will most likely be.

3.2 Training for classification

Two modules in VFAT need to be trained prior to usage. These include ICA/PCA and NPclassify. Training for both has been designed to be as simple as possible in order to maintain the ease of use goal of the iRoom project. Additionally and fortunately, training of both modules is relatively quick with ICA/PCA taking less than a minute using the FastICA algorithm under Matlab and NPclassify taking around two hours using gradient descent training. Since we only need to train once, this is not a prohibitive amount of time.

3.2.1 Training ICA/PCA

Training was completed by using 145 randomly selected images from a range of different image topics. Images were obtained as part of CD-ROM photo packages, which had the images sorted by topic. This enabled us to ensure that the range of natural images used in training had a high enough variety to prevent bias towards one type of scene or another. For each image, 300 features were extracted using the Monte Carlo / Visual saliency method described earlier. In all this gave us 43,500 features to train ICA / PCA on. The results are shown on table 1. For most channels, a reduction to two channels from six still allowed for over 90% of variance to be accounted for. However, directional channels that measure direction of motion and orientation of lines in an image needed three dimensions to still account for more than 90% of all variance. Assuming that the data is relatively linear and a good candidate for PCA reduction, this suggests that we can effectively reduce the number of dimensions while still retaining most of the information obtained from feature vectors.

Dimensions	1	2	3
Channels	Variance Accounted For		
Red / Green	88.7905	97.1248	98.7467
Blue / Yellow	86.6544	96.427	98.39
Luminance	80.8271	95.1773	97.7363
Orientation 0	52.3626	74.1747	91.5371
Orientation 45	64.4308	85.5804	95.4715
Orientation 90	57.9926	83.3922	94.501
Orientation 135	65.189	85.4022	95.3182
Complex Lines	61.7169	84.6236	96.2552
Complex Junctions	49.2324	76.6401	89.4178

Table 1: Following PCA the amount of variance accounted for was computed for each type of channel. Each channel started with six scales (dimensions). For many channels, 90% of variance is accounted for after a reduction to two dimensions. For all others, no more than three dimensions is needed to account for 90% of variance.

Visual inspection of ICA/ PCA results seems to show the kind of output one would expect (figure 6). For instance, when two channels are created from six, they are partially a negation to each other. On the red/green channel, one of the outputs seems to show a preference for red. However, the other channel does not necessarily show an anti-preference for red. This may suggest that preferences for colors may also depend on the scales of the images. That is, Since what makes the six input images to each channel different is the scale at which they are processed, scale is the most likely other form of information processed by ICA. This might mean for instance that the two channels of mutual information contain information about scaling. We might guess that of the two outputs from the red/green

channel, one might be a measure of small red and the other of large green things. If this is the case it makes sense since in nature, red objects tend to be small (berries, nasty animals, etc.) while green things tend to be much more encompassing (trees, meadows, ponds).

3.2.2 Training NPclassify

To hone the clustering method we use basic gradient decent with sequential quadratic programming using the method described by [4]. This was done offline using the matlab optimization toolbox. For this study, error was defined as the number of classes found versus how many it was expected to find. Thus, we presented the clustering algorithm with 80 training images. Each image had a certain number of objects in it. For instance an image with a ball and a wheel in it would be said to have two objects. The clustering algorithm would state how many classes it thought it found. If it found three classes in an image with two objects then the error was one. The error was computed as average error from the training images. The training program was allowed to adjust any of several hard or soft parameters for NPclassify during the optimization.

The training data was comprised of eight base objects of varying complexity such as balls and a wheel on the simple side or a mini tripod and web cam on the more complex side. Objects were placed on a plain white table in different configurations. Images contained different numbers of objects as well. For instance some images contained only one object at a time, while other contained all eight. A separate set of validation images was also created. These consisted of a set of eight different objects with a different lighting created by altering the f-stop on the camera. Thus, the training images were taken with an f-stop of 60 while the 83 validation images were taken with an f-stop of 30. Additionally, the angle and distance of view point is not the same between the training and validation sets. The validation images were not used until after optimal parameters were obtained by the training images. Then the exact same parameters were used for the validation phase.

Our first test was to examine if we could at the very least segment images such that the program could tell which objects were different from each other. For this test spatial interaction was taken into account. We did this by adding in spatial coordinates as two more feature vectors in with the original set of 14 ICA / PCA reduced features. The sum total of spatial features were weighted about the same as the sum total of non-spatial features. As such, the membership of an object in one segmented class or the other was based half by its location in space and half by its base feature vector composition. Reliability was measured by counting the number of times objects were classified as single objects, the number of times separate objects were merged as one object and the number of time a single object was split into two unique objects. Additionally, there was a fourth category for when objects were split into more than three objects. This was small and contained only four instances.

The results were generally promising in that based upon simple feature vectors alone, the program was able to

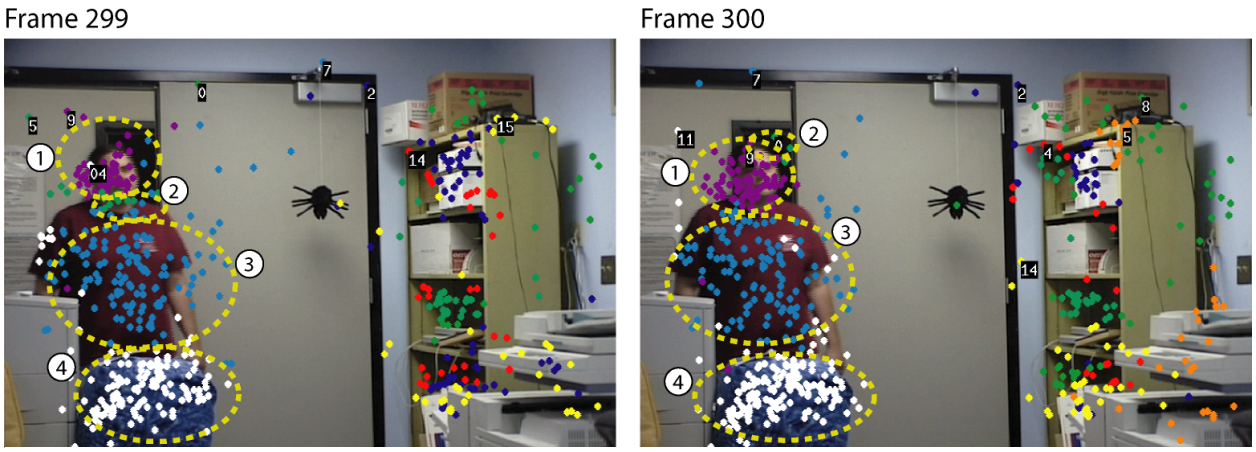

Figure 7: Tracking from frame 299 to frame 300 the shirt on the man is tracked along with the head without prior knowledge of what is to be tracked. It should be noted that that while the dots are drawn in during simulation, the elipses are drawn in by hand for help in illustration in gray scale printing.

segment objects correctly with no splits or merges in 125 out of the 223 objects it attempted to segment. In 40 instances an object was split into two objects. Additionally 54 objects were merged as one object. While on the surface these numbers might seem discouraging there are several important factors to take into account. The first is that the program was segmenting based solely on simple features vectors with a spatial cue. As such it could frequently merge one shiny black object into another shiny black object. In 62 % of the cases of a merger, it was obvious that the merged objects were very similar with respect to features.

3.3 Contiguity

Contiguity has been tested but not fully analyzed (figure 7). Tracking in video uses parameters for NPclassify obtained in 3.2.2. Thus, the understanding of how to track over consecutive frames is based on the computers subjective understanding of good continuity for features. In general, classes of features can be tracked for 15 to 30 frames before the program loses track of the object. This is not an impressive result in and of itself. However, several factors should be noted. First is that each object that VFAT is tracking is done so without priors for what the features of each should be. Thus, the program is tracking an object without having been told to either track that object or what the object its tracking should be like. The tracking is free form and in general without feature based priors. The major limiter for the contiguity of tracking is that an object may lose saliency as a scene evolves. As such an object if it becomes to low in saliency will have far fewer features selected for processing from it, which destroys the track of an object with the current feature qualities. Thus, the next module for VFAT will most likely be a working memory module that holds information about objects even after they have disappeared briefly.

4. CONCLUSION

VFAT seems to hold promise as a useful module in the iRoom. It is both efficient and easy to train. It also seems generally effective at creating useful classes of features. Since each module runs in about 30 ms, it should be able to run in real time mode on a Beowulf cluster. Additionally, it seems to have to ability to track objects without prior knowledge of what the objects are supposed to be like which it is tracking. Further development based on iRoom and VFAT will integrate more sophisticated tracking systems as well as memory systems.

ACKNOWLEDGEMENTS

I would like to thank Mike Olson for his help and suggestions. This research is supported by the National Imagery and Mapping Agency, the National Science Foundation, the National Eye Institute, the Zumberge Faculty Innovation Research Fund, the Charles Lee Powell Foundation and Aerospace Corporation.

REFERENCES

1. Hyvärinen A, 1999, Fast and Robust Fixed-Point Algorithms for Independent Component Analysis, IEEE Transactions on Neural Networks 10(3):626-634.
2. Itti L, Koch C, 2001, Computational Modeling of Visual Attention, Nature Reviews Neuroscience, 2(3): 194-203.
3. Mundhenk T N, Navalpakkam V, Makaliwe H, Vasudevan S, Itti L, (2004) Biologically inspired feature based categorization of objects, Proc. SPIE Human Vision and Electronic Imaging IX, San Jose, CA
4. Powell, M J D, 1978, A Fast Algorithm for Nonlinearly Constrained Optimization Calculations, Numerical Analysis, ed. G.A. Watson, Lecture Notes in Mathematics, Springer-Verlag, Vol. 630.