eyeris

Final Report

Client
Stephen Gilbert, VRAC

Advisor
Daji Qiao

Project Members
Justin Derby, Kris Scott, Tyler Burnham,
Arjay Vander Velden, Will Bryan, Scott Connell

Project MAY13-20

Table of Contents

1. Introduction

1.1 Overview

Section 1 is the introduction and includes a description of the project and references
Section 2 provides a system overview.
Section 3 contains the system context.
Section 4 describes the system design method, standards and conventions.
Section 5 contains the component descriptions.
Section 6 contains the hardware components
Section 7 contains the test plan and results
Appendix A contains the project plan
Appendix B contains useful diagrams
Appendix C contains research information
Appendix D contains the user manual

1.2 Purpose

This document describes the design decisions made for Project Eyeris. It encompasses the hardware that will be used, the open source software that will be used, and the software that will be implemented. The goal of this document is to provide the necessary design knowledge for others to be able to replicate our system.

1.3 Scope

The product being produced is a mobile eye tracking solution that streams wirelessly. The system will include stereoscopic shutter glasses with an embedded computer system and a battery. Viewers of the study will be able to view what the user is seeing with added eye position data since we are streaming the data wirelessly. The product will also track both eyes in order to tell the depth at which the user is looking for 3D environments.

1.4 Definitions, Acronyms, Abbreviation

Term	**Definition**
User	The user wearing the glasses who is having his/her eyes tracked.
Administrator	The user administering the use of the glasses and running the server.
Viewer	User(s) who are able to watch the stream over the internet.
Raw Video	Video that is directly coming from the cameras
Encoded Video	Video that has been analyzed
Glasses	The glasses the user is wearing the is housing the cameras, embedded systems, and battery.

Embedded Board	The board that does the processing of the cameras and outputs the (x,y) coordinates of the eye-tracking data, as well as the compressed video from the out-facing camera. The local storage will also be on this board.
Broadcasting Server	Where the Administrator and Viewers will connect to see data and streams of the Glasses from the user
Embedded Computer System	The combination of the two Gumstix embedded boards and the embedded board to transmit the video data wirelessly all worn by the user
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
ISO	International Organization for Standardization
IEEE	Institute of Electrical and Electronics Engineers (read *I-triple-E*)
RANSAC	Random Sample Concensus (an ellipse fitting algorithm)
TBB	Intel® Threading Building Blocks for C++ which provides multi-threading for multi-core processors
Boost	C++ library that provides support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing

1.5 References

1.5.1 Calibration Free Algorithm

Ablassmeier, Markus. *Calibration-free Eye Tracking by Reconstruction of the Pupil Ellipse in 3D Space*. *Eye Tracking Research & Applications: Proceedings ; ETRA 2008 ; [Eye Tracking Research and Applications Symposium] ; Savanna, Georgia, USA*. By Stefan Kohlbecher, Stanislavs Bardinst, Klaus Bartl, Erich Schneider, and Tony Poitschke. New York, NY: ACM, 2008. 135-38. *Eye Tracking Research & Application*. 26 Mar. 2008. Web. 20 Oct. 2012. <http://www.informatik.uni-trier.de/~ley/db/conf/etra/index.html>.

1.5.2 Starburst Algorithm

Dongheng Li; Winfield, D.; Parkhurst, D.J.; , "Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches," *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on* , vol., no., pp.79, 25-25 June 2005. doi: 10.1109/CVPR.2005.531 <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1565386&isnumber=33215>.

1.5.3 ITU Gaze Group (IT University of Copenhagen, Denmark)

Hansen, John P., Dan W. Hansen, Javier S. Agustin, Sune A. Johansen, Henrik H. Tomra Skovsgaard, and Martin Tall. "ITU GazeGroup." *ITU GazeGroup*. ITU Copenhagen, n.d. Web. 24 Oct. 2012. <http://www.gazegroup.org/home>.

1.5.4 Japan Head-Mounted Display

Rodriguez-Silva, D.A.; Gil-Castineira, F.; Gonzalez-Castano, F.J.; Duro, R.J.; Lopez-Pena, F.; Vales-Alonso, J.; , "Human motion tracking and gait analysis: a brief review of current sensing systems and integration with intelligent environments," *Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2008. VECIMS 2008. IEEE Conference on* , vol., no., pp.166-171, 14-16 July 2008. doi: 10.1109/VECIMS.2008.4592774 <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4592774&isnumber=4592737>.

1.5.5 Pupil Tracking on Off-Axis Images

Swirski, Lech, Andreas Bulling, and Neil Dodgson. Robust Real-time Pupil Tracking in Highly Off-axis Images. University of Cambridge. ACM, 2012. Web. 24 Oct. 2012. <http://www.cl.cam.ac.uk/research/rainbow/projects/pupiltracking/>.

1.5.6 Overview of H.264

Wiegand, Thomas, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. "Overview of the H.264/AVC Video Coding Standard." *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY* 13.7 (2003): 560-576.

1.5.7 H.264 Standard

Kalva, Hari. "The H.264 Video Coding Standard." *IEEE MultiMedia* 13.4 (2006): 86-90.

2. System Overview

2.1 System Characteristics

The system will stream the out-facing view of the user to allow real-time analysis by the administrator and viewers. There can be several viewers at any one time, which means that there is a possibility of numerous concurrent users. The client side portion of the system will be platform independent to accommodate the different operating systems of the Viewers. The embedded board will store the out-facing video of the user’s view and the eye coordinates associated with those frames as a backup to curb loss in wireless connectivity or poor streaming latency. This will provide functionality to the administrator to download high-quality data for later analysis.

2.2 System Architecture

2.2.1 Software System Architecture

Our system will be based on a client-server architecture for streaming video. Clients will connect to the server to grab data and streams. The client will be written in c++ using the QT framework. The QT framework can but use to make cross-platform applications to not limit the type of operating system a viewer will need.

The internal camera software will be written so that it takes advantage of the hardware it will be on. It will be designed with multiple processors in mind.

2.2.2 Hardware System Architecture

The glasses will have three cameras: one outward-facing and two inward-facing, one per eye. Each camera will send their video feeds to the eye-tracking board. The eye-tracking board will analyze the video streams using the edge detection algorithm to determine the edges of the pupil. Once calculated, the edge points will be sent to the main embedded board. The outward-facing video feed will be pulled using the main board. This board will synchronize the outward video with the edge points, and send it over the wireless network to the server. The server will then run the fit ellipse algorithm using the given points to determine the direction the eye is looking. The server then paints the point onto the outward-facing video frame and sends to the client.

See Figure 4-1 for the functional block diagram of this system.

2.3 Infrastructure Services

2.3.1 Security

Since the system will be using a client-server architecture, security will be needed on the outward facing server end. Any data that gets sent to the server needs to be sanitized and error checked to make sure malformed, or illegal data cannot compromise the system.

2.3.2 Logging

The system should log everything to provide assistance in case anything unforeseeable happens. The levels of logging are as defined:

Number	Level	Description
-1	OFF	No messages will be logged
0	FATAL	Messages indicating a major failure, leads to shutdown
1	ERROR	Messages indicating a non-fatal error has occurred
2	WARN	Messages indicating a potential problem
3	INFO	Messages for informational purposes
4	DEBUG	Messages for fine grain debugging purposes

5	ALL	All messages will be logged, regardless of level

The default logging level will be 3 for INFO. All messages with a lower or equal level value as the logger is configured will be logged. Example is for level 3 (INFO), the following messages will be logged: INFO, WARN, ERROR, FATAL.

3. System Context

See Appendix B, High Level System Diagram

3.1 Server

3.1.1 Data

The data that the server will hold will be the video stream of the outward facing camera as well as the x,y components of the eyes.

3.1.2 Data Format

The video format will be encoded using H.264 RTP for the reason that it is a standard in the field and it is well-known and well-implemented. The x,y components of the eyes will be included in each frame of the video stream.

3.1.3 Failure

If no clients or administrators are connected, the video stream and data is dropped.

3.2 Ethernet Connection

The ethernet connection is used to connect the Gumstix and Pandaboard together.

3.2.1 Data

The data will be vectors of points from the Edge Detection algorithm passed via UDP socket.

3.2.2 Data Format

The data format will be serialized vectors of points including a camera id, size of vector, and points.

3.2.3 Failure

If the Ethernet connection fails between the Gumstix and Pandaboard, the Gumstix will continue to try to initiate communication and send points.

4. System Design

4.1 Design Method and Standards

We are using an object oriented design approach for this product. The benefit of this is for future code reuse and modularity. Modularity in our system will allow new features in the future to be added easier.

4.2 Documentation Standards

For each software file implemented, there should be a header containing a description of what the file does and any legal information it might have. For every class defined, there should be a description of the purpose of the class. Every function needs to be commented with the description, parameters, and return information. Additional comments inside the function body will be used at the developer's discretion. Commenting conventions that will be used will be Doxygen compliant. As example is as follows:

```
/**
* Summary here; one sentence on one line (should not, but can exceed 80 chars).
*
* A more detailed description goes here.
*
* A blank line forms a paragraph. There should be no trailing white-space
* anywhere.
*
* @param string $first
*   "@param" is a Doxygen directive to describe a function parameter. Like some
*   other directives, it takes a term/summary on the same line and a
*   description (this text) indented by 2 spaces on the next line. All
*   descriptive text should wrap at 80 chars, without going over.
*   Newlines are NOT supported within directives; if a newline would be before
*   this text, it would be appended to the general description above.
* @param int $second
*   There should be no newline between multiple directives (of the same type).
* @param bool $third
*   (optional) TRUE if Third should be done. Defaults to FALSE.
*   Only optional parameters are explicitly stated as such. The description
*   should clarify the default value if omitted.
*
* @return array
*   "@return" is a different Doxygen directive to describe the return value of
*   a function, if there is any.
*/
```

4.3 Naming Conventions

Our code will be using CamelCase for coding. Camel case is the practice of writing words with some inner uppercase letters, such as compound words or phrases in which the elements are joined without spaces, while each element has a capital letter within the compound. The first letter will always be capitalized for class declarations, all others will have the first letter not capitalized.

All files will include at most one class declaration in the upper level. Its respective header file (if the file has one) shall be named the same, but with a different file extension (.h or .hpp for example). If the file holds a class, it shall be named as the class name of the upper level.

4.4 Programming Standards

4.4.1 Coding Standards

The C++ standards that will be used is the ISO/IEC 14882:2011 standard. The C Standards that will be used is the ISO/IEC 9899:2011 standard or better known as C11 standard.

4.4.2 Internet Protocol Standards

The User Datagram Protocol (UDP) will be used to provide streaming for our application. The Transmission Control Protocol (TCP) will be used to provide commands throughout the application.

4.4.3 IEEE Standards

Since our system will be using wireless transmission, one of the standards we will be using is the 802.11 Wireless LAN standard (IEEE Std 802.11-2012). Also, since our system will be using video, the system will conform to the IEEE Standard on Video Techniques: Measurement of Resolution of Camera Systems (IEEE Std 208-1995). For H264 video encoding and decoding, the system will conform to the ITU-T H.264 standard.

Our team will be following the IEEE Systems and Software Engineering Standard for Architecture Description (ISO/IEC/IEEE 42010:2011(E)) as well as the IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE Std 828-2012)

4.4.4 Software Libraries

OpenCV 2.4.3

Open source image processing library.

Gstreamer 0.10

Open source multimedia streaming library.

Eigen 3.1.2

Open source matrix library.

Boost 1.51

Open source C++ library.

Qt 4

C++ graphical user interface framework

4.4.5 C++ Conventions

Include statements will be at the top of the file underneath the header. Macro declarations will be under the include statements.

Class Declaration

The following is how classes will be defined in a header file. The indent size will be set to 4 spaces.

```
class MyClass
{
public:
    void myFunction( Type paramOne, Type paramTwo );
protected:
private:

};
```

Method Declaration

This is how functions will be written in the source file. There will be no space between the function name and the parentheses containing the parameters. There will be a space between the parentheses and the first and last parameters. The indent size will be 4 spaces.

```
void MyClass::myFunction( Type paramOne, Type paramTwo )
{
    // code goes here
}
```

Alternative to be used when the function declaration has too many parameters to fit within the 80 character length.

```
void MyClass::myFunction( Type paramOne,
         Type paramTwo )
{
    // code goes here
}
```

If/Else if/Else Statements

Here is the template for If, else if, and else statements. There should be a space after the parenthesis and after the last value check before the closing parenthesis. The indent size will be 4 spaces. Assignment statements will never occur in the if or else if condition check.

```
if( value == someValue )
{
    // code goes here
}
else if( value == someOtherValue )
{
    // code goes here
}
else
{
    // code goes here
}
```

For loop

The increment value should be declared in the for declaration. A space should be between the open parenthesis and value declaration and the post loop increment and the closing parenthesis. Indent size will be 4 spaces.

```
for( int i = 0; i < someInt; i++ )
{
    // code goes here
}
```

While loop

The value being checked in the while condition should be declared outside the while loop. The value should never be assigned in the while condition. A space should be between the open parenthesis and the value as well as between the end of the condition check and the closing parenthesis. Indent size will be 4 spaces.

```
int value = 0;
while( value < 100 )
{
    // code goes here
}
```

4.5 Software Development Tools

For our project we are using netbeans for our c++ programming. We will be using c for our embedded programming, also we will be working on a linux kernel. Tools for forming our documents: Microsoft Word Microsoft PowerPoint and yed (for our diagrams).

4.6 Outstanding Issues

4.6.1 Depth Tracking

We ran out of time and didn’t have the entire system up and running to be able to correctly implement and test.

4.6.2 3D Eye-tracking

We wanted to use the one time calibration algorithm using two cameras per eye but due to priorities of implementing the rest of the system, this was never finished.

4.6.3 Startup scripts for Gumstix and Pandaboard

For usability, the Gumstix and Pandaboard should automatically start the program on boot up so the user doesn’t have to manually start it over serial or ssh.

4.6.4 New hardware

We would have liked to get some different hardware instead of the Gumstix. An alternative to using Gumstix would be to use another Pandaboard instead or check out http://www.hardkernel.com/renewal_2011/products/prdt_info.php for a more powerful and better supported embedded board.

4.6.5 Unknown cause of memory leak
We are experiencing a memory leak in the Gumstix and Pandaboard which we believe is caused by some memory not being freed properly by Gstreamer like it should. This needs further inspection since Gstreamer typically takes care of freeing the buffers but maybe there is something we are not freeing.

4.6.6 iMotions interface
We ran out of time to speak with the guys at iMotions but the code is written in a way that should be easy to write the data in the format they need to be able to interface with their software. The implementation should take place on the server.

4.6.7 Battery
We ran out of time to perform power characterization tests to get an accurate representation of full power draw. Also the Gumstix cannot be powered over USB so a custom battery would probably be more appropriate for this system.

4.7 Decomposition Description

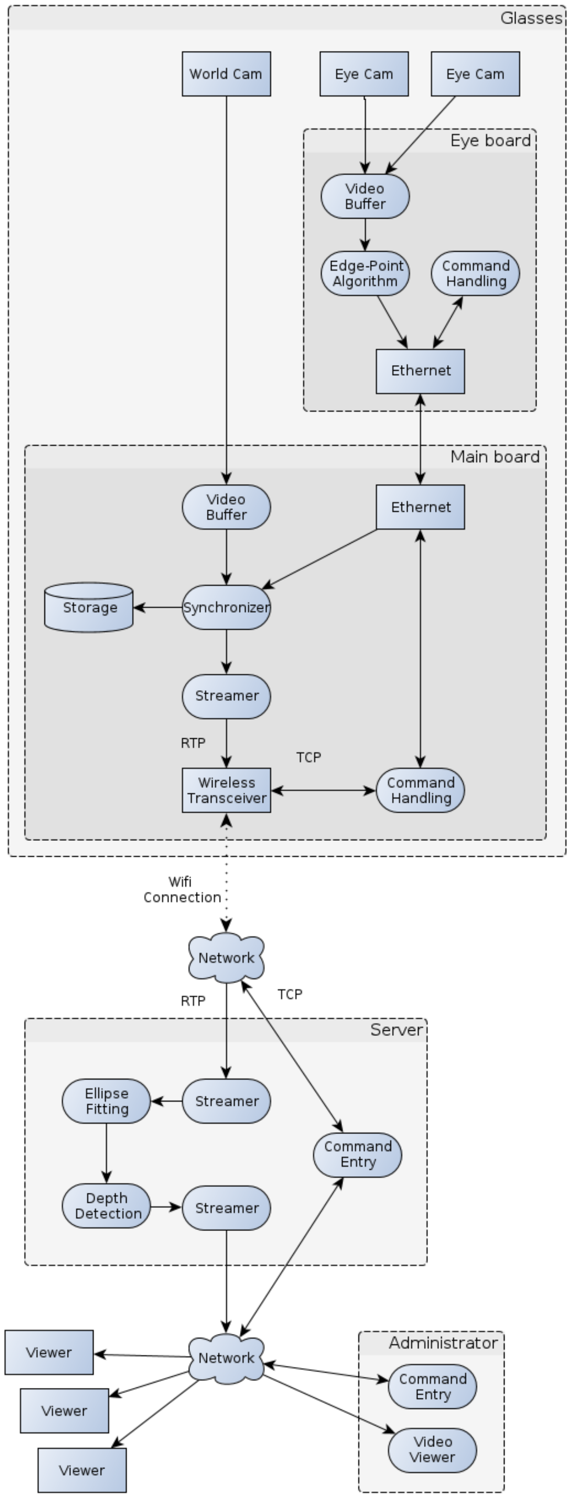


Figure 1: Functional Decomposition Diagram

5. Software Component Description

5.1 2DEllipse Module

5.1.1 Purpose

This module will be used to find the ellipse of the eye from the video frame.

5.1.2 Function

The functionality will be to grab the next frame from the Video Buffer module and find the ellipse of the eye from the image to provide to the eye tracking algorithm.

5.1.3 Subordinates

Eye-Tracking Algorithm Module

5.1.4 Dependencies

Have to have a frame in the Video Buffer.

5.1.5 Interfaces

```
/**
 * Calls to get the next frame and then analyzes it to detect the ellipse.
 * @return Ellipse object
 */
 public Ellipse getElipse(VideoFrame frame)


/**
  * Gets the identity matrix
  */
 void setIdentity();


 /**
  * Set the source matrix
  */
 void setSource(float x0,
         float y0,
         float x1,
         float y1,
         float x2,
         float y2,
         float x3,
         float y3);


 /**
  * Set the destination matrix
  */
 void setDestination(float x0,
         float y0,
         float x1,
         float y1,
         float x2,
         float y2,
         float x3,
         float y3);
```

```
/**
 * Computes the transform matrix to speed up calculations
 */
void computeWarp();

/**
 * Multiplies matrices
 */
void multMats(float srcMat[], float dstMat[], float* resMat);

/**
 * Uses matrix to map points from a unit square to a quadrilateral
 */
void computeSquareToQuad(float x0,
        float y0,
        float x1,
        float y1,
        float x2,
        float y2,
        float x3,
        float y3,
        float* mat);

/**
 * Uses matrix to map points from a quadrilateral to a unit square
 */
void computeQuadToSquare(float x0,
        float y0,
        float x1,
        float y1,
        float x2,
        float y2,
        float x3,
        float y3,
        float* mat);



/**
 * Gets the warp matrix
 */
float* getWarpMatrix();

/**
 * Warps the given source matrix to the destination matrix
 */
void warp(float srcX, float srcY, float &dstX, float &dstY);

/**
 * Warps the given source matrix to the destination matrix
 */
static void warp(float mat[], float srcX, float srcY, float &dstX, float &dstY);
```

5.1.6 Resources

N/A

5.1.7 References

See the reference 1.4.5 for an explanation of the algorithm being used.

5.1.8 Processing
RANSAC ellipse fitting algorithm
Haar-like features detection algorithm
Image histogram processing
OpenCV
Boost
TBB

5.1.9 Data
Ellipse which contains the data points that will construct an ellipse

5.2 Sockets Module

5.2.1 Purpose
The purpose of this module is to be able to interface with the sockets we are connecting to make videos.

5.2.2 Function
The function of this module is to be available to the synchronizer to be able to push the data through.

5.2.3 Subordinates
N/A

5.2.4 Dependencies
Must have a wired/wireless transceiver to connect with.

5.2.5 Interfaces
```
/**
 * Gets the address of this socket
 * @return
 */
sockaddr_in* getAddress();

/**
 * Sets the address of this socket using the sockaddr_in struct
 * @param addr
 */
void setAddress(sockaddr_in addr);
/**
 * Sets the address of this socket
 * @param ip in dotted string
 * @param port
 */
void setAddress(std::string ip, unsigned short port);
/**
 * Gets the socket file descriptor for this socket
 * @return
 */
int getSocketFileDescriptor();
```

```
   /**
    * Sets the socket file descriptor for this socket
    * @param sockfd
    */
   void setSocketFileDescriptor(int sockfd);
   /**
    * Adds the incoming socket to the list of connected sockets
    * @param socket
    */
   void addConnectedSocket(Socket* socket);
   /**
    * Removes the socket from the list. Index is the index in the list.
    * @param index
    * @return -1 for not deleted, 1 for deleted
    */
   int removeConnectedSocket(int index);

   /**
    * Grabs the pointer to the connected socket at the given index
    * @param index
    * @return pointer to the Socket, NULL if error
    */
   Socket* getSocket(int index);


TCP Socket
  /**
    * Initializes the socket by registering it with the OS
    * @return socket fd
    */
   int init();
   /**
    * Used for the TCP Client to connect to the provided address
    * @param to
    * @return
    */
   int connectSocket(sockaddr_in to);
   /**
    * Used for the TCP Client to connect to the provided address
    * @param ip
    * @param port
    * @return
    */
   int connectSocket(std::string ip, int port);
   /**
    * Listens for connections, used for TCP Server
    * @param numberOfConnections that can be queued
    * @return
    */
   int listenFor(int numberOfConnections);
   /**
    * Returns a socket fd for the connecting client. Used for TCP Server
    * @return
    */
   int acceptSocket();
   /**
    * Receives data on this when acting as TCP Client
    * @param buffer
    * @param size
    * @param clientAddr
    * @return
    */
   int receiveData(char* buffer, int size, struct sockaddr_in* clientAddr);


UDP Socket
  /**
    * Initializes the socket with the OS
    * @return
    */
   int init();
   /**
    * Receives the data, populates the client address
    * @param buffer
    * @param size
    * @param clientAddr
    * @return
    */
   int receiveData(char* buffer, int size, struct sockaddr_in* clientAddr);
```

5.2.6 Resources

The physical transceiver.

5.2.7 References

UDP and TCP protocols.

5.2.8 Processing

Running a real time stream through the wireless transceiver.

5.2.9 Data

Simple data that does not need to be stored other than.
Sockets in which we are connecting.

5.3 Configuration Module

5.3.1 Purpose

The purpose of this module is to be able to interface with the glasses and configure them to work with everyone’s eyes using different parameters.

5.3.2 Function

The configuration module will store the parameters of the application.

5.3.3 Subordinates

Client, Server, Gumstix, Pandaboard, Logger

5.3.4 Dependencies

Logger

5.3.5 Interfaces

```
 /**
  * Saves the configuration to the configuration file
  * @param key
  * @param value
  * @return true if saved
  */
 bool save_configuration( std::string key, std::string value );

 /**
  * Remove the configuration from the config file
  * @param key to remove
  * @return true if succeeded, false if key did not exist
  */
 bool remove_configuration( std::string key );

 /**
  * Search through the config file for the key
  * @param key what you are searching for
  * @param if you want to save the value found, pass in a pointer
  * @return index of key.  -1 if didn't exist
  */
int search_for_key( std::string key, std::string* value );

/**
 * Print out the entire config file for human consumption
 * @return true if printed, false otherwise
 */
bool print_all_configs();

/**
 * Get the Singleton instance
 * @return pointer to instance
 */
static ConfigModule* getInstance();

/**
 * Convert from string to datatype T
 * @param paramVal
 * @param defaultVal
 * @return
 */
template<class T>
static T convert(std::string paramVal, T defaultVal);

/**
 * Convert datatype to string
 * @param paramVal
 * @return string representation of input
 */
template<class T>
static std::string convert(T paramVal);
```

```
/**
 * Retrieve configuration saved in the config file
 * @param key what you want to retrieve the value of
 * @param default_value in case key does not exist, this value
 * will be set in its place
 * @return value
 */
template<class T>
T retrieve_configuration( std::string key, T default_value );

/**
 * Make the configuration entry into a single string
 * @param key
 * @param value
 * @return a string of the configuration entry
 */
template<class T>
static std::string stringify( std::string key, T value );

/**
 * Give back a configuration entry, given a string.
 * @param line stringified entry
 * @param key reference to key that it will save to
 * @param value reference to value that it will save to
 */
template<class T>
static void parseWithDelimiter( std::string line, std::string& key, T& value );
```

5.3.6 Resources

File
Memory

5.3.7 References

N/A

5.3.8 Processing

Writes and reads from a file different parameters of the application.

5.3.9 Data

N/A

5.4 GVideo Module

5.4.1 Purpose

The purpose of this module is to provide an easy way of interfacing with Gstreamer.

5.4.2 Function

The gvideo module will provide functions to easily link appSink/appSrc's together to talk to any custom code. It provides an easy way of grabbing any video data and manipulating the images using OpenCV.

5.4.3 Subordinates

Client, Server, Pandaboard, and Gumstix codebases.

5.4.4 Dependencies

Gstreamer version 0.10
Gstreamer App library version 0.10

5.4.5 Interfaces

```
/**
 * Creates a pipeline from the given strings
 * @param pipeline0 First pipeline
 * @param pipeline1 Second pipeline (Optional)
 */
void createPipeline(std::string pipeline0, std::string pipeline1 = "");


/**
 * Starts the pipelines up
 * @param async Set to true to run on another thread
 */
void startPipe(bool async = true);


/**
 * Stops the pipes
 */
void stopPipe();


/**
 * Send signal to pause pipelines
 */
void pausePipe();

/**
 * Send signal to play pipelines
 */
void playPipe();

/**
 * Gets the pipelines that were created
 * @param pipelineNum Which pipeline to grab (0 or 1)
 */
GstElement* getPipeline(int pipelineNum = 0);


/**
 * Register an appSink callback handler
 * @param sinkPipeNum Pipeline that this appSink is on (0 or 1)
 * @param appSinkName appSink name within the pipeline
 * @param appSinkCallback Function to call when appSink gets a frame
 * @param userData Any user data to pass into the appSinkCallback function
 * @param appSrcName appSrc name within the OTHER pipeline
 */
void registerAppSinkCallback()

/**
 * Tries to grab a gstElement from the corresponding pipeline
 * @param pipeline Which pipeline to grab from (0 or 1)
 * @param elementName Element to try and grab
 * @return The element, if found
 */
GstElement* getElement(int pipeline, std::string elementName);
```

```
/**
 * Pushes a buffer to an appSrc
 * @param appsrc appSrc to push to
 * @param buffer Image buffer
 * @return Status of pushing the buffer to the appsrc
 */
GstFlowReturn pushBuffer(GstElement *appsrc, GstBuffer *buffer);

/**
 * Clears the appSink callback
 */
void unregisterAppSinkCallback();

/**
 * Registers the bus callback to a custom function
 * @param busCallback Function to call for the bus
 */
void registerBusCallback(gboolean(*busCallback)(GstBus *bus,
                                                GstMessage *message,
                                                gpointer *ptr));

/**
 * Registers an appSrc callback (This does not need to be called if you
 * registered an appSink callback and handed it the appSrc also...)
 * @param sinkPipeNum Which pipeline to register the appSrc on (0 or 1)
 * @param elementName name of the appSrc element within the pipeline
 * @param on_pad_addedFunc Function to call when a pad has been added
 * @param userData User data to pass to the on_pad_addedFunc when called
 */
void registerNewSrcCallback();

/**
 * Clears the appSrc callback
 * @param sinkPipeNum Which pipeline (0 or 1)
 * @param elementName appSrc element to clear
 */
void unregisterNewSrcCallback();

/**
 * Clears the registered bus callback
 */
void unregisterBusCallback();
```

5.4.6 Resources

N/A

5.4.7 References

See Gstreamer general documentation.

5.4.8 Processing

N/A (Depends on what is using the module).

5.4.9 Data

N/A

5.5 Message Module

5.5.1 Purpose

The purpose of the Message module is to provide a consistent interface for communicating between the different nodes in the system (the Pandaboard to the Server, the Server to the Client, and the Client to the Server/Pandaboard).

5.5.2 Function

It provides a consistent structure with enums to specify what type of message is being sent so that the implementation may decide easily what action to take when receiving this message.

5.5.3 Subordinates

Implementations of the Pandaboard, Server, and Client (in the launcher.cpp)

5.5.4 Dependencies

TCP connections between the Pandaboard, Server, and Client(s).

5.5.5 Interfaces

```
/*
 *    Sets the message and the size of the message
 */
void setMessage( std::string msg, size_t size );

/*
 *    Sets the header
 */
void setHeader( size_t hdr );


/*
 *    Returns the header
 */
size_t getHeader();


/*
 *     Returns the message
 */
std::string getMessage();

/**
 * Sets the size of the message data
 * @param size
 */
void setSize( size_t size );
/**
 * Returns the size of the message data
 * @return
 */
size_t getSize();
```

```
/**
 * Sets the data type associated with the message
 * Use enum message_type
 * @param type
 */
void setType( size_t type );

/**
 * Returns the data type associated with the message
 * @return
 */
size_t getType();

/*
 *    Sets the header and the message at the same time
 */
void setHeaderAndMessage( size_t hdr, std::string msg, size_t size );
```

5.5.6 Resources
Network

5.5.7 References
N/A

5.5.8 Processing
Constructs, serializes, and deserializes uniform messages.

5.5.9 Data
Uses custom Message data and strings to store and serialize.

5.6 Utils Module

5.6.1 Purpose
The purpose of this module is to provide basic utils that do fit into any other module, but may be used by multiple.

5.6.2 Function
The utils module will provide misc. utilities to any code that might need it.

5.6.3 Subordinates
2DEllipse, and Server codebases.

5.6.4 Dependencies
Boost

5.6.5 Interfaces
```
/**
 * Push an element onto the queue
 * @param element
 * @return
 */
bool push(const T& element)
```

```
/**
 * Poll from the front of the queue. Blocking and
 * it removes the element from the queue
 * @param ret
 * @return
 */
bool poll(T& ret)
/**
 * Shows the front element of the queue but doesn't remove it.
 * Blocking call
 * @param ret
 * @return
 */
bool peek(T& ret)
/**
 * Sets the state
 * @param isRunning
 */
void setState(bool isRunning)
/**
 * Empties the queue
 */
void emptyQueue()
/**
 * Returns the size of the queue
 * @return
 */
long unsigned int size()
```

5.6.6 Resources
N/A

5.6.7 References
N/A

5.6.8 Processing
N/A

5.6.9 Data
N/A

5.7 Logger Module

5.7.1 Purpose
The purpose of this module is to be able to log what our code is doing.

5.7.2 Function
This will output to the admin when looking at the product running. Can be switched to multiple levels of outputs.

5.7.3 Subordinates
All modules are subordinates.

5.7.4 Dependencies

Configuration Module

5.7.5 Interfaces

```
/**
 * Log levels:
 *
 * The ALL Level has the highest possible rank and is intended to turn on all logging.
 * The DEBUG Level designates fine-grained informational events that are
 *   most useful to debug an application.
 * The INFO level designates informational messages that highlight the progress
 *   of the application at coarse-grained level.
 * The WARN level designates potentially harmful situations.
 * The ERROR level designates error events that might still allow the
 *   application to continue running.
 * The FATAL level designates very severe error events that will presumably
 *   lead the application to abort.
 * The OFF Level has the lowest possible rank and is intended to turn off all logging.
 *
 */
enum LogLevel {
    /**
     * The ALL Level has the highest possible rank and is intended to turn on all logging.
     */
    ALL = 5,
    /**
     * The DEBUG Level designates fine-grained informational events that are
     * most useful to debug an application.
     */
    DEBUG = 4,
    /**
     * The INFO level designates informational messages that highlight the
     * progress of the application at coarse-grained level.
     */
    INFO = 3,
    /**
     * The WARN level designates potentially harmful situations.
     */
    WARN = 2,
    /**
     * The ERROR level designates error events that might still allow the
     * application to continue running.
     */
    ERROR = 1,
    /**
     * The FATAL level designates very severe error events that will
     * presumably lead the application to abort.
     */
    FATAL = 0,
    /**
     * The OFF Level has the lowest possible rank and is intended to turn off logging.
     */
    OFF = -1,
};

/**
 * Get the shared Logger instance
 */
static Logger* getInstance();
```

```
/**
 * Sets the output stream of where the logger should write to
 */
void setOutputStream(std::ostream* stream);

/**
 * Log a message with the current log level
 */
void log(LogLevel level, std::string message);

/**
 * Sets the lowest log level that gets reported
 */
void setLevel(LogLevel level);
```

5.7.6 Resources

Files
Memory
Terminal

5.7.7 References

N/A

5.7.8 Processing

The processing will be distributed to where it is implemented.

5.7.9 Data

N/A

5.8 pointSerialize Module

5.8.1 Purpose

The purpose of the pointSerialize module is to create a consistent serialized data structure before and after being sent over the network.

5.8.2 Function

The pointSerialize module takes in the specified data structure and serializes it in order to be sent over the network or otherwise passed around.

5.8.3 Subordinates

Gumstix and Server.

5.8.4 Dependencies

The 2DEllipse and OpenCV

5.8.5 Interfaces

```
 /**
  *      A poorly-named struct that holds information to be serialized and sent
  *      from the algorithm board (Gumstix) to the server
  */
 struct whoknows
 {
     int x;
     int y;
     short cameraID;
     std::vector<cv::Point2f> pointvector;
 };

 /**
  *      A simple union to convert between an int and 2 shorts
  */
 union intToShort
 {
     int i;
     short s[sizeof(int)/sizeof(short)];
 };

/**
 * Easy union for conversions between a short and 2 bytes
 */
union ShortByteUnion {
    short asShort;
    char  asBytes[sizeof(short)];
};

/**
 *
 * @param input Array to convert
 * @param output a pointer in memory, this function will new an array for you!
 * @param shortLength Length of the short array (input)
 * @return the length of output
 */
int shortArrayToByteArray(short* input, char** output, int shortLength);

/**
 *
 * @param input Array to convert
 * @param output a pointer in memory, this function will new an array for you!
 * @param charLength Length of the char array (input)
 * @return the length of output
 */
int byteArrayToShortArray(char* input, short** output, int charLength);

/**
 * A function to return the defined header size
 * @return POINT_SERIALIZE_ROI_HEADER_SIZE
 */
int getPointHeaderSize();
```

```
/**
 * Takes the whoknows struct and puts in a short buff
 * @param w struct to serialize
 * @param buff destination buffer
 * @return buffer size
 */
int serializeROIpupilXY( struct whoknows* w, short** buff );

/**
 * Takes a buffer and puts the information into a whoknows struct
 * @param w destination struct
 * @param buff input buffer
 * @return cameraID
 */
short deserializeROIpupilXY( struct whoknows& w, short* buff );
```

5.8.6 Resources
N/A

5.8.7 References
N/A

5.8.8 Processing
Serializes and de-serializes the struct into a short buffer.

5.8.9 Data
The data will be all stored on the implementation end, this just provides a method of storage.

5.9 Edge Detection Module

5.9.1 Purpose
The purpose of this module is to provide an easy way of moving the point tracker (pupil edge detection) between any solution that needs to analyze eyes.

5.9.2 Function
The module will open a Gstreamer feed of each eye and runs the 2DEllipse fit algorithm on each eye. It then returns all the point data that is around the user's pupil.

5.9.3 Subordinates
Gumstix

5.9.4 Dependencies
2DEllipse, Gstreamer, Boost, OpenCV

5.9.5 Interfaces

```
/**
 * Starts the threads
 */
void start();
```

```
/**
 * Stops the threads
 */
void stop();
/**
 * Notifies the object to update the configuration parameters
 * Should be called when a set parameters message has been received
 */
void refreshParameters();
/**
 * Registers a function to call when writing the buffer out.
 * @param writeCallback
 */
void registerWriteCallback(int(*writeCallback)(const char* buffer, int size));
/**
 * Unregisters the function and writes out to default stdout
 */
void unregisterWriteCallback();
```

5.9.6 Resources
N/A

5.9.7 References
See section 1.4

5.9.8 Processing
We use the High Angle Pupil-Tracking Algorithm to gather the point data about each pupil.

5.9.9 Data
Returns a list of OpenCV points

5.10 Ellipse Fitting Module

5.10.1 Purpose
The purpose of this module is to take point data and fit it to an ellipse.

5.10.2 Function
To provide an easy way of fitting multiple points to an ellipse.

5.10.3 Subordinates
Server

5.10.4 Dependencies
OpenCV, 2DEllipse, Boost

5.10.5 Interfaces
We use the High Angle Pupil-Tracking Algorithm

5.10.6 Resources
N/A

5.10.7 References
See section 1.4

5.10.8 Processing
Depending on which ellipse fitting algorithm is used, OpenCV will perform ellipse fitting or a general RANSAC algorithm will retrieve the best fit for an ellipse.

5.10.9 Data
Returns an OpenCV rotated rectangle from analyzing a list of OpenCV points.

6. Hardware Component Description

6.1 Microsoft LifeCam HD-6000
The HD-6000 is a USB 2.0 camera. Its sensor can capture video at a maximum resolution of 1280x720 and a maximum framerate of 30 fps. Removed from its case, the camera assembly is just under 2.5 cm x 2.5 cm. This camera is supported by Video4Linux drivers.

6.1.1 Purpose
Two of these cameras captures the motion of both eyes. Its small form factor allows it to fit inside the active shutter glasses.

6.1.2 Function
The cameras are configured to stream YUV video at 160x120 @ 30fps.

6.1.3 Interfaces
Hardware: USB 2.0
Software: Video4Linux2 drivers
The drivers are used to manually control the focus

6.1.4 Power
Powered through USB

6.1.5 References
Product page: http://www.microsoft.com/hardware/en-us/p/lifecam-hd-6000-for-notebooks

6.2 Logitech C270
The C270 is a USB 2.0 camera. Its sensor can capture video at a maximum resolution of 1280x720 and a maximum framerate of 30 fps. Removed from its case, the camera as sembly is about 6 cm x 2 cm. This camera is supported by Video4Linux drivers.

6.2.1 Purpose
This camera captures video of the world as seen by the user.

6.2.2 Function
The camera is configured to stream YUV video at 620x480 @ 30fps.

6.2.3 Interfaces
Hardware: USB 2.0
Software: Video4Linux2 drivers
The drivers are used to manually control the focus

6.2.4 Power
Powered through USB

6.2.5 References
Product page: http://www.logitech.com/en-us/product/hd-webcam-c270

6.3 Gumstix DuoVero
The DuoVero Crystal COM features the dual-core OMAP 4430 Digital Video Processor running at 1GHz with 1GB RAM on board.

6.3.1 Purpose
The two Gumstix boards will interpret the images from the eye cameras and send data about the eye direction to the main board. Dedicating one Gumstix per eye will divide the work needed to process the video streams from the cameras and ensure that the images are processed as fast as possible.

6.3.2 Function
Each of the two Gumstix boards will read and interpret the images from the cameras. The boards will be running the images through the Ellipse and Eye-Tracking algorithms, and then send the X,Y coordinates to the main board over Ethernet.

6.3.3 Interfaces
The Gumstix uses the Parlor expansion board in order to provide an easily accessible interfacing peripheral. This board has on it USB ports that will be used to connect the eye-cameras, as well as an Ethernet port which will be used to communicate to the Pandaboard

6.3.4 Power
Powered via expansion board (DuoVero series or custom) connected to dual 70-pin connector.

6.3.5 References
Product page: https://www.gumstix.com/store/product_info.php?products_id=285

6.4 PandaBoard ES
Pandaboard ES is a low-cost, open OMAP 4 mobile software development platform. It's built around the Texas Instruments OMAP4460 processor.

6.4.1 Purpose
This serves as the central board which controls all of the other modules and components connected to the glasses. It then streams to the server.

6.4.2 Function

The Pandaboard will organize and communicate all of the necessary data. It will grab the outward-facing video feed and compress it. It will also grab the data from the edge-finding algorithm from the Gumstix. The Pandaboard will then send the points over UDP, along with the outward-facing camera's video stream, to the server.

6.4.3 Interfaces

The Pandaboard will connect to the Gumstix using an Ethernet local network connection. The video feed will be captured over the USB port from the camera. It will use its embedded wireless 802.11n to stream out the video and coordinates, as receive commands from the administrator.

6.4.4 Power

The PandaBoard can be powered at 3.7 - 5 V from a battery source.

6.4.5 References

Product page: http://www.pandaboard.org/content/platform

7. Test Plan and Results

7.1 Sub-Systems/Critical Components

1. Gumstix
2. Pandaboard
3. Cameras
4. Eye-Tracking modules
5. Configuration module
6. Streaming module

7.2 Test Strategy

Engineering Specification #	Metric	Acceptance Criteria	Tolerance Limit (+/-)
System Properties			
ES1	Cameras (FPS)	30	5
ES2	Eye-tracking Algorithm (FPS)	30	15
ES3	Eyes tracked	2	0
ES4	Cost (dollars)	1000	1000
ES5	Glasses (pounds)	2.3	.7
ES6	Backpack (pounds)	5	2
Needs			
ES7	Mobile	Yes	None
ES8	Real time	Yes	2 seconds
ES9	Boom	No	None
ES10	Synchronized	Yes	None
ES11	Non-obtrusive	Yes	None

7.3 Functional and Feature Tests

Test Number	Specification Number	Test Description
1	ES1	The system will run metric code to verify that we are getting the correct amount of frames.
2	ES2	The system will run metric code to verify that we are getting the correct amount of frames.
3	ES3	The eye-tracking algorithms will error if there is not two eyes incoming. We will test this when we are debugging the eye-tracking algorithm.
3.1	ES3	We will check that both cameras are picking up eyes for algorithm accuracy and accountability.
4	ES4	We will make sure to keep track of what we are spending.
5	ES5	We will use multiple scales to weight the product.
6	ES6	We will use multiple scales to weight the product.
7	ES7	The subject will move around with the system we will see if it still works.
8	ES8	We will measure the latency between the capture of the camera and seeing it on the client.
9	ES9	We will verify that the glasses don’t stick out from the face more than two inches. We will test this on many different subjects.
10	ES10	We will measure the latency between each eye.
11	ES11	The glasses will be tested by many users and give verbal response on the system obtrusiveness.

7.4 Test Equipment Available

1. Multimeter
2. Server
3. Router
4. Test subject
5. Cameras
6. 3D Screen with content
7. Scale

7.5 Design Test Verification

7.5.1 Results

1. Component – acceptable
2. Subsystem – acceptable
3. Integration – neutral
4. Reliability – Acceptable

7.5.2 Logistics and Documentation

Tests were performed in using a 3D monitor, a windows machine, and a pair of Nvidia stereo-shutter glasses. The windows machine was running a video game for content purposes only. We were able to test this thoroughly with our required specifications outlined in part one of this document.

7.5.3 Analysis of Data

Looking at our data we come within two feet of what the user is looking at. Our requirements are met except for the resolution requirements. This is due to an unneeded requirement we created.

7.5.4 Conclusion

Overall this product was a success, but many things can still be added. We have made our product extremely modular so anyone could add or take away features. The few issues we ran into were resolved and could be made more reliable with more time on this project.

Appendix A

Problem Statement

Mobile Eye-Tracking glasses are needed to be able to determine where one is looking at any given point in time. There are currently no mobile solutions that track both eyes, which would allow for 3D eye-tracking. Also, very few mobile solutions allow real-time streaming. This would allow multiple people to watch a study as it is being conducted rather than analyzing the data afterwards. There exhibits a need to stream this over WiFi networks so that this is compatible with multiple platforms and applications.

Concept Sketch

See Appendix B, Concept Sketch.

Functional Decomposition Diagram

See Appendix B,

High Level System Diagram

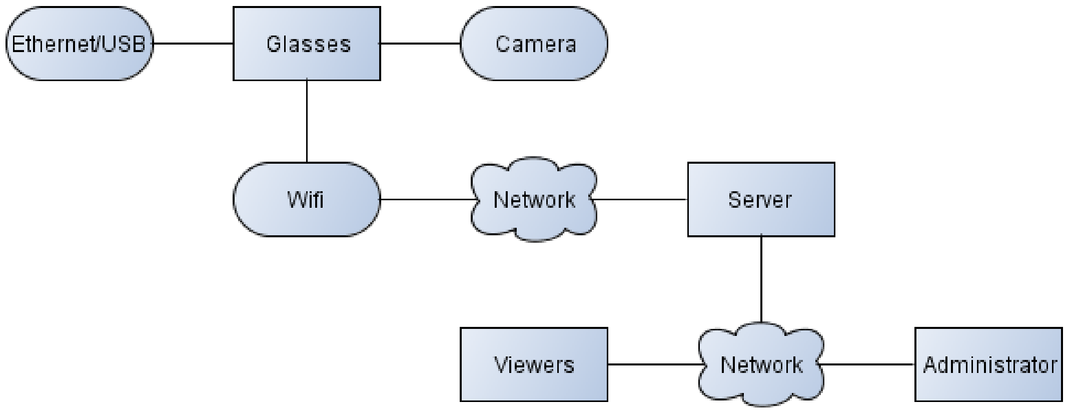


Figure B2: High level System Diagram

System Block Diagram

System Description

In order to accomplish the goal of streaming outward facing video with eye tracking data, we will be using two USB IR (Infrared) cameras – each facing an eye – to determine where the user is looking by calculating the convergence of the users' two eyes. The edge detection algorithm for each eye will be done on-board in the user's backpack. The edge points can then be sent to a server wirelessly for the ellipse fitting algorithm. The server then embeds the direction into the video feed of the point-of-view, outward facing high definition camera and is streamed wirelessly to the user.

Operating Environment

The solution will mainly be used in the C6 and Mirage 3D virtual environments which will be indoors and have strong WiFi connections. Ideally, our solution will be easy to adapt to a regular pair of glasses to be used outdoors or anywhere else given that there is strong WiFi for streaming.

User Interface

The device will stream video via WiFi to the client's computer. The client can connect to the device locally to run calibration and access recorded video.

The client software will display the outward facing video feed of what the wearer is seeing with a cross-hair of where the user is looking on top of the video feed, as well as the depth and accuracy rating. The client software will have the ability to provide calibration. The client software will also provide feedback about which eyes are being effectively tracked. There will also be numerous value/slider bars to fine-tune the eye tracking algorithm to the current user. If the client software loses connection to the eye tracking head gear, the client software will continually attempt to re-establish a connection to resume streaming. The glasses will save any footage and data on a hard drive that was not streamed.

Functional Requirements

- The glasses shall track both eyes.
- The software shall track the eyes at an error no less than 1.5 degrees.
- The outward-facing video and x,y coordinates for each sensor camera shall be cached on the on-board system for a 2 hour window.
- The outward-facing video data shall be transmitted wirelessly to client machines in real-time.
- The x,y coordinates shall be transmitted wirelessly to client machines in real-time.
- The real-time streaming shall have a delay of no more than 10 seconds.
- The client application shall display the x,y coordinates as an overlay to the outward-facing video data received.
- The client application shall be able to interface with the iMotions software.

Non-Functional Requirements

General

- There shall not be a boom extending in front of the face.
- The glasses used should be Active Shutter Glasses.
- The outward-facing video shall be displayed in color.
- The battery shall last for a time no shorter than three hours.
- The glasses should not be invasive or obstructive.

Video Quality

- The outward-facing camera shall be displayed at no lower quality than 720p (resolution of 1280×720).
- The sensor cameras shall be displaying frames at no lower rate than 30 fps.
- The sensor cameras shall capture video at resolution 640×480.

- The Field of View (FOV) shall be no less than 56 degrees horizontal by 40 degrees vertical.

Size Constraints

- The glasses shall be no wider than 10 inches
- The backpack shall not weigh an amount greater than 5 pounds (2268 grams)
- The glasses shall not weigh an amount greater than 2.3 pounds (1050 grams)

Market and Literature Survey

We have talked to various companies and people who are in the field of using eye tracking data, like iMotions and researchers at Iowa State University. They both expressed that this project will be very useful when it is made. We have also researched many different existing commercial solutions for eyetracking, and have experimented with existing products, such as the VT2, and the Mobile Eyes XG.

The following list is a review of current eyetracking solutions available in the market:

Tobii

- A very nicely packaged, small mobile unit.
- Very expensive (~$10,000 based on 2011 pricing data)
- Real-time analysis is done on the computer, not on the device.

Mobile Eyes XG

- Fairly heavy Mobile unit.
- Expensive.
- Requires a lot of calibration for each new user/session.
- Software interface is not user-friendly.
- Real-time analysis is not over Wi-Fi, only either on the handheld device itself or to the computer through Ethernet.

VT2

- Is not a mobile unit. Sits on a desk under a desktop's monitor, or under a laptop's screen.
- Requires a lot of calibration for each new session.
- Cheaper than the above solutions, but still expensive.

The following is the product that we plan to make:

Eyeris

- Inexpensive
- Un-intrusive/lightweight mobile unit.
- Will require very little calibration.
- Transmits analysis data over Wi-Fi in real-time.

Deliverables

- Eyetracking glasses prototype
- Receptive Software Interface
- Documentation and User Manual
- Demonstration of functionality

Work Plan

Work Breakdown Structure

We will be using the Agile software development methodology rather than the Waterfall methodology, so we will be using sprints, rather than a traditional Gantt-style process.

- Will and Kris – Build the hardware/ work on firmware between hardware and software.
- Tyler and Justin- Working with eye tracking algorithm and codec decoding.
- Arjay and Scott – working with intermediate between software and hardware, like codec compression and the streaming process.

Resource Requirements

This must be a relatively inexpensive project, but an exact budget is not defined.

Project Schedule

See Appendix B, Gantt Chart and Schedule List

Risks

- Dangers involving eye damage when exposed to Infrared Light.
- Dangers involving Lithium Ion batteries.

Appendix B

Concept Sketch

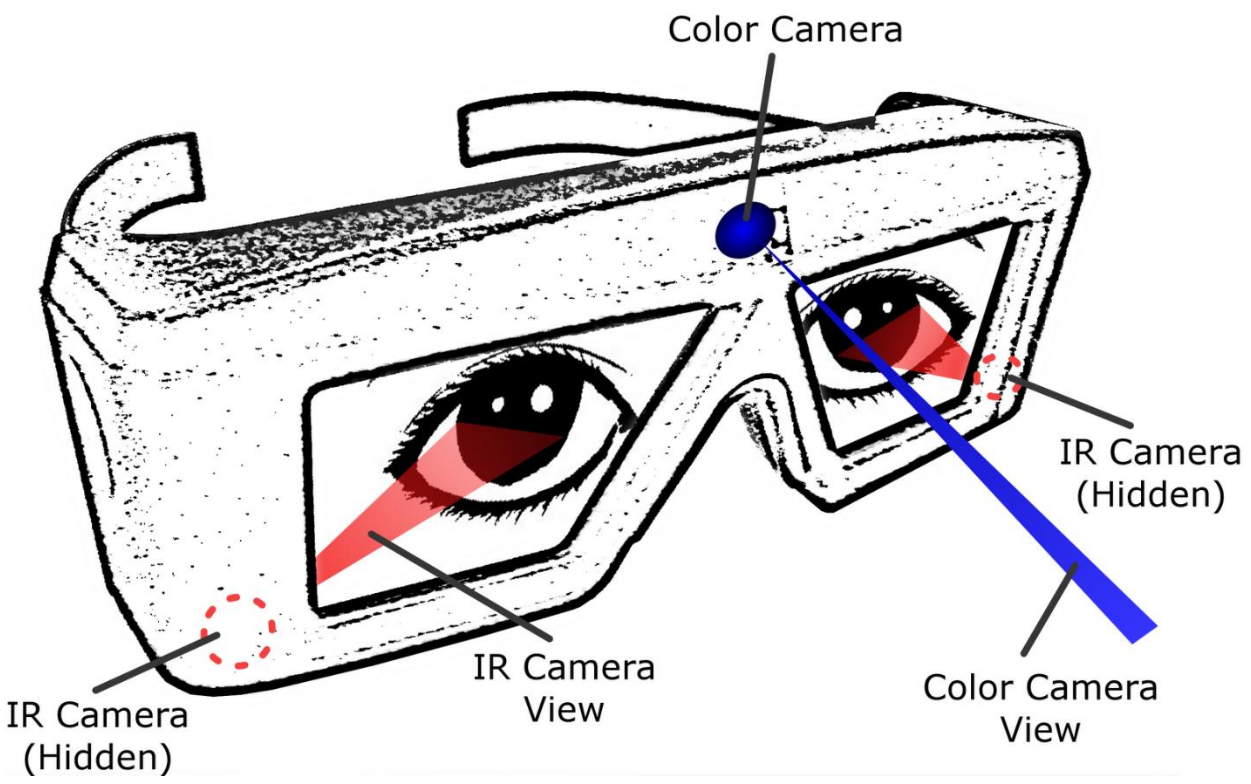


Figure B1: Concept Sketch

High Level System Diagram

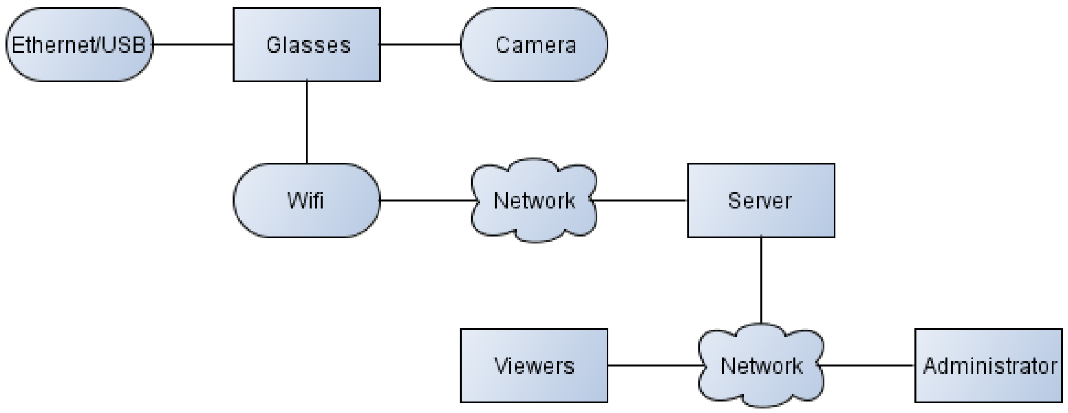


Figure B2: High level System Diagram

System Block Diagram

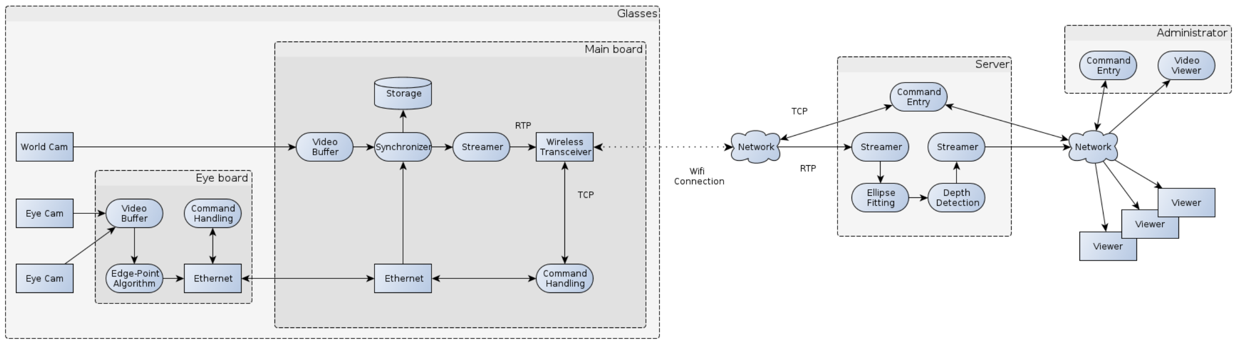
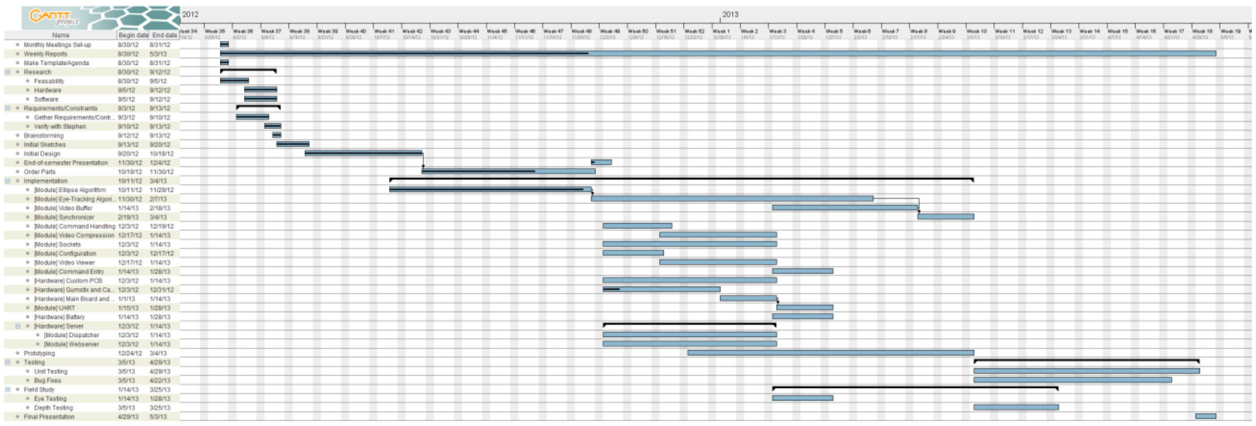


Figure B3: System Block Diagram

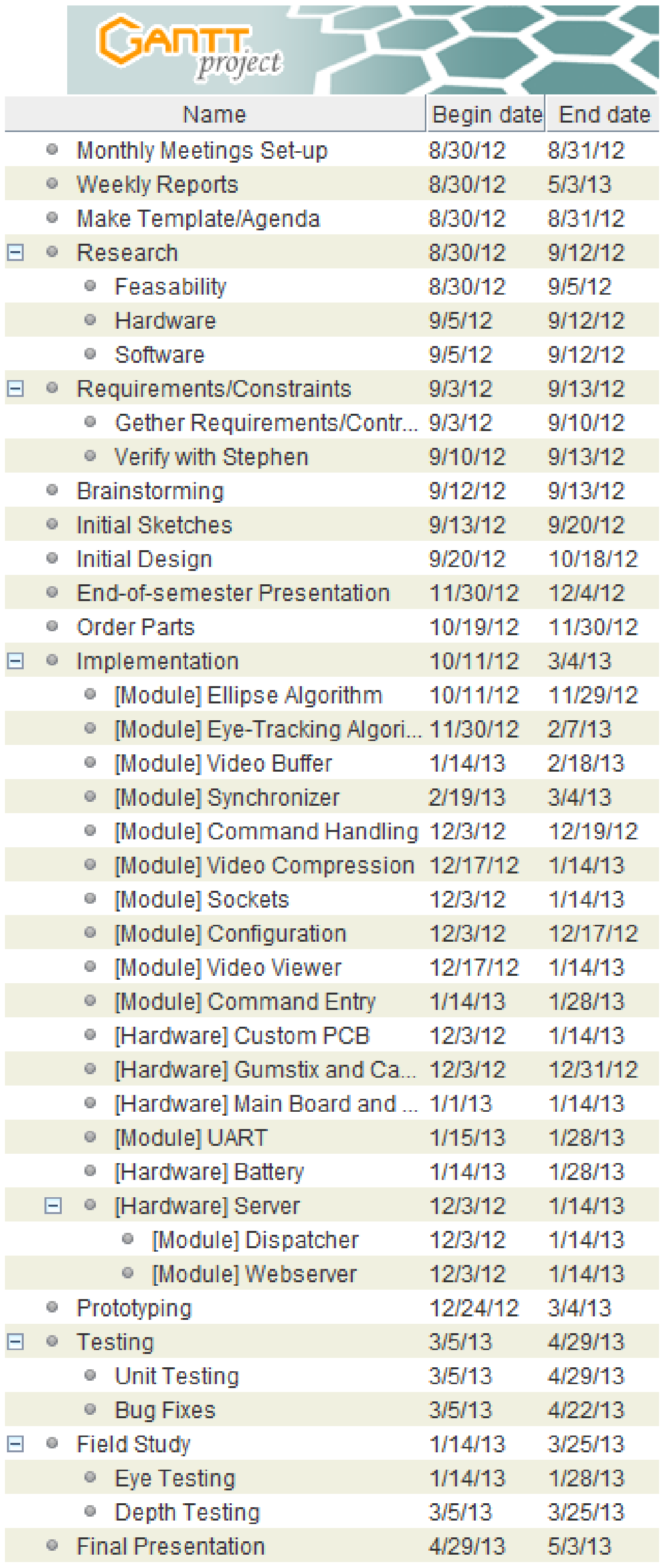
Gantt Chart



Name	Begin date	End date
Monthly Meetings Set-up	8/30/12	8/31/12
Weekly Reports	8/30/12	5/3/13
Make Template/Agenda	8/30/12	8/31/12
Research	8/30/12	9/12/12
Feasability	8/30/12	9/5/12
Hardware	9/5/12	9/12/12
Software	9/5/12	9/12/12
Requirements/Constraints	9/3/12	9/13/12
Gether Requirements/Contr...	9/3/12	9/10/12
Verify with Stephen	9/10/12	9/13/12
Brainstorming	9/12/12	9/13/12
Initial Sketches	9/13/12	9/20/12
Initial Design	9/20/12	10/18/12
End-of-semester Presentation	11/30/12	12/4/12
Order Parts	10/19/12	11/30/12
Implementation	10/11/12	3/4/13
[Module] Ellipse Algorithm	10/11/12	11/29/12
[Module] Eye-Tracking Algori...	11/30/12	2/7/13
[Module] Video Buffer	1/14/13	2/18/13
[Module] Synchronizer	2/19/13	3/4/13
[Module] Command Handling	12/3/12	12/19/12
[Module] Video Compression	12/17/12	1/14/13
[Module] Sockets	12/3/12	1/14/13
[Module] Configuration	12/3/12	12/17/12
[Module] Video Viewer	12/17/12	1/14/13
[Module] Command Entry	1/14/13	1/28/13
[Hardware] Custom PCB	12/3/12	1/14/13
[Hardware] Gumstix and Ca...	12/3/12	12/31/12
[Hardware] Main Board and ...	1/1/13	1/14/13
[Module] UART	1/15/13	1/28/13
[Hardware] Battery	1/14/13	1/28/13
[Hardware] Server	12/3/12	1/14/13
[Module] Dispatcher	12/3/12	1/14/13
[Module] Webserver	12/3/12	1/14/13
Prototyping	12/24/12	3/4/13
Testing	3/5/13	4/29/13
Unit Testing	3/5/13	4/29/13
Bug Fixes	3/5/13	4/22/13
Field Study	1/14/13	3/25/13
Eye Testing	1/14/13	1/28/13
Depth Testing	3/5/13	3/25/13
Final Presentation	4/29/13	5/3/13

Figure B4: Gantt Chart

Schedule List



Name	Begin date	End date
Monthly Meetings Set-up	8/30/12	8/31/12
Weekly Reports	8/30/12	5/3/13
Make Template/Agenda	8/30/12	8/31/12
Research	8/30/12	9/12/12
Feasability	8/30/12	9/5/12
Hardware	9/5/12	9/12/12
Software	9/5/12	9/12/12
Requirements/Constraints	9/3/12	9/13/12
Gether Requirements/Contr...	9/3/12	9/10/12
Verify with Stephen	9/10/12	9/13/12
Brainstorming	9/12/12	9/13/12
Initial Sketches	9/13/12	9/20/12
Initial Design	9/20/12	10/18/12
End-of-semester Presentation	11/30/12	12/4/12
Order Parts	10/19/12	11/30/12
Implementation	10/11/12	3/4/13
[Module] Ellipse Algorithm	10/11/12	11/29/12
[Module] Eye-Tracking Algori...	11/30/12	2/7/13
[Module] Video Buffer	1/14/13	2/18/13
[Module] Synchronizer	2/19/13	3/4/13
[Module] Command Handling	12/3/12	12/19/12
[Module] Video Compression	12/17/12	1/14/13
[Module] Sockets	12/3/12	1/14/13
[Module] Configuration	12/3/12	12/17/12
[Module] Video Viewer	12/17/12	1/14/13
[Module] Command Entry	1/14/13	1/28/13
[Hardware] Custom PCB	12/3/12	1/14/13
[Hardware] Gumstix and Ca...	12/3/12	12/31/12
[Hardware] Main Board and ...	1/1/13	1/14/13
[Module] UART	1/15/13	1/28/13
[Hardware] Battery	1/14/13	1/28/13
[Hardware] Server	12/3/12	1/14/13
[Module] Dispatcher	12/3/12	1/14/13
[Module] Webserver	12/3/12	1/14/13
Prototyping	12/24/12	3/4/13
Testing	3/5/13	4/29/13
Unit Testing	3/5/13	4/29/13
Bug Fixes	3/5/13	4/22/13
Field Study	1/14/13	3/25/13
Eye Testing	1/14/13	1/28/13
Depth Testing	3/5/13	3/25/13
Final Presentation	4/29/13	5/3/13

Figure B5: Schedule List

Appendix C

Eye Tracking Algorithm Research

Calibration Free Algorithm

The Calibration Free algorithm reconstructs the pupil ellipse for 3D space. It can do this by using two calibrated cameras per eye. It finds the intersection of two cones through a series of matrix operations.

Pros

- Developed for a mobile system
- Calibration free - you only need to calibrate it one time and then it will work on anyone.
- Accuracy within ~1.5 degrees based on how well one-time calibration is done and image noise
- 2D tracking to 3D projection (Can find convergence based on ray casts)
- Algorithm is specified (but not implemented) in the document

Cons

- Requires two cameras for each eye, which means more image analyzing
- Many variables need to be known before the one-time calibration
- Use of matrix and trig functions - could slow the algorithm and not provide real-time
- Needs another algorithm to provide the ellipses to do calculations - can be provided by numerous libraries.

Starburst Algorithm

The Starburst Algorithm is an open sourced algorithm that does dark-pupil infrared tracking. It is a hybrid between feature-based and model-based tracking which is a mixture for runtime performance and accuracy.

Pros

- Developed for a mobile system.
- Source code already available. Developed in C++
- Accuracy to within one degree of visual angle based on how well the calibration was done.
- Only requires one camera per eye.
- The algorithm doesn’t seem to have a lot of processing intensive steps so performance is better than pure model based algorithm.

Cons

- Will require calibration every time based on a number of points in the scene.
- Not as accurate if the glasses move at all, calibration may need to be done again.

Japan Head-Mounted Display

This solution was developed for severely disabled people so that they can operate things with only eye movement. The first (and documented) solution has a built in display, while the second iteration has no display, with the user looking at a computer screen.

Pros

- Uses a compensation algorithm to help improve accuracy of the algorithm
- Calibration is a two step process

Cons

- No documented implementation to look at for help
- Needs at least two mirrors and a very complicated setup
- No error of accuracy given.
- Has display on system which we don't need.

ITU Gaze Group Algorithm

The ITU GazeGroup out of ITU in Copenhagen created an algorithm to do eye tracking. They used a system consisting of cameras and a computer. They developed a lot more for gaze based interactions.

Pros

- Already has been used in a system before.
- Open sourced and already developed.

Cons

- Developed in *C#* and requires *.Net* framework, essentially very difficult to do on an embedded platform.
- Publications are unavailable to see specifics about the algorithm.
- Calibration requires the subject to hold head very still.

For our project, we will be using the *Calibration Free Algorithm*. We are choosing this because of the accuracy and limited calibration. We are concerned that the matrix operations will require a lot of cpu time but we are working with hardware that should defer this problem. The limited calibration gives us more flexibility and accuracy in case the glasses move even a little bit. Hopefully, if we have time, we will implement the *Starburst Algorithm* in place of our ellipse-finding algorithm if the current ellipse algorithm is slower. Since the *Starburst Algorithm* is 2D to 2D mapping, we can rewrite it to give us the 2D ellipse it finds and then feed that into the *Calibration Free Algorithm* to provide 2D to 3D tracking with better accuracy.

Appendix D

User Manual

The Two Pipelines

1. GUMSTIX - PANDABOARD - SERVER - CLIENT COMPUTER

   This is the mobile pipeline that is defined in the project. The advantage to this design is that it is mobile and you can attach a battery to it. This was the intended pipeline for the project. Its downsides, however, are there that the user cannot see the eye-cameras as he/she is putting on the glasses, so it is hard to know if the eyes are visible or not. This pipeline uses ducati h.264 encoding, which has specialized hardware and can be very fast.

2. “PANDASTIX” = LAPTOP - SERVER - CLIENT COMPUTER

   This is the pipeline that makes testing and development a lot easier. The user can use this project to combine both the Gumstix’s and the Pandaboard’s processes on to the same computer. With this pipeline, the user can easily adjust the glasses easily by looking at the eye-camera feeds and making sure the eye is completely visible (more information on this in the *CALIBRATION* section). This requires a computer with at least 2 different USB buses, because the eye-cameras in conjunction with the outward-facing camera require too much USB bandwidth and must be each on different USB buses. This also requires a pretty high-performance computer because it uses x.264 encoding, which is processor-based and may slow down the computer.

Setup

1. Start the server up and begin the server process.
2. Connect the two eye cameras to a USB hub, and then into the Gumstix. Plug the outward facing camera into the Pandaboard. (For PandaStix, plug the USB hub with the eye-cameras into one of the USB buses on your computer, and the outwarding facing camera into another. Then skip to step 8).
3. Give power to both the Pandaboard and the Gumstix (via a battery, or plug in the power adapters to a wall outlet).
4. Get the Pandaboard hooked up to a network via WiFi.
   a. Refer to *PARAMETERS* for more information.
5. Connect the Pandaboard and Gumstix together via an Ethernet cable.
6. The Gumstix process can be started via a serial connection. This will eventually be changed to start the process upon boot.
7. Once the Gumstix process is started, the Pandaboard process may now be started. The easiest way to access the Pandaboard is via an SSH connection.
8. Open up the client GUI on any computer(s), enter in the server IP, and click Connect.
   a. A video screen will pop up.
   b. refer to *CALIBRATION* for more information.

CONFIGURATION PARAMETERS

There are a few parameters that might be needed to changed for this to work (these can be changed on each system individually in the *settings.configurations.eyeris* file, and some of these can be edited in the *Defaults.h* file that defines the default values).

1. IP addresses

a. The Pandaboard-Gumstix communication should not need to be modified but can be if preferred.

b. The Pandaboard-Server communication will need adjustments. The Pandaboard IP is currently registered at Iowa State as 'eyeris-panda.student.iastate.edu' but that is currently registered to a specific student's Net-ID and will need to change when this student leaves. The server's IP will need to be the IP number (or hostname, for simplicity) of what the server is running on.

2. Ports

. These will probably not need to change unless it is preferred.

3. The majority of the other parameters are for the eye-tracking algorithm and can be adjusted as needed according to their function.

CALIBRATION

1. Expected order of the cameras plugged in is the following:

a. Left eye camera must be plugged in first (/dev/video0)
b. Right eye camera must be plugged in second (/dev/video1)
c. World cam is (/dev/video0) on the Pandaboard.
d. PandaStix ONLY: world cam is expected to be plugged in third (/dev/video2)

i. WARNING: your computer may have a built-in camera that will be defaulted to /dev/video0. If so, you will have to modify the cameras expected by the PandaStix code.

2. Put on glasses and make sure that the user's eyes are easily viewable by the cameras. If running on the Gumstix-Pandaboard pipeline (versus just a computer), you will have to look at the ellipses in the top-left corner of the client's video screen to determine if the algorithm is able to see the pupils very well or not. If you are using the PandaStix pipeline, you can enable the PANDASTIX_SHOW_EYECAMS define at the bottom of *Defaults.h*

TROUBLESHOOTING

It's not working! Well, let's figure out why not...
Common things that you may have forgotten:

- Plug the cameras in in the correct order (see *CALIBRATION*). Make sure that if you are using the PandaStix pipeline, you are plugging in the eye-cameras and the outward-facing camera into different USB buses.
- Are the cameras able to get a good image of the person's eyes? Is it flooded with outside light?

- Make sure that all of the IPs and ports match up. Note that all of the network communications are:
  - Gumstix - Pandaboard (this is statically defined and should never be conflicted)
  - Pandaboard - Server
  - Server - Client

Well... it's working, but it's going really slow! What's going on?
First, understand that the full pipeline usually has a ~3 second delay, and the PandaStix project can have anywhere between a 1 and a 10 second delay (depending on hardware). However, the video IS supposed to be quite smooth from both the full pipeline and the PandaStix branch. If there is every a full "graying-out" of the video

- Is the server fast and good at graphics processing? Remember that the server needs to decode h.264, use OpenCV to analyze the frames, paint the eye data, and then re-encode h.264.
- If you are using the PandaStix pipeline, is your computer fast enough and good at graphics processing? This doesn't do as much graphic processing as the server so doesn't require quite as much power, but you still need some.
- Is the network that you are connected to fast and free of congestion? This requires a lot of traffic use - there is a lot of information (both information, data points, and control messages) being sent over the network very quickly, so you must ensure that the network you are connected to can handle all of this traffic. If you are having problems during the day, you might want to try later at night when fewer people are connected to it.

To debug through the actual code, we recommend using an IDE debugger, like Netbeans. Watching what the server is printing is very useful for understanding what is making it to the server and what is being forwarded on correctly.

About
Project Eyeris is team May 13-20 for the 2012-2013 school year at Iowa State University. It is comprised of six members three computer engineers, two software engineers and one electrical engineer. The system cost $538.20. It is designed with open source software and made in one year by this team. If you want any more information please visit our website: http://seniord.ece.iastate.edu/may1320/.