Final Report

To: Mark Brehob
Eric Kumpf

From: Garrison Bellack (gbellack), Computer Engineer
John Connolly (johnconn), Computer Engineer
Loren Wang (wloren), Computer Engineer
Tzu-Fei Yu (tzufeiy), Electrical Engineer
Zheng Hao Tan (tanzhao), Computer Engineer

Date: December 14, 2015

I. Introduction and overview of project

Foreword

Since the beginning of human interaction, names have played a crucial role. Names serve as an identifier for a person, and without them people have no identity. In our modern society dominated by social media, interactions have become more frequent but less substantial, which has sparked a phenomenon where it has become harder to remember people's names. One of the most important things in the social world is remembering people's names. To quote Dale Carnegie, "A person's name is to him or her the sweetest and most important sound in any language". However, too often we can't remember the names of people we meet, even after multiple interactions.

We decided to address this issue by applying it alongside the recent surge in popularity of smart, wearable tech. We created OmniView, a pair of smart glasses with facial recognition capabilities, that can serve as an aid to the user in identifying a person and their name. The purpose of this report is to provide a detailed description of our project, as well as detail our experiences and what we have learned. This report will be written with concurrent engineers and managers in mind to ensure the results of our project are accessible to a larger audience.

Summary

At the College of Engineering Design Expo, we proudly presented OmniView with its two modes, primary and secondary. Primary mode is the normal mode of operation which displays people's names. Secondary mode adds new people to the database by recording their introduction then storing their parsed name along with their picture. However, due to how loud it was in the Expo, we could only demonstrate primary mode. We would give OmniView to people who stopped by our booth and as soon as they put it on, they could tell each of the group members' names by looking at us. OmniView is innately demo-able and people were very impressed by the project.

We used open source facial recognition software to identify people. The project, OpenBR, is available for download[1]. To parse people's names we first used Google's speech to text api[2] to

[1] http://openbiometrics.org/
[2] https://www.google.com/intl/en/chrome/demos/speech.html

convert the audio into words. From there we used NLTK[3] (Natural Language ToolKit) to detect if there were any names in the words spoken.

This report describes the resources used and tasks completed for this project. Below are our goals for OmniView. All Milestones were met.

Design Criteria	Importance	Likelihood	Finished State
Facial recognition correctly determines new/old faces > 80%	Fundamental	Will	Yes. > 90% facial recognition
OLED displays name and info quickly enough such that user will not have to wait	Fundamental	Expect	Yes. Recognition in < .20 seconds
Battery life > 1 hour	Important	Will	Yes. Battery life > 5 hours
User has simple/efficient control of the glasses	Important	Will	Yes. Single push button control
Weight should not exceed such a limit that it will cause pain or discomfort to the user	Important	Expect	Mostly. Initial prototype had unbalanced weight
Software & Microphone are able to capture and person's name	Optional	Stretch	Mostly. Next iteration will have a more sensitive microphone.
Flexible PCB	Optional	Stretch	No. Decided not to use a flexible PCB.

[3] http://www.nltk.org/

II. Description of project

Our goal was to create a device that aids people in remembering names. We believe this product could see widespread casual use, but would be especially useful for people with a bad memory or those entering a large/new social environment such as school, work or a party. We believe it will be used in these situations because the price of not remembering someone's name can be quite high. Especially in business, where having positive work relationships is critical, a lapse in social graces could mean a very real financial setback.

We thought that the optimal way to accomplish this goal is through a device like Google Glass. A small camera can be used to take pictures of whomever the device wearer is looking at, and a display near the eye can be used to alert the wearer of the person's name. From this idea, OmniView was born.

OmniView is a pair of glasses with a camera that can instantly display the name of the person you are looking at. Using an existing database of face-name pairs such as a school registry, the glasses could take a picture of the person you are looking at and send the data to a server where the person's unique facial structure is analyzed. The server would then return the name of the matching entry, showing the name of the person in front of you on a display in the corner of your vision. A new entry could also be dynamically created by taking a picture and sending that information to the server along with accompanying audio of their introduction or only sending the picture and adding their name to the server later. This sort of solution could be very helpful in new social settings, where it can be difficult to remember the names of many new people, or in scenarios where an old acquaintance's name is suddenly forgotten.

With our monetary and time budgets, we have accepted that it is impossible to make OmniView subtle enough to be unnoticeable. In the end, we had great success on the technical side of our project, and succeeded in implementing all of the functionality we sought to. Even with this success, our work is not done. With wearable devices, there are important concerns beyond the technical details. It is not yet socially acceptable to walk around wearing a piece of tech on your face. While we thought that OmniView could be immensely valuable if executed properly, we are also worried about the complexity of the project. The idea is in a lot of ways, a minimalist Google Glass, and so we were worried that we wouldn't be able to complete all of our goals in the timeframe allotted. Even Google has run into issues with their wearable product. The bulkiness of Glass is an issue that prevents widespread use of the device. It is far too obvious that someone is wearing Glass, and this bothers people. Those some complaints can be applied to OmniView – our device is too heavy and intrusive to be comfortably worn. All of this led us to

determine that OmniView as a technical project is feasible, but OmniView as a commercial product is not feasible yet.

ii. Describe the system architecture, including a detailed block diagram (not drawn by hand) of the system including peripherals, processors and the like. Give an explanation of how the architecture works – what the function of each of the devices is and how you communicate to them.

OmniView is comprised of a few submodules: A camera, a microcontroller, a display, a microphone, and a web server. The interactions between these five components powers OmniView, and can be seen in Figure 1. At a high level, the application runs by taking a picture, sending it to the web server, waiting to receive a response from the web server, and writing that response onto the display.

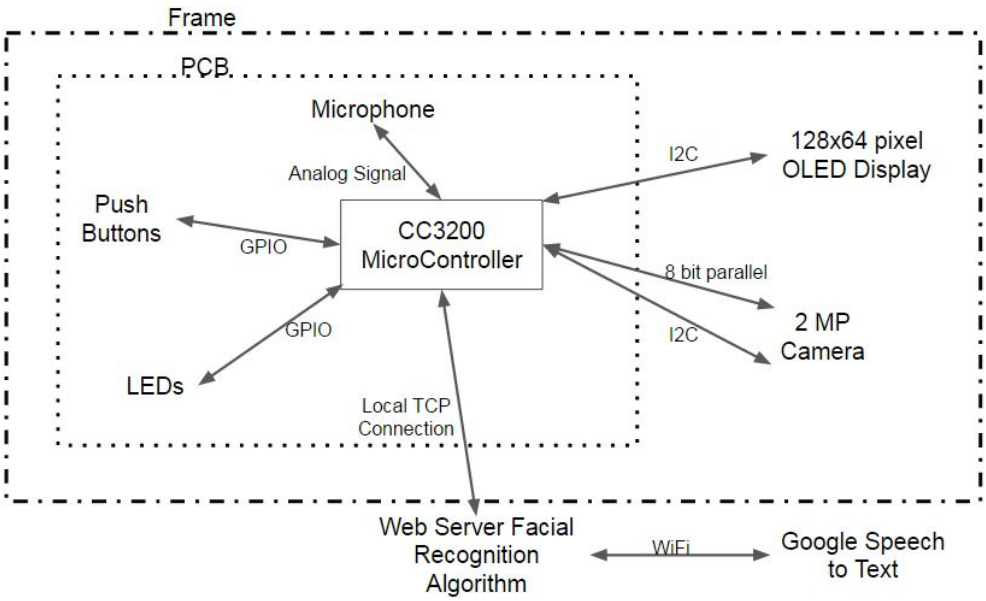


Figure 1: Project Schematic Layout

The first module that will be discussed is the camera. The camera is the lynchpin of OmniView; it provides the images that OmniView compares to the face-name database, so that the application can determine the names of people the device wearer is interacting with. We chose to use the MT9D111 camera for this application. This camera has 2 Megapixel image quality, and communicates with the microcontroller through I2C and an 8-bit parallel data-bus. The I2C line is used to configure the camera, while the 8-bit parallel bus is used to transfer the

actual image from camera to microcontroller. The microcontroller polls the camera once every two seconds, receiving a fresh image and then sending it to the web server at these times.

The web server is used to perform the facial recognition algorithm and query the database of face-name pairs. Using an external server and database allows us to reduce the amount of computations needed to be performed by OmniView. This plays an important role in extending the battery life of the application. Hosting the facial recognition algorithm on the server also allowed us to run the application loop very quickly. This is because the processor of the server is much better than that of our device. We relied on an open-source software for our facial recognition, with good results. Facial recognition using openBR[1], an open-source image processing library, was about 85% accurate. openBR uses the eigenfaces algorithm, a derivative of PCA (principle component analysis) method of machine learning to recognize the faces. The server code starts by initializing all the pictures in the existing database in a process known as "enrollment". This extracts the main features of every picture. Doing this enrollment process before communicating with the device allows us save an extreme amount of time during our main application loop, allowing us to run at an O(1) speed. This is to say that the number of pictures in the database increases the initialization time of the algorithm, but has very little effect on the main loop. Once this process is done, the server connects to the TCP socket initialized by the embedded device. With a wireless connection established, the server can now receive a picture. The picture is saved onto the server, and also enrolled by the algorithm. This allows it to be compared with the existing pictures in the database, and a similarity score is assigned to each pair. The name of the face that was closest to that of the new picture is then sent through the TCP connection back to the device.

The communication between the server and microcontroller is conducted using WiFi. This is done for two reasons: it expands the maximum communication distance between the server and the device, and it allows for a higher throughput of data. Images are fairly data intensive, and we want the latency between an image sent to the server and the corresponding text sent back to the microcontroller to be as short as possible. Using WiFi does introduce at least one added layer of complexity to the device: the registering of the device as a station in a network. Instead of creating a dedicated user interface on the glasses for setting up connectivity, we instead allow for OmniView to be connected to as an access point, and communicate with a user's personal computer. Through this method, OmniView can be registered on the network through the use of a personal computer.

We chose to use TI's CC3200 microcontroller as the on-board processor for OmniView. This microcontroller has a dedicated core that implements the WiFi WLAN and TCP/IP stacks. It also has an on-chip 80 MHz Cortex M4 processor to perform other application logic and interface with peripherals. The microcontroller includes 2 UART modules, 1 SPI module, 1 I2C module, and an 8-bit parallel camera interface. We decided on the TI CC3200 because it has built in support for a 2 megapixel camera and a core that handles the WiFi protocol. OmniView is very space constrained, because it is worn on the face. Having a chip that can handle WiFi and natively communicate with a camera is the best solution for a space-constrained system, because it allows us to avoid introducing even more chips into the system to handle WiFi and camera communication. Another important reason for using a chip that includes everything we needed is that the vendor (Texas Instruments) provided sample code for implementing WiFi on the microcontroller. We relied heavily on this sample code during the course of the project in order to reduce development time. If we used a dedicated WiFi module, it is unlikely that we would be able to find sample code ported to our specific microcontroller.

We decided to use an Organic Light Emitting Diode (OLED) display as an interface to the user. This is the medium through which OmniView shows the device wearer people's names. OLED displays require less power than other common display solutions. OLED displays only draw power based on the sections of the display that are lit. Another important factor is weight of the display; OLED displays are lighter than LCDs. Communication between the display and the microcontroller is done via a common serial protocol. OLED displays in our size range support I2C, SPI, or both. Our system requirements make it impossible to use SPI as means of communication, simply because we do not have enough pins. For this reason, we use the same I2C port that the camera uses to communicate to the display.

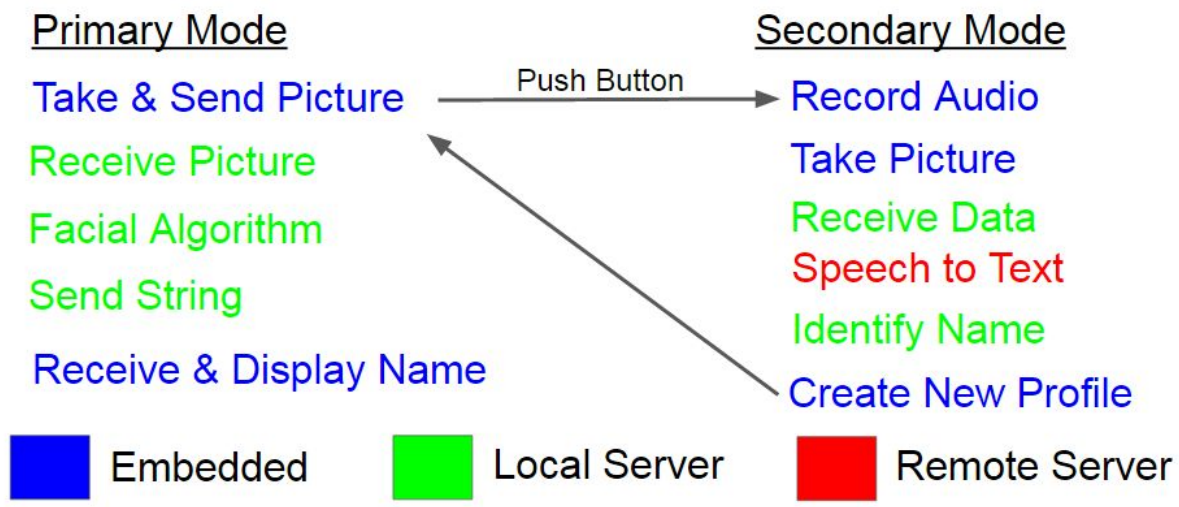


Figure 2: Modes of Operation

The microphone and push button are the parts of the system used to implement the secondary mode of operation for OmniView. The secondary mode of operation is used to dynamically add a face-name pair to the database. The push button is used by the user to switch OmniView from primary to secondary operation. Once secondary mode is activated, the device then grabs data from the microphone for 10 seconds, and then captures an image of who the device wearer is looking at. OmniView then transmits that data to the web server. The web server then parses that microphone data and determines which name was spoken during that period and associates it with the image that was sent with the microphone data. Parsing text from speech is a non-trivial task, and for this reason, we employed Google Speech[3] to convert the data from audio to text. After this was completed, the text was piped to a natural language toolkit (NLTK) that was hosted on the external server. This allowed us to pick out any proper nouns that existed in the sentence. These nouns were assumed to be the name of the person in the picture, and a folder was created for this new entry.

III. Milestones, schedule and budget

We stuck to our original schedule pretty well through the project development cycle. There were multiple tasks that were shifted around the timeline, such as ordering components, which we wanted to order by late September but did not do so until mid November. This was rather trivial as we didn't know what we need on our PCB to get the components to work. We also started working on the microphone about a week in advance, since we were fairly certain that a stretch goal is plausible based on our progress before Thanksgiving break. We also added a few more tasks, such as spinning out a separate sandwich board just in case we needed it. We provided more details about this below.

These were the milestone deliverables that we signed up for during the project proposal stage:

Milestone 1:

Tasks	Outcomes
Prototype is capable of displaying meaningful text on the OLED display.	On time.
Basic WiFi module is able to send/receive data from server.	On time.
PCB schematic design should be done.	We finished the PCB schematic design about 24 hours earlier, but the peripheral board wasn't done until 2 days later because we didn't expect it to take so much time. We were kind of ambitious of getting this done by the time allocated.
The image recognition algorithm should already be implemented and able to show meaningful information.	This was mostly done by the time the project proposal was due.

Milestone 2:

Tasks	Outcomes
PCB should have been ordered by now.	We were about 24 hours late for this because we have having some doubts about our main board layout. Fortunately, we pushed this back and managed to fix them before we ordered them.
The OLED display and WiFi software should be finished with only minor bugs left.	On time.
Have decided on whether we want to implement functionalities related to the microphone.	On time.
Frame of glasses should be 3D printed by now.	We decided to put this task till later because during this stage of the project, we were running into issues with our CC3200MOD package. We felt that we needed the extra manpower (Tzu-Fei) to come over and help us desolder/resolder until we can successfully program the board. We eventually finished it after 4 days from the proposed deadline.

IV. Lessons learned

What went well? What went poorly? If you could send a short memo back in time to your group when you first started, what would it say? What technical material do each of you feel you've learned from the project?

Over the course of the project, we have learned a lot about what it takes to create a complex embedded system. There is a design phase – where components and tools are chosen for the system. Next there is a development phase – where the idea begins to be actualized. Then comes the assembly phase – where each module is integrated into the finalized system. We learned a lot about the design and assembly phases of embedded system on the way to our finished project.

The most important lesson learned is that the design phase is the most important part of system implementation. There is a very real possibility that an idea will have to be completely reworked in the prototype phase when the engineer realizes that the design isn't feasible. Every stage of the project needs to be properly thought out. If there is even a single mistake in the design, then at best it will be a few extra hours creating a work-around for the system to function. At worst, it could mean the creation of an entirely new system.

One prime example of a mistake in the design stage leading to enormous problems down the road is an issue we had with our printed circuit board (PCB). During the schematic design of the PCB, our engineers chose to go with a package that would save us space and simplify the design process significantly. This probably resulted in 8 man-hours saved in the design of the schematic and layout of the PCB. However, that part is incredibly difficult to solder. The result was an added 40 man-hours spent trying to solder this component to the PCB.

A few things went very well. Using components supported by our vendor, Texas Instruments, sped up some areas of development significantly. An important reason for choosing the camera that we did is because TI has sample code for interacting with this device. Cameras are relatively complex, and choosing to use a camera that had the interfacing done for us was a huge boon. Having everything incorporated into our microcontroller was a major factor in our completing the software development on time. This allowed us to spend more time working through the unexpected issues that arose in the project, such as the PCB soldering issue described above. Another such unexpected issue was the difficulty we had working with TI's code. While there was plenty of sample code provided for us, it was extremely difficult to follow, and some of the

sample applications TI are not well documented. For instance, one sample application instructs users to use "the browser of her choice" to connect to the microcontroller. We have found that Chrome, Firefox, and Internet Explorer are not compatible with this sample application, and when we tried to use these browsers the sample application failed. It was only through many hours of trial and error that we determined Safari to be the only browser that was compatible with the microcontroller and sample application.

We have gained a wide variety of skills from software to hardware development through the creation of OmniView. Several of our group members are now proficient in the eagle CAD tool. Beyond that, these group members have a much more solid theoretical understanding of the best practices involved with circuit board layout. We also needed to lay out a 50 ohm impedance trace for OmniView's antenna, which was a valuable experience and gave us some insight on the different factors that affect trace impedance at very high frequencies. Another hardware related skill improved upon in this project is soldering of surface mount components. We had two PCBs to populate, with a wide variety of packages, some of them extremely difficult to solder. The positive to this struggle is that several of our team members are now extremely proficient in soldering a wide variety of surface mount components. Our group has gained experience in using large open source tools through this project as well. We used several open source tools, including openBR[1], for image analysis, and Google Speech[3] for audio processing. This project has taught us how to integrate these tools into an existing application.

The most important thing that we learned in this project is system design and architecture. In our careers as students at the university, we have never had the opportunity to design a system from top to bottom like we have in this course. The project idea and implementation are entirely up to us. Designing a system with both hardware and software components from scratch is a very difficult thing to do, because of the numerous unforeseen difficulties that arise in system design. One such difficulty is circuit board fabrication. Designing a circuit board with a CAD tool is easy enough, but fabricating the board is another matter. Usually circuit board fabrication will cost at least $100, and it takes at least a week to receive the board after an order. From a software perspective, the system becomes very complex, and larger than the software projects students are accustomed to in their classes. This project gave us experience in managing such a large system.

If we could send a memo back in time to our group, the memo would detail some architecture changes that should be made. Our choice of processor is currently in a quad flat no-lead package (QFN). We made this decision because it simplified the schematic, but it raised the difficulty of our soldering job by an order of magnitude. This letter would tell our group to spend the extra

few hours using a different part in the schematic, so that we would not have to spend 50 hours soldering. In addition, the letter would detail spending a bit more time on the architecture of the system. The chip that we used, TI's CC3200, ended up being very difficult to work with. In hindsight, it might have been worth choosing a chip from a different vendor, because of the huge amount of difficulty that we had dealing with Texas Instruments.

V. Contributions of each member of team

Team Member	Contributions	Effort
Garrison Bellack	In the early phase of the project, Garrison helped Loren in bringing up the facial recognition software on the server side. He implemented some of the TCP networking code. He then did most of the integration tasks as different components were added to the project. In other words, the interfaces were written for him, then he integrated them and debugged it. Most of the software refinements (timing constraints and requirements) during the later stage were handled by him and Loren.	20%
John Connolly	John and Tzu-Fei did the schematic design and board layout. John also helped out with surface-mount/soldering related work. After we got the board to be programmed, he helped develop the microphone interface. He also helped the team to debug camera issues (both large and small cameras).	20%
Loren Wang	Loren was the main guy who did server side code for both facial recognition software and natural language processing via Google Speech. He also helped the team to debug camera issues (both large and small cameras). Most of the software refinements (timing constraints and requirements) during the later stage were handled by him and Garrison.	20%
Tzu-Fei Yu	Tzu-Fei and John did the schematic design and board layout. Besides that, he joined Zheng Hao to order the components needed. He also helped us with surface-mount/soldering work and made the 3D-printed glasses from scratch.	20%

Zheng Hao Tan	Zheng Hao helped in the initial board bring up and setting up the toolchain (IDE + flashing software). He then worked on the display interface and helped Tzu-Fei on part selection/ordering them. He then helped out with soldering work. Once we got the board programmed, he worked on both a timer interrupt and button (GPIO) interrupt interface. The timer interrupt interface is then used by John for microphone purposes.	20%

VI. Cost of Manufacture

The quote had failed to estimate the price for the main system board at first. After a few trials, we have determined that the minimum trace was too small (0.01mm). By carefully scanning through the board layout and fixing the width of the trace and polygon, we successfully generated a quote. According to the table below, which shows the result of the quote, the total cost for populating the whole system is about $728 if we requested only one unit. In contrast, this is below the $1000 budget that was given to us for the project. If we had limited the amount of money spent on the prototyping phase, we could afford this cost. Unless we would like to have spare ones in case the board is accidently burnt, one of each board should satisfy our application. The limiting quantity is four, where the total price for both board is $1000.75, and this will exceed our budget, leaving no room for us to prototype it.

On the other hand, one of the downsides of this is that once a mistake is unintentionally made, there will be no room to make another iteration for the PCB. Additionally, the manufacturing time is a major factor since the given project developing time span is about three months. The 22 day processing and shipping time will take up almost ⅓ of our project schedule. Furthermore, the price of the board increases dramatically if we want a faster turnaround time. The minimum manufacturing period is 8 days, and the cost for both boards is $1972.82. This price is not feasible. Most of the expenses goes to the assembly category when processing time is long, and goes to both PCB manufacture and assembly when the time is short. As a result, with the given time and budget, it is more desirable not to go with this option. Even if time allows, one would save more money without having the board house to assemble the board but doing oneself. In general, this option is not reasonable.

OmniView Main Board						
Days	Quantity	PCB ($)	Parts ($)	Assembly ($)	Total ($)	Unit Price ($)
22	1	57.81	77.43	290.34	425.57	425.57
22	2	57.81	101.13	308.86	467.79	233.9
22	4	110.61	148.42	345.89	604.93	151.23
8	1	696.76	77.43	435.51	1209.7	1209.7

OmniView Peripheral Board						
Days	Quantity	PCB ($)	Parts ($)	Assembly ($)	Total ($)	Unit Price ($)
17	1	15.32	65.65	222.28	303.24	303.24
17	2	15.32	81.29	234.71	331.31	165.65
17	4	25.63	110.61	259.00	395.82	98.955
7	1	364.05	65.65	333.42	763.12	763.12

VII. Parts

Budget

The items and price are listed in the table below:

Item for prototyping	Quantity	Total Cost ($)
OLED display Breakout board	4	78.00
TI CC3200 LAUNCHXL Development board	2	66.98
TI CC3200 MOD Development board	1	34.99
TI CC3200 Camera Booster Pack	2	44.49
MT9D111 Camera Breakout	2	33.2
Microphone Breakout	1	7.95
Item used on final product	**Quantity**	**Total Cost ($)**
LDO regulators (3.3V, 2.8V, 1.8V) TPS79328DBVRG4 TPS79318DBVR TPS73733DCQ	20	Free
MT9D111 Camera (flex module)	5	24.66
AVX 145602030000829	10	8.80
OLED Module	4	25.23
PCB Main Board	2	152.03
PCB Peripheral Board	2	86.03
PCB Sandwich Board	3	4.05
TI CC3200 MOD MCU	5	Free

TI CC3200R1M2RGC	3	31.37
Glasses Frame	2	Free
Lens	1	10.99
Passive components (Capacitors, Resistors, Inductor, LEDs, Cable, Connectors, ferrite beads, uFL connector, button switch, pin head, ribbon cable, wires, Amplifier MAX9814, Microphone electret CMA-4544PF-W)	-	90.00
Lipo Battery Charger	3	23.85
Lipo Battery 850mAh	2	19.90
Lipo Battery 500mAh	1	7.95

The total expenditure for this project is $751.47.

VIII. References and citations

Our main repository is split into two sections, one of them is the firmware for the facial recognition glasses and the other is server side code. Here is a brief overview of how we laid out the main repository, as well as the files and interfaces for the components that we use.

omniview/main.c :

This file contains the main function and several initialization functions for the facial recognition glasses.

```
#define FACE_RECOGNITION_TASK_NAME "FaceRecognitionTask"
#define FACE_RECOGNITION_TASK_STACK_SIZE 8192
#define FACE_RECOGNITION_TASK_PRIORITY 2

// EFFECTS: Initializes the board.
static void InitializeBoard();
```

omniview/mode.c:

This file contains the two modes that the facial recognition is capable of doing, as well as module initialization functions. All modules (display, camera etc.) are included and used here. Primary mode basically takes an image every second and returns the name of the person in the image.

```
// EFFECTS: A freeRTOS task that runs the facial recognition mode. This periodic task is
scheduled to run every second.
void FaceRecognitionMode(void *pvParameters);
```

omniview/pinmux.c:

This file is originally provided by Texas Instruments in their sample applications, and it contains the pin configuration and modes for the facial recognition glasses. We have modified it extensively to fit our pinout requirements.

```
// EFFECTS: Configures the pinout for this project.
```

void PinMuxConfig(void);

omniview/modules/display/display.c + omniview/modules/display/display.h:

These files contains the OLED display interface.

```
#define SSD1306_I2C_ADDRESS   0x3C    // 011 110+SA0+RW, SA0 is grounded.

/* PIN LAYOUT CONFIGURATIONS */

#define DISPLAY_RESET_PIN       0x10 /*gpio 28 for display reset */

/* OLED CONFIGURATIONS */

#define SSD1306_LCDWIDTH              128     /* Display width */
#define SSD1306_LCDHEIGHT             64      /* Display height */

#define SSD1306_SETCONTRAST                         0x8100
#define SSD1306_DISPLAYALLON_RESUME                 0xA400
#define SSD1306_DISPLAYALLON                        0xA500
#define SSD1306_NORMALDISPLAY                       0xA600
#define SSD1306_INVERTDISPLAY                       0xA700
#define SSD1306_DISPLAYOFF                          0xAE00
#define SSD1306_DISPLAYON                           0xAF00

#define SSD1306_SETDISPLAYOFFSET                    0xD300
#define SSD1306_SETCOMPINS                          0xDA00

#define SSD1306_SETVCOMDETECT                       0xDB00

#define SSD1306_SETDISPLAYCLOCKDIV                  0xD500
#define SSD1306_SETPRECHARGE                        0xD900

#define SSD1306_SETMULTIPLEX                        0xA800

#define SSD1306_SETLOWCOLUMN                        0x0000
```

```
#define SSD1306_SETHIGHCOLUMN                   0x1000

#define SSD1306_SETSTARTLINE                        0x4000

#define SSD1306_MEMORYMODE                      0x2000
#define SSD1306_COLUMNADDR                      0x2100
#define SSD1306_PAGEADDR                        0x2200

#define SSD1306_COMSCANINC                      0xC000
#define SSD1306_COMSCANDEC                      0xC800

#define SSD1306_SEGREMAP                        0xA100

#define SSD1306_CHARGEPUMP                      0x8D00

#define SSD1306_EXTERNALVCC                     0x0100
#define SSD1306_SWITCHCAPVCC                    0x0200

/* SCROLLING DEFINES */
#define SSD1306_ACTIVATE_SCROLL                 0x2F00
#define SSD1306_DEACTIVATE_SCROLL               0x2E00
#define SSD1306_SET_VERTICAL_SCROLL_AREA        0xA300
#define SSD1306_RIGHT_HORIZONTAL_SCROLL         0x2600
#define SSD1306_LEFT_HORIZONTAL_SCROLL          0x2700
#define SSD1306_VERTICAL_AND_RIGHT_HORIZONTAL_SCROLL 0x2900
#define SSD1306_VERTICAL_AND_LEFT_HORIZONTAL_SCROLL  0x2A00

#define BLACK 0
#define WHITE 1
#define INVERSE 2

/* NAME PARAMETERS */
#define MAX_NAMES_ALLOCATED 5
#define MAX_NAME_LENGTH 15
#define NAME_TEXT_SIZE 1
```

```
#define TEXT_COLOR WHITE

/* Swaps the two values */
#define swap(a, b) { int16_t t = a; a = b; b = t; }
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))

/* EFFECTS: Displays the name on the screen. */
extern void DisplayName(const char *firstName, const char *lastName);

/* Printing functions */
extern void ClearPrintDisplayLine(const char* str);
extern void DisplayPrint(const char *str);
extern void DisplayPrintLine(const char *str);
extern void PrintHelper(uint8_t c);

extern void FillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
extern void DrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
extern void DrawFastVLineInternal(int16_t x, int16_t __y, int16_t __h, uint16_t color);
extern void DrawFastHLineInternal(int16_t x, int16_t y, int16_t w, uint16_t color);

extern void DrawChar(int16_t x, int16_t y, unsigned char c, uint16_t color, uint16_t bg, uint8_t
size);
extern void SetCursor(int16_t x, int16_t y);
extern void SetTextColor(uint16_t c);
extern void SetTextSize(uint8_t s);
extern void SetTextWrap(uint8_t w);

/* EFFECTS: Changes to new line */
extern void IncrementLine();

/* REQUIRES: The reset pin.
 * EFFECTS: Initializes the display pins.
 */
extern void InitializeDisplay();

/* EFFECTS: Send and receive the payload */
```

```
extern void Send(uint16_t payload);

/* Display */

/* EFFECTS: Clears the display. */
extern void ClearDisplay();

/* EFFECTS: Displays whatever that is stored in buffer */
extern void Display();

/* EFFECTS: Dims the display */
extern void Dim(int dim);

/* EFFECTS: Turns the display on. */
extern void DisplayOn();

/* EFFECTS:d Turns off the display. */
extern void DisplayOff();

/* Scrolling */

/* EFFECTS: Sets the scroll direction */
extern void StartScrollRight(uint8_t start, uint8_t stop);

extern void StartScrollLeft(uint8_t start, uint8_t stop);

extern void StopScroll();

/* REQUIRES: the x, y, and color.
 * EFFECTS: Sets a pixel.
 */
extern void DrawPixel(int16_t x, int16_t y, uint16_t color);
```

omniview/modules/display/buffer.c:

This file contains the display buffer. The reason why we have this in a separate file is because the display had initialization values that we wanted to keep, and by adding this huge buffer definition in our display function, it will make that file really big and hard to read. Hence, we decided to split it into a separate file and link it up.

static uint8_t buffer[SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH / 8] = ….

omniview/modules/display/font.c:

This file contains the font settings for text on our OLED display. For the same reason above, we decided to move it to a separate file.

/* This file contains the font used in printed text. */

// Standard ASCII 5x7 font
static const unsigned char font[] = ...

omniview/modules/interrupts/button_interrupt.h +
omniview/modules/interrupts/button_interrupt.c:

These files contain interfaces for triggering a GPIO interrupt via one of our buttons. Some of the macros are linked to other macros defined by Texas Instruments.

/* This is the amount of time to count to account for button debouncing. */
#define BUTTON_DEBOUNCE_THRESHOLD 4

/* This determines the button interrupt priority */
#define BUTTON_INTERRUPT_PRIORITY 10

/* Macros for interrupt button pins */
#define INTERRUPT_BUTTON_PIN PIN_01
#define INTERRUPT_BUTTON_GPIO_PIN GPIO_PIN_2
#define INTERRUPT_BUTTON_BASE_ADDR GPIOA1_BASE
#define INTERRUPT_BUTTON_GPIO_HW_INT INT_GPIOA1

/* EFFECTS: Initializes all the interrupts used in this application */

```
extern void InitializeInterrupts();

/* EFFECTS: Disable and unregister all interrupts */
extern void DeinitializeInterrupts();

/* EFFECTS: Toggles the mode based on button press (ISR) */
static void ButtonPressIntHandler(void);
```

omniview/modules/interrupts/timer_interrupt.h + omniview/modules/interrupts/timer_interrupt.c:

These files contain interfaces for triggering a timer interrupt. This is mainly used by the microphone interface for audio recording purposes during secondary mode.

```
#define TIME_IN_MSECS 100
#define SAMPLE_RATE 10000
#define MIC_SAMPLE_SIZE 2

/* EFFECTS: Initializes Timer A and enables it */
extern bool recordingDone;

extern void TimerConfigNStart(uint32_t sampleRequest, char *buf);

extern uint32_t currentSamples;

/* EFFECTS: Deinitializes Timer A and unregisters it */
extern void TimerDeinitStop();

/* EFFECTS: Timer Interrupt handler  */
extern void TimerPeriodicIntHandler(void);
```

omniview/modules/microphone/microphone.h + omniview/modules/microphone/microphone.c:

```
// EFFECTS: This function accepts a pointer to a buffer
```

```
//and the number of seconds of recording is desired
//and fills the buffer with microphone data
//it returns the number of bytes of data stored in the buffer.
//The current system is set up to record 10000 samples per second.
//this corresponds to 20000 bytes per second
// some common values of numSeconds
// milSec = 1000 -> 20000 bytes
uint32_t GetAudio(char * buf, int milSec);

// EFFECTS: Grabs an ADC sample
inline uint16_t GetMicSample();

// EFFECTS: Initialize the ADC associated with the microphone, must be called before
//you do any mic stuff
void InitializeMicrophone();
```

omniview/modules/networking/tcp_network.h + omniview/modules/networking/tcp_network.c:

These files contain the TCP networking interface required to send data/files across the network (or in our case, interact with our server).

```
extern int InitTcpServer(unsigned short port);

extern void SendInt(int sockID, int num);

extern void TakeAndSendPicture(int sockID);

extern void TakeAndSendRecording(int sockID, int milSec);

extern void ReceiveString(int sockID, char* stringBuf, int bufSize);

void Delay(int count);

void WaitForAck(int sockID);
```

void SendData(int sockID, UINT8* fileData, int fileSize);

void SendFile(int sockID, UINT8* fileData, int fileSize);

omniview/modules/camera/mt9d111.h + omniview/modules/camera/mt9d111.c:

These files contain the driver for the camera model that we are currently using.

long CameraSensorInit();

long StartSensorInJpegMode(int width, int height);

long CameraSensorResolution(int width, int height);

omniview/modules/camera/i2cconfig.h + omniview/modules/camera/i2cconfig.c:

These files contain the I2C interface for the camera.

unsigned long I2CInit();

unsigned long I2CBufferRead(unsigned char ucDevAddr, unsigned char *ucBuffer,
unsigned long ulSize,unsigned char ucFlags);

unsigned long I2CBufferWrite(unsigned char ucDevAddr, unsigned char *ucBuffer,
unsigned long ulSize,unsigned char ucFlags);

omniview/modules/camera/camera_app.h + omniview/modules/camera/camera_app.c:

This file contains the high level interface for camera functions.

void InitCameraComponents(int width, int height);

void CamControllerInit();

unsigned short StartCamera(char **WriteBuffer);

long SetCameraResolution(int width, int height);

SERVER SIDE CODE

omniview/modules/face_software/faceRec.cpp
This file contains the main server loop as well as several functions that help it in running.

//Adds new value to buffer and returns the name that holds the majority in the buffer
string majority(string new_val);

//Facial recognition algorithm returns the file path of the picture with the closest similarity to the captured image. This function extracts the name of the person from the filepath. (The image resides in a folder named after the person).
void extractName(double score, QString file_input);

//Runs the main loop for facial recognition. Inits tcp socket and database of pictures. Waits for tcp message, which determines whether to run primary or secondary mode. Primary mode runs the main facial recognition algorithm with the received picture, and sends the detected name back to the server. Secondary mode receives data, and sends audio to wav_transcribe.py. This will return a string with the processed name, and create a new folder with the picture and name.
int main();

omniview/modules/face_software/tcpClient.cpp +
omniview/modules/face_software/tcpClient.h

Houses the server side networking code used to communicate (by TCP) with the embedded device.

//initializes the socket number to the same number as on the device
void initSocket(int& sockfd);

//sends a string to the device
void sendString(int sockfd, string str);

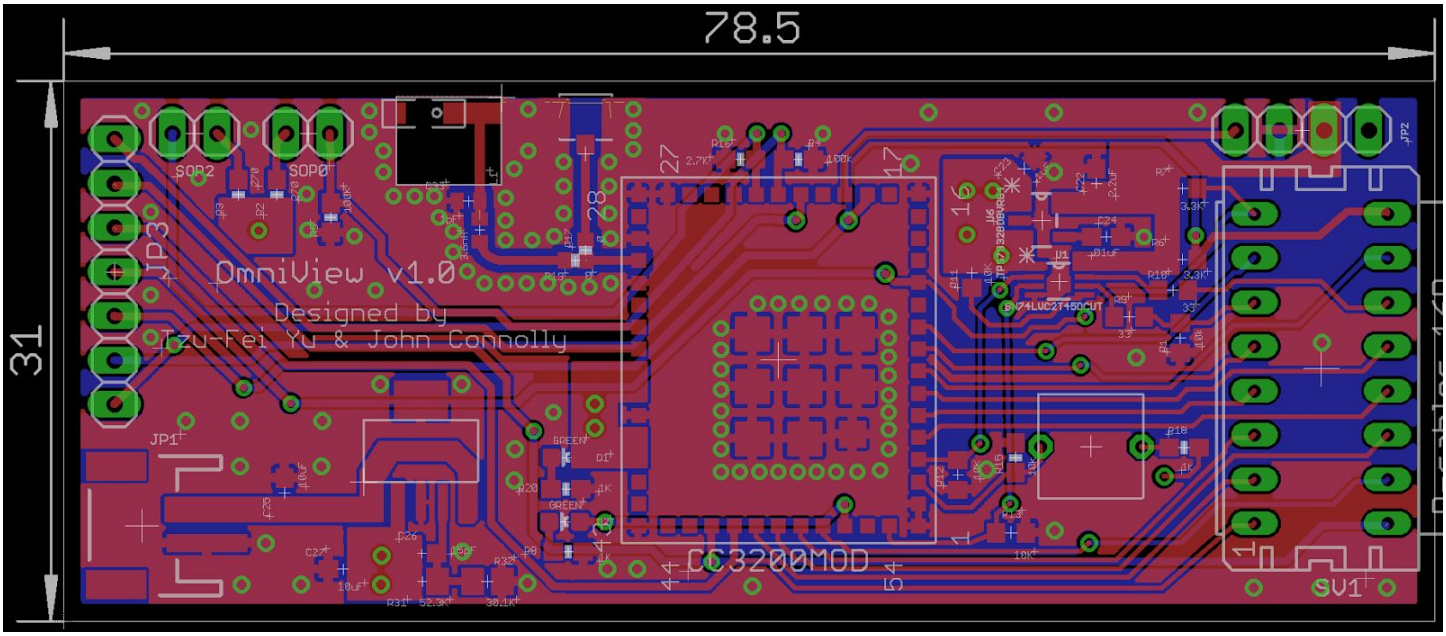
//main receive loop, receives raw data bytes from the device
int recvMain (int sockfd);

omniview/modules/face_software/wav_transcribe.py
Called by the main facial recognition loop. Communicates with google's speech to text recognition server. Sends a string to parseName and receives names back. Sends names back to facial recognition main.

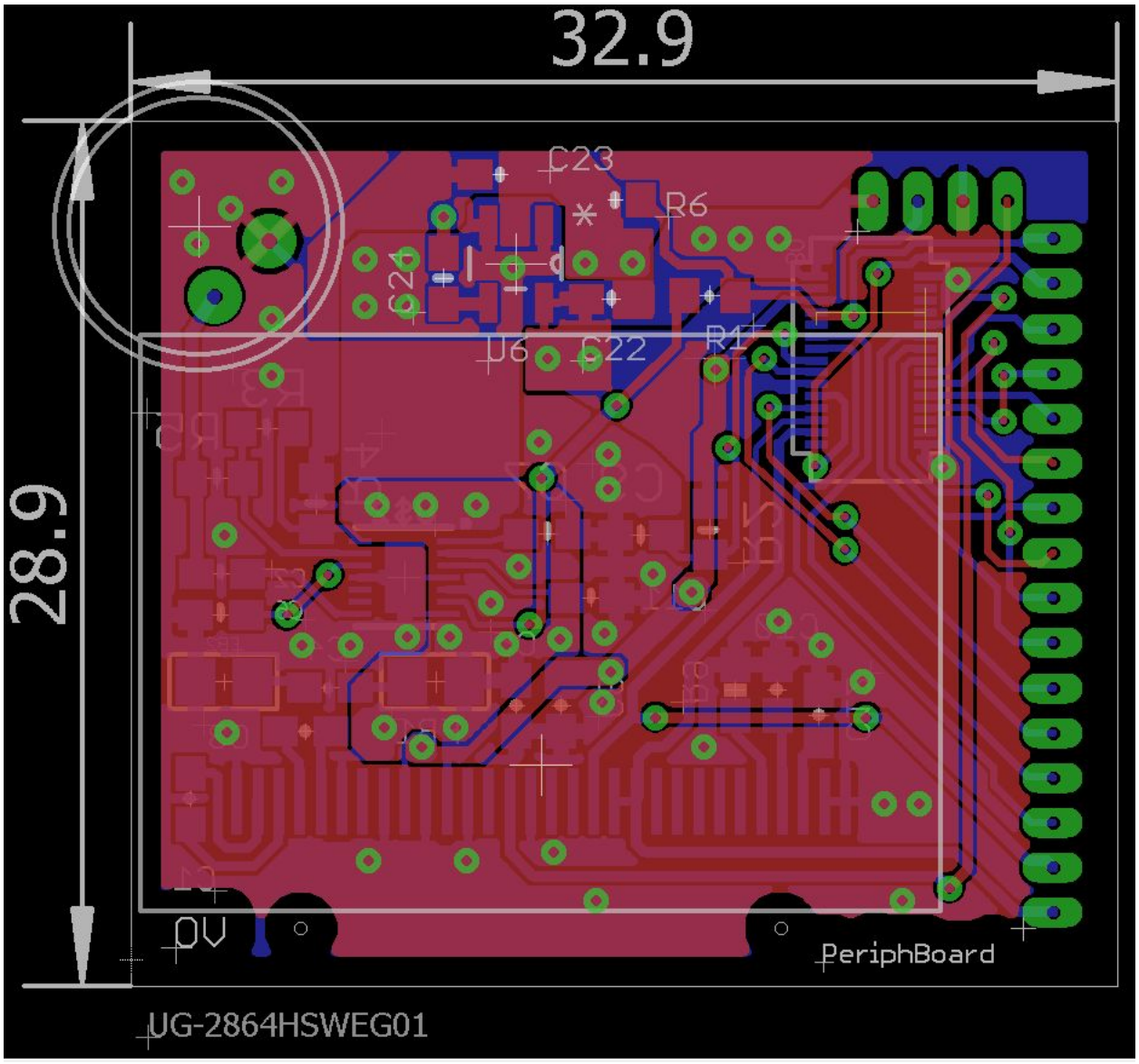
omniview/modules/face_software/parseName.py
Uses a natural language processing toolkit to process the string. Returns the names identified by the algorithm to wav_transcribe.

omniview/modules/face_software/training
Database holding our folders of faces. Test pic gets compared to this database to find the most similar face. A folder is created for each unique person in our dataset. The folder, named after the person, houses the pictures of the person inside.
Eg. omniview/modules/face_software/training/john_connolly/john1.jpg is the first picture of John Connolly in the database.

OmniView Main Board



OmniView Peripheral Board



Display (SSD1306 126x64 OLED Display):

We bought a display breakout board from Adafruit and we were thankful that there is pre written library for Arduino on GitHub[4]. Nevertheless, there was a fair amount of work trying to port it over to our project. For example, some of the Arduino library calls were not given, and since this particular library is a subclass of another Adafruit library that handles graphics on displays, we had to merge them together. I2C functions were designed differently on both platforms, and we had to move data bits and send them in chunks as opposed to Arduino's append style function calls. Hence, there were some logic that was modified to get this to work. Besides that, we did not have a delay function in our platform, and this required us to manually calculate and implement them based on clock cycles.

Camera (MT9D111):

There were limited cameras that were compatible with our CC3200 microprocessor, and since Texas Instruments had sample applications (and some libraries) for the MT9D111 camera, we decided to use it for OmniView. However, we had to decouple dependencies on our camera, as well as fix some of the timing constraints that TI violated in their drivers. For example, they did not follow the hard reset timing guidelines on the datasheet, and that caused us some trouble when we were debugging the smaller camera. On our final PCB, we were using a smaller version of the MT9D111, and it took us a day to realize that the slave address is different than the larger camera (0x90 as opposed to 0xBA). Although they have the same model numbers, our smaller camera had issues when the internal clock is enabled despite us not using it at all. The library had to be modified to fit our needs.

Microphone:

We purchased a breakout board from Adafruit, and there were several sample application on how to use them. Since the interface was fairly trivial, we implemented them from scratch for our own application.

MCU (CC3200MOD):

Since we went for the CC3200MOD, we were lucky enough to find someone who did some of the CC3200MOD pin layout for us[5]. Other than that, most of the layouts were done by hand.

[4] https://github.com/adafruit/Adafruit_SSD1306
[5] https://github.com/alikian/CC3200MOD

3D-printed glasses:

The 3D printed glasses and housing was designed using solidworks and printed by the makerbot in the mesh lab. The housing for both the boards were done from scratch. However, the main frame of the glasses was taken online [6] and modified to fit our needs. Bassically, the glasses was completely redesigned but the base sketch was referenced from online sample. We measured the dimensions and eliminated the unnecessary parts to reduce weight. Knobs and matings are added so that the frame is able to hold our housing and it can be printed separately.

Others:

There were some sample applications that used timer and GPIO interrupts, and we were able to understand and port them over to our application. All the pins and timing constraints had to be changed, but the basic idea of initializing, enabling, clearing and disabling them were taken from some of the examples given. In other words, there were good wrappers/interfaces provided by TI for handling interrupts.

[1] **http://openbiometrics.org**

[2] **http://www.nltk.org**

[3] **https://www.google.com/intl/en/chrome/demos/speech.html**

[4] **http://www.imagemagick.org/script/index.php**

[5] **http://sox.sourceforge.net/**

[6] **https://grabcad.com/library/software/solidworks/tag/glasses**