A review of work on modular standardization and color tracking

T. Nathan Mundhenk

Foreword: This document outlines the proposed iRoom Expert Module Standard (IEMS) as well as gives a review of a real time color tracker, which will be used as an initial first vision tracker in the iRoom. The IEMS standard should help illuminate how as more powerful trackers are developed they can be integrated into the iRoom as easily as possible. This document contains all new material some of it mostly technical. The table of contents is there so that one can skip such sections. The technical sections are for the groups benefit. Note: I treated this as an informal write-up.

Table Of Contents:

1 Color Tracker

The color tracker is designed to track objects based upon their color. It uses HSV color space since it separates hue from light intensity and is more invariant to lighting conditions. It works in a simple fashion by selecting pixels in an image which fall within certain color thresholds. It then links candidate pixels into contiguous discrete blobs. Each blob can then be edited out based upon its properties.

The color tracker then merges all non-edited blobs into a mother-blob. It finds the controid of the mother-blob and uses that to define the location to track. The color tracker works are 30 frames per second on a PC running above 1.5 GHz. It runs at reasonable speeds on any modern PC.

1.1 Color Tracker files

These are the primary files used by the color tracker. Note that this is the second version of the color tracker.

1.1.1 test-colorSegment2.C

This is the running binary for the color tracker. It sets up variables used by the color tracker. Additionally it creates connections to the camera controller and frame grabber. It works by first creating the tracking objects

```
segmentImageMerge2 segmenter(2);
```

This creates a single color tracker, but tells it to instantiate two trackers. That is, you create one tracker per image stream, but you can have several trackers per image. This allows you to track for instance skin colors and hair colors.

Next you will tell the tracker what colors to track off of.

```
segmenter.SIMsetTrackColor(13,20,0.17,0.3,156,30,0,true,15);
```

This tells the tracker the color range that is acceptable for a track in HSV color space. The inputs to this method in order are

- Mean Hue – This is the mean hue value for the tracker to track. This is a number between 0 and 360.
- Hue Bound – This is the value +/- the Hue mean which may be tracked. If the Mean hue is 13 than setting this to 20 means that any hue value that falls between 0 and 33 is considered a valid value.
- Mean Saturation – This is the mean saturation value for the color. This is a number between 0 and 1
- Saturation Bound – This works the same as the Hue bound.

Mean Luminance(Value) – This is the mean luminance value for the tracker. It is a number between 0 and 255.

Luminance bound – This is the same as the other bounds.

Instance – This is the tracker instance you are setting. If you set two trackers with the constructor than this is either a 0 or 1. If you set three than this would be a number from 0 to 2.

Use Color Adaptation – Set this to true if you wish to use color adaptation. I strongly recommend using this.

Average Interval for Adaptation – This is how many frames you want the color adaptation to average over. 15 would in essence set this to ½ a second if you are pulling 30 frames per second.

```
segmenter.SIMsetAdaptBound(50,0,.6,.10,170,50,0);
```

This creates hard boundaries for color adaptation. In essence, the color adaptation can conceivably adapt to all sorts of values. This keeps it from jumping off and tracking really weird stuff. Set this as a bound above and below which you know the color tracking would be absurd. The values are the upper and lower bounds for HSV.

Upper Hue Bound – This is a number 0 to 360 which is the highest hue value acceptable.

Lower Hue Bound – This is a number 0 to 360 which is the lowest hue value acceptable

Upper Saturation Bound – This is a number 0 to 1 which is the highest Sat. value acceptable.

Lower Saturation Bound – This is a number 0 to 1 which is the lowest Sat. value acceptable

Upper Luminance Bound – This is a number 0 to 255 which is the highest Lum. value acceptable.

Lower Luminance Bound – This is a number 0 to 255 which is the lowest Lum. value acceptable

```
segmenter.SIMsetFrame(0,0,width/4,height/4,width/4,height/4,0);
```

This method when Called sets a frame boundary for processing in every image. Each number here is devided by four since each image sent to the tracker is decimated (reduced to ¼ height ¼ width) from the original. Here, the entire image is frame is processed by the tracker.

Left of Frame – This is a number 0 to ¼ image size that is the left most of the frame

Bottom of Frame – This is a number 0 to ¼ image size that is the bottom most of the frame

Right of frame - This is a number 0 to ¼ image size that is the right most of the frame

Top of frame - This is a number 0 to ¼ image size that is the top most of the frame
Width of Image – This is the actual width of the image
Height of Image – This is the actual height of the image
Instance – This is the tracker instance (the same as above) that you are setting

```
segmenter.SIMsetCircleColor(0,255,0,0);
segmenter.SIMsetCircleColor(0,0,255,1);
segmenter.SIMsetBoxColor(255,255,0,0);
segmenter.SIMsetBoxColor(255,0,255,1);
```

These lines do not effect the functioning of the tracker. They instead define the colors used in feedback For instance, the first line states that a tracked object will be highlighted with a green circle (RGB 0,255,0) in the tracker instance 0, The box color is the bounding box drawn around the object being tracked (also RGB). More on this in section 2.

```
segmenter.SIMsetAdapt(3,true,3,true,3,true,0,true);
```

This defined the behavior of the color trackers adaptation. Specifically, whether a channel should adapt and how much breadth it should give new adaptation.

Standard deviation of hue – This is how many standard deviations out to consider the bound for hue. The tracker will work by finding the mean hue. This will then become the new hue value to track. Any pixel that has a hue value +/- *x* standard deviations will be tracked. As such this states that the mean hue value +/- 3 standard deviations will be tracked.
Use color adaptation on Hue – Turns on hue adaptation
Standard deviation of Saturation – This is how many standard deviations out to consider the bound for saturation. This works the same as for Hue.
Use color adaptation on Saturation– Turns on Saturation adaptation.
Standard deviation of Luminance – This is how many standard deviations out to consider the bound for luminance. This works the same as for Hue.
Use color adaptation on Luminance– Turns on luminance adaptation.
Instance – This is the tracker instance you are setting.
Use cluster on loss of track – When set to true, if a Loss of track is encountered, the new colors will be set based on clustering of values from the image. The mean and standard deviation from the cluster that most closely matches the initial HSV values set in SIMsetTrackColor is used as the template for setting new HSV colors. If this is set to false, on a Loss of track, colors are reset back to the colors set in SIMsetTrackColor. In essence, this just defines how to reset adapted colors when the target track is lost.

```
segmenter.SIMSetCluster(width/4,height/4,2,.80,.10,.10);
```

If you are using cluster on Loss Of Track this line defines some of the behaviors of the clustering algorithm. The other behaviors are set in NPclassify.conf. In essence, the most important thing to set here is the weighting for H,S an V channels. This means that when the cluster algorithm attempts to reset to H,S and V values, it will give more weight to one or the other when it creates its probability matching.

Image Width – This is the width of the image.
Image Height – This is the height of the image.
Number of tracker instances – Yes quite repetitive.
Hue Weight – A number from 0 to 1. The values for H,S and V should add up to 1
Saturation Weight – A number from 0 to 1. The values for H,S and V should add up to 1
Luminance Weight– A number from 0 to 1. The values for H,S and V should add up to 1

```
segmenter.SIMtrackImage(ima,&display,0,&Aux1);
```

This is the main tracking method. You will call this for each frame. ima is the image grabbed to be processed. It will be decimated to a size of ¼ . &display should be a pointer to an output image. This will contain an image with the bounding boxed and circles highlighting the target. &Aux1 should be a pointer to an image of size 100 x 450. This will contain the H,S and V adaptation tracking bars output.

```
segmenter.SIMmergeImages(&display);
```

Call this method to post processes the results of the tracker. You must call this if you wish to remove blobs that do not fit ideal constraints. It is recommended to call this. Hand it the pointer from SIMtrackImage.

```
segmenter.SIMgetImageTrackXY(&modi,&modj,0);
```

This will return the X and Y values of the center of mass for the mother blob that is left over from the track. This is the value which will be used to move the camera. Note also that these outputs may need to be transformed.

```
segmenter.SIMreturnLOT(0)
```

This is true if the tracker records a loss of track. Otherwise it is set to false. This is per instance.

```
segmenter.SIMreturnCandidateImage(0);
```

This returns a Boolean image that shows what image pixels were considered candidates to be tracked. This is a very useful image as it shows what pixels the tracker is actually tracking.

1.1.2 segmentImageMerge2.C

This is a method meant to do a few different things. First it is the method that has been called in all instances in section 1.1.

1. It manages the color adaptation
2. It calls segmentImageTrack to edit blobs
3. It calls segmentImage to find candidate blobs
4. It calls NPclassify if a Loss Of Track is noted
5. It can merge different instances of a track into a coherent track (thus it's called merge)

Basically, you will not have to worry much about its workings unless you wish to merge the results of several tracker instances or if you wish to change the method of color adaptation. Some parameters are adjustable and can be found in *blob.conf*. This method also contains experimental methods for multi camera tracking. Ignore these for the time being.

1.1.3 segmentImageTrack2.C

This method is responsible for editing blobs and for discovering a Loss Of Track. It is called by segmentImageMerge after it has found all candidate blobs. This method works by analyzing each blob to discover if it is trackable. Blobs are either marked as dead or live. Things that will kill a blob are:

1. Blob is too large are too small
2. Blob is too thin or too wide
3. Blob is to far from the last known good track. That is, in the last iteration, the final blob had a centroid of X,Y. This blob is killed if it's to far from that location.
4. Blob has to large or to small a mass

These parameters are set in blob.conf. The tracker looks at each blob and checks it against these criteria. If all blobs are killed, then a Loss of Track is noted. Otherwise, all left over blobs are merged into a mother-blob. The mother-blobs center of mass is noted and becomes the location *X,Y* to track and move the camera to.

NOTE: The values from blob.conf are stored in a single object of class blobProp. This means that these values are adjustable from within software.

1.1.4 segmentImage2.C

This is the center of the tracker, which may be called directly. By calling it directly you simply receive all the blobs without editing and color adaptation. It works by taking in an image and labeling each pixel as falling within threshold values for H,S and V. It can also be called on R,G and B values. Each pixel that can make up a blob is called a candidate pixel. The pixels are edited later for fitness. Here is the basic order of operation and a description of the process:

1. All pixels are scanned: If their H,S and V values all lie within threshold they are labeled as candidates. Currently, this does not take into account the modulo properties of Hue. This is in the works.
2. Edit pixels: If a pixel has no 4 connectivity neighbor, kill it. This removes pixels that do not touch any other pixel and consequently and safely cuts back on noise. If a pixel is a candidate from 1, it must be a candidate for two iterations to be a full candidate. Otherwise it is a potential candidate. What this means is that a pixel must be a candidate for two or more iterations in order to be considered in the following stages. This cuts back on noise and pixels, which may only appear for one iteration. In essence this acts as a band pass filter of sorts.
3. Link Pixels: All pixels which are still candidates are linked into blobs. This creates blobs of pixels which are contiguous and disconnected from other blobs. Each blob is then treated as a separate blob which allows editing in segmentImageTrack.

It is important to note that this method will only produce lists of candidate blobs given H,S and V color thresholds. Thus, segmentImageMerge will act to control the H,S and V values and segmentImageTrack will act to edit the blobs.

1.2 Running the Tracker

The tracker as is currently set is more sensitive to my skin color, you may need to set it more towards yours. Other than that, it is ready to run. Compile by using the –tprogs switch:

```
gmake –tprogs
```

It uses the model manager so you can look and see what model definitions you can pass to it. A basic run command would be something like:

```
Test-colorSegement2 –fg-type=1394
```

This tells the program to run using the firewire camera. Be sure the firewire is set up.

Figure 1 shows what the output of the tracker is giving you. You do not need to hook of a robot camera in order to run the tracker. It will not wait for the servo. Indeed, even when

the tracker is running on a robot camera, it will continue to track a target even while the camera is moving. To make the camera move smoothly a delay is introduced into the move command. That is, when the camera is ordered to move, it responds with the amount of time the movement will take. I timer counts down and once it expires, the camera can be moved again.

Interfacing between the camera and tracker is very straight forward. The tracker gives the camera control class an X,Y coordinate in the image and the camera controller figures out about how much it should move to put that coordinate in the center of view. In essence the camera controller will attempt to put the target in the center of frame.

1.2.1 How to make the camera move

To command to move the camera from the tracker is found in the lines:

```
if(camPause.get() > delay)
{
 int modi,modj;
 segmenter.SIMgetImageTrackXY(&modi,&modj,0);

 /* this is * 8 since we need to convert the image to
    the size 640 x 480 at the moment
    This will be fixed very soon!
 */

 modi = modi*8;
 modj = 480-modj*8;

 /* make sure out coordinates are OK */

 if(modi > 0 && modi < 640 && modj > 0 && modj < 480)
 {
  if(segmenter.SIMreturnLOT(0) == false)
  {
   camPause.reset();
   delay = camera->moveCamXYFrame(modi,modj);
  }
 }
}
```

This checks first if timer camPause has expired. It then fetches the mother-blob center of mass coordinates from SIMgetImageTrackXY. This is transformed for image size and if the coordinates are OK and no Loss Of Track is encountered in SIMreturnLOT, then the

camera is moved with moveCamXYFrame, which returns the delay for the next movement.

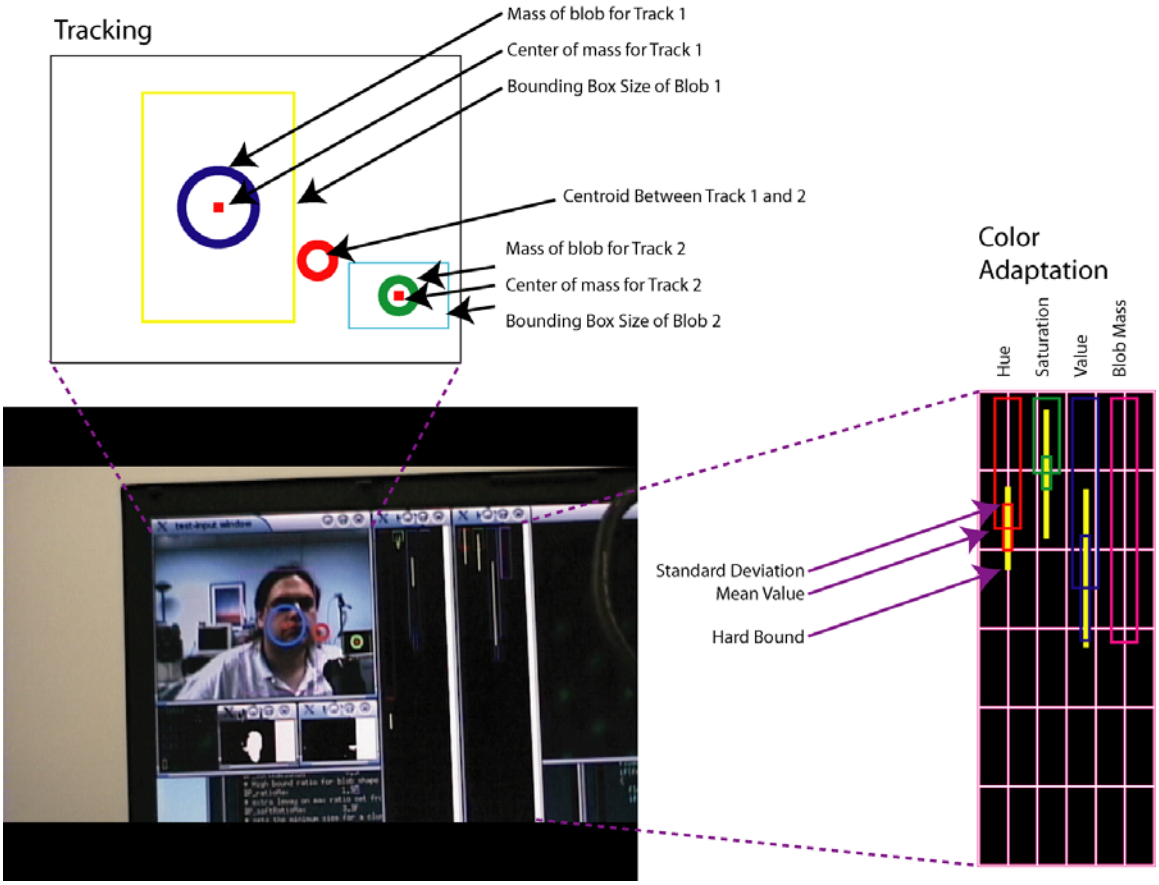


Figure 1. The windows created by the tracker show what is currently being tracked by each instance of the tracker. It also shows a bounding box which indicates the height and width of each mother-blob. A centroid is shown that shows the center between the two or more tracking instances. Color adaptation is also shown. This displays the color thresholds for H S and V by their main color and their high and low values computed from the standard deviation. The hard bounds for color adaptation is also shown. The smaller window shows the candidate pixels. If a pixel is a candidate it is colored white.

2 iRoom Expert Module Standard (IEMS)

2.1 Explanation

The iRoom expert module standard (IEMS) is being developed to accomplish several objectives. The first and most notable is it should be developed to allow as effortless as possible sharing of resources authored by different individuals as part of the iRoom development project. Additionally, it should set a standard for information among experts that forces the authors of modules to think about the extensibility of the information, which they can provide.

IEMS will be an attempt to create a standard which is flexible enough to allow for a variety of different kinds of experts, but will require a minimum standardization such that others can connect modules with as little effort as possible. As such what will be defined has nothing to do with the internal workings of each module. As such they will be for all intents and purposes black boxes. What will be defined is the resources shared. That is, the computation that goes into creating an image for instance is less important than that the image result is what a module states it should be. An example might be that a saliency map if shared can be created in any way the author sees fit so long as what comes out meets the definition of a saliency map. Eccentricities of that saliency map will be communicated if necessary via wrapping resource sharing.

The IEMS standard will in essence define the way in which resources are shared without defining the internal working of any module. Additionally, this means that modules may work on their own local time. If a global timing is required then a global timer will become a module and any module that wishes to be globally timed will communicate with that module. However, it should be ideal that all modules be semi-autonomous agents that need not have global timing. This creates a more libertarian environment where each module is assumed to be a trusted expert. As an expert it can have some free will to do what it thinks is best given the information at hand.

IEMS will define several important components in order to accomplish this. IEMS defines what kind of information and resources are to be shared and how they are to be shared. Some types of resources and information are required while others are optional. Additionally, each module should list what it can provide as well as what it needs. What is created is four essential components. First, the backend is a strict computational resource that is mostly private. It may be shared with other modules, but that is discouraged. It essentially is private link whereby modules obtain and process internal variables. The other side is the frontend. This is split between passive and active channels. As will be discussed later, these are either communications without overt intention to cause action, or communication intended overtly to cause action on another module. The forth type of information is the menu. This is the database that a module shares about it’s resource database.

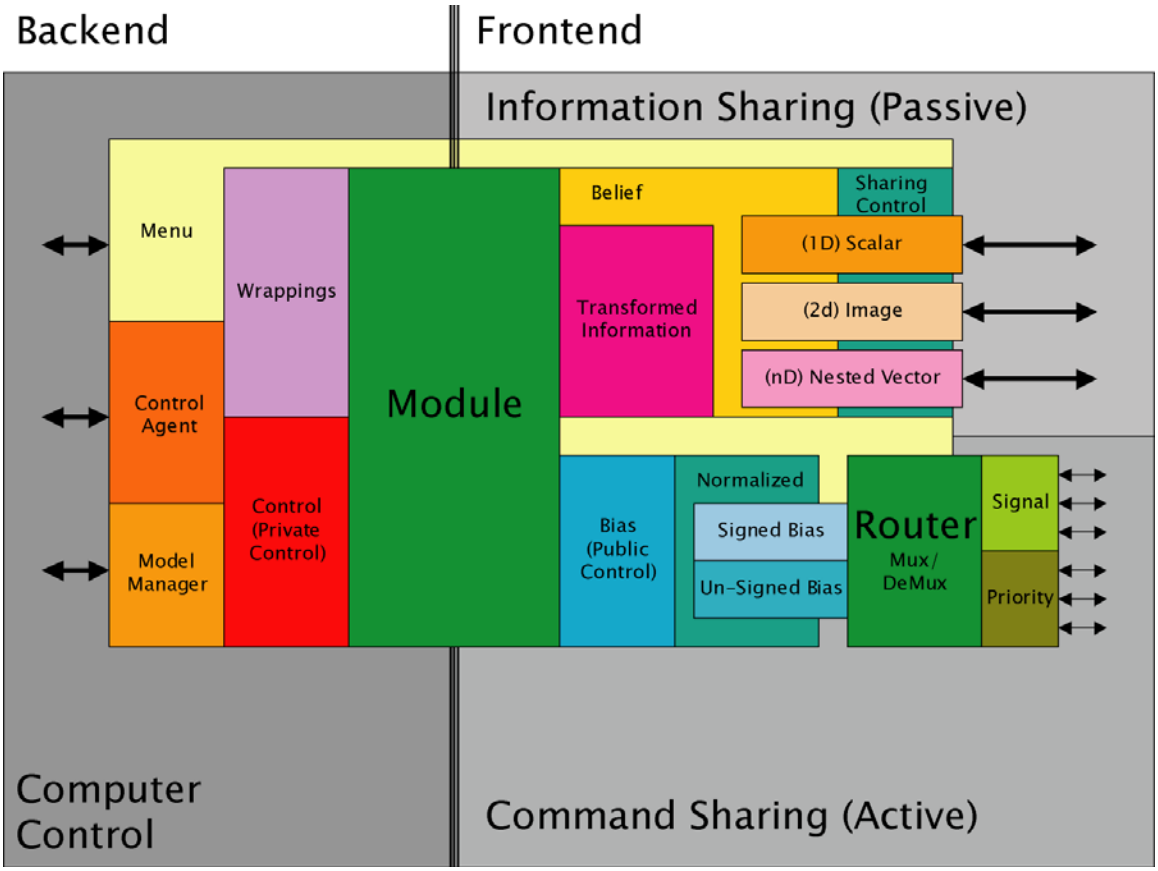
It is important to note that the interface to any module should appear to be a finite state machine. That is, each module should appear to the outside would as only a single thread. If it is multi threaded on the inside, this is fine, so long as this fact is hidden from other modules. This way, each module appears to work at a constant rate and on a single time frame. This way a module needs to understand less about the timing of other modules.

2.2 IEMS Components

2.2.1 Backend

The first type of data used in IEMS is the back end. This is information that is primarily private. These are control parameters and initial variables that need only be understood by the author. For instance, initial sizing of arrays may be passes via the backend. As such the backend can be defined however the module author sees fit so long as it does not interfere with the workings of other modules.

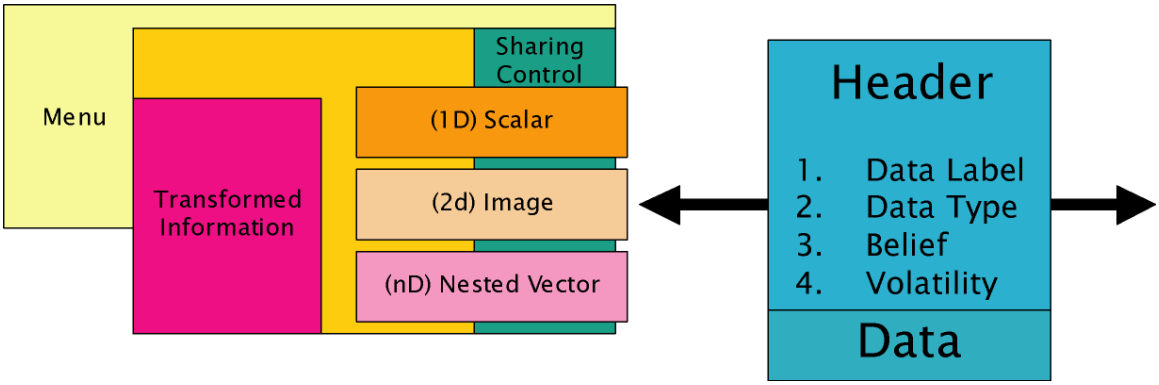
Backend resources are listed in the menu in so far as to prevent interference from other modules. However, no other module should need to know about the back end. What should understand the backend of a module might be a global initialization routine that starts the iRoom system. It will need to know how to start each module at the backend. The backend should then have an access point to a control agent which can launch it. This should be much like starting or stopping Linux services. Each module will have an interface with a start/stop command or simply a default constructor. The backend should



be further defined. However, the backend should hide complexities where possible.

2.2.2 Frontend Passive

The other two types of information passed by the module are the frontend. These are broken into passive and active components. Passive information can be thought of as information sharing and does not necessarily compel another module to act. Active information on the other hand is designed to attempt to change another modules behavior overtly. The difference between passive and active information can be thought of like this; Passive information might be “Hey did you hear the Romans are invading” while active information might be like “Lets get out of here”. The first only shares information, what the agent does with that information is up to the agent. In this case, the agent might like the Romans, so it decides to stay. The later example would be if your friend tries to



persuade you to leave. As such active and passive components can interact or be separate.

Passive information should be designed to maximize logical information in a manner that is coherent between modules. It is suggested that it be constrained to scalars, images and STL vectors. Additionally, each passive communication should be accompanied by a belief tag. This is the degree to which the module passing the information expresses confidence in the information. Since each module is supposed to be an expert, it must also express the confidence in its results.

The header should also contain control information. Such information might be the volatility and expiration of data. Since global timing is not enforced, each module must have information that allows it act upon information with the understanding of the timeliness of the information. As such each communication will carry a sequential number stamp. Perhaps a simple unsigned long integer. Additionally, the header will carry the performance specs of the module. This would be how fast information is being processed.

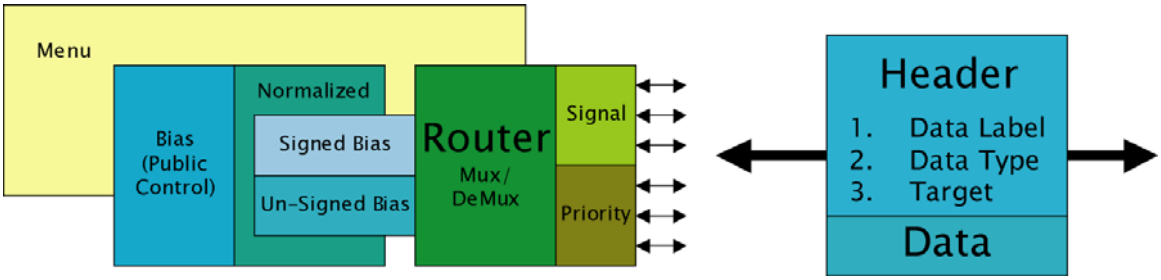
How information is shared must be decided. Should passive information be actively routed to any subscribing module or should a module merely place information in a common location? Both may be appropriate. Subscription based sharing is a problem if receiving modules are slower than the sending module while placed sharing creates

problems with mutual exclusion of resources such that modules will have to wait while the sending module writes to a common memory.

2.2.3 Frontend Active

Active information is in the form of biases. A routing/mixing layer will handle biases as they come in from multiple modules. A bias should be related to either internal states of a module that are known or frontend passive information that has been passed from another module. As such one module may simply compel another module to alter its behavior. The other module however, may choose not to be compelled. From a communication standpoint the receiving module must communicate this by either creating an echo about what it will do or the sending module must observe the receiving modules behavior and attempt to bias the receiving module even more if the wanted behavior is not observed. This means that any module may compel another module to act if it is connected and it acts with a strong enough bias. Bias information should probably be normalized to be either between –1 an 1 or 0 and 1. This will allow authors to know for sure the maximum and minimum biases that can be transmitted without having to spend time to look it up which can be a time consuming operation.

The active component can be combined with passive information. This might be the same as giving a passive piece of information a priority. For instance, “The Romans are invading, lets get out of here” would be and example of combined active and passive. Information is combined with a bias to compel a module to act on it. However, the module is only compelled and not forced. This means that active information is only a bias. The standard might define that it is the compelling modules duty to observe the target modules behavior to see that it is behaving in the way it wishes. If not, the



compelling module compels to greater a extent if it believes it needs to. It might be that another module is also compelling it. As such, a module may compel another module to not compel the first one. Biases then can set up a competition for resources. It is important that any bias itself can be biased so that competition can take place.

2.2.4 Wrappings using the Menu interface

Modules will integrate information and resource sharing via a wrappings protocol. Each module will have a *menu* where it will list what resources it has and needs in as simple and efficient a manner as possible. This will list for instance all the biases that are available as well as the information shared passively. This will allow either a software

agent or a human operator to connect modules as easily as possible. The menu will also contain backend information for control purposes. The backend menu components may be selectively distributed. That is, frontend menu items will be public to all but backend menu items will be classified in essence.

The menu might contain key value pairs of strings. Additionally, it might contain pointers to active and passive data sockets as well as information on data types used. It should be noted that the menu should not only be informative, but it should be efficient. Thus, it should allow other modules to collect shared pointers to data objects.

The menu might work as follows. When module B activates, it quarries module A. Module A hands module B the menu. Menu B selects the information is desires and copies a pointer to this data. Additionally, if data is on a subscription basis, the module B will give module A a pointer to where to place the data when it is finished. In essence, the menu will also keep track of pointers for routing data. This should be simplified as much as possible.

The menu may allow self-integration of modules or proactive integration by an organizing agent such as a person or another software expert. That is, it should allow the wrapping protocol to be used.

3 Integrating IEMS and iRoom Components

3.1 The color tracker

The color tracker described in chapter 1 is an expert on tracking objects using color data. However it is not very smart. IEMS if followed correctly will allow for a more sofistiacted tracker to be inserted such as the saliency based tracker currently under development. Alternatively IEMS should allow the two to work in concert as a mixed expert system. For instance, the color based tracker might augment the probability map used in the saliency based tracker. This might create a more accurate track. The color based tracker might also be used to create a mask for the saliency based tracker as well. This can happen since IEMS will dictate a standardized output for both the color tracker and saliency tracker. Thus, the saliency tracker will be integrated without changing the color based tracker.

3.1.1 The role of the color tracker in iRoom

The color tracker works by finding blobs of specified color, but has multiple statistical based adaptation routines. This will allow the handoff algorithm to track a single person and perhaps multiple people with different colored shirts in real time. The color tracker has been augmented to make it more robust and is being augmented to add certain likely components for the final version of IEMS.

The color tracker is currently operational and ready to track targets. The document contains the how-to written on using the color tracker. Further, tests of the trackers features can be viewed online at www.nerd-cam.com.

The color tracker will act to find a person in the room based on skin color. It works by taking in a raw image and returning the coordinates in the image where the target centroid is located. Handoff by trackers will be handled by Jacob et al. and should be highlighted in his write up. Additionally, the color tracker will work as an expert along side in cooperation with Nitin's color tracker. This will be done by trading off the Color trackers performance with the tracker designed by Nitin's accuracy. For instance one experiment will use the color tracker to create a mask for Nitin's skin tracker which will then hand back enhanced color values. Using this idea in mind, the clustering algorithm NPclassify has been used to characterize colors in a statistical fashion if a Loss Of Track is detected by the color tracker.

3.1.2 The color tracker as an IEMS module

No single component is in IEMS format as the standard needs to reach final agreement. However, preliminary decisions allow modules to be augmented in preparation for IEMS.

As such the color tracker has been augmented to have bias slots and provide a probability of good track. This will allow a coordinating hand off module to make decisions about how to treat the tracking from individual cameras. For instance, a camera with a high probability is a more trustworthy camera from which to acquire a track on a moving target.

As the target moves through the room, each camera will run a color tracker module. This will report back to the coordination module(s) which will decide based upon information from the cameras, which one to use for tracking of the moving target.

3.2 Saliency Based Tracking

The saliency based tracker complete but untested. Thus, the color tracker will be used while the saliency based tracker undergoes the next phase of development. When it is completed it will fulfill a similar role as that of the color tracker. However, it should be more robust and allow for more sophisticated biasing of features to track more different kinds of objects. Additionally, the saliency based tracker will have the ability to learn the individual features of targets. This separates it from the color based tracker which only learns in the short term based on adaptation. The color tracker must be told what to look for.

The saliency based tracker will use a combination of feature clustering, independent component analysis and Bayesian methods to learn new targets. Localization developed by Pradeep will enhance it's ability to learn entity specific traits. That is, knowing where something is will help it to learn if something is the same thing. This will integrate with the handoff modules developed by Jacob to allow tracking of multiple objects simultaneously. It will also enable scene understanding by storing information about a variety of objects in the room. For this, the new IEMS module will need to be created which can understand the interaction between entities.

4 Who Moved the Camera? Agency in the iRoom.

It became obvious based upon discussion that several different modules may want to move the camera. The standard method in this case is to create a mutex lock that allows only one resource to move the camera at a time. However, we propose a Brain Operating Principle solution. The idea is similar to who gets to move your arm in the brain and links back to agency.

Each module can move the camera. Similarly, different modular experts in the brain can give move commands to the arm. This is evidenced in the diverse number of regions related to alien hand phenomenon. Additionally, this creates the necessity of agency in action creation. Thus, the modules that can move the camera will cooperate with an executive to understand who what made the camera move. This means that the camera complex should be able to know if it moved a camera or if an external agent moved the camera. This will allow me to simulate a schizophrenia in the camera room and create the foundation of my thesis. That is, each module will have some power to move the camera. Cameras may also be moved by external agents. A central executive will have to decide if the camera was moved by one of the modules and if so which one. The modules will moves the camera based upon inhibition and gating factors that they will project onto each other. Thus, the wrong module may be able to get control of the camera, but the central agent will still understand that it created the action much the same as in Turret's syndrome. Additionally, we will explore under what circumstances the camera can move and the central executive does not understand that it did it. The groundwork here must be in creating a realistic system such that a realistic augmentation of the system can create a schizophrenic situation.

4.1 Challenges of Agency Experiments in the iRoom

It may be unrealistic to create a real model of PFC/BG/AMYG/HIPPO etc. interactions to create the agency experiment. Instead it should suffice to have competing and cooperating modules working to control the camera. As such, the realism should come in the form of the types of interactions ,but not modules themselves. If the interactions are realistic they should create an agency dilemma for the executive algorithm solve. Realistic augmentation by interfering with the functioning of the executive should be the focus of an artificial lesion. Ideally, the iRoom should also hallucinate an agent during the lesion condition. This may be accomplished by giving the executive a realistic working memory and some idea of the identity of agents that could act in the room. As such, given it's knowledge of agents and actions, will it hallucinate the same as a human?

Additionally, the iRoom agency experiment should integrate with the saliency code surprise mechanism. Thus, the agency code will have to understand surprise as part of its functioning.