

HUDware

Sanghoon Lee, EE, David Meschisen, EE, Jori M. Platt, EE, Minwo Wang, EE

Abstract—HUDware is an augmented reality (AR) heads-up-display (HUD) device that connects a skier to their mobile device. HUDware uses wink detection to allow the user to manipulate the device to view data like kinematics or messages within their ski goggles distraction-free. It is an attachment that connects to the skier’s existing goggles to both display data and receive input from the user. Information generation is handled by the mobile phone application which takes in the control signals from the wink detector and outputs text for data values.

I. INTRODUCTION

BEING alone with nature is one of the great things about skiing. The downside to this solitude is isolation. If a friend or family member is trying to reach the skier, the skier will remain blissfully unaware. Even if they were to know that a message has been sent, they would need to then go through the trouble of reading the message in the middle of the run. using simple gestures.

Skiers and snowboarders commonly embrace the cold and blistering weather to tackle difficult slopes and terrain. When they head out, they take their cell phones. This could be for safety, for music, or for the many features and applications that now can add to the skiing experience. Applications for mobile devices can now track a skier’s speed, distance traveled, and location. Unfortunately, all this information is locked away until the skier returns to base. Live statistics are unachievable due to the inaccessibility of mobile devices while skiing. In order to change a song, a skier will have to remove their glove in the freezing temperature, dig through their coat to find their phone, and then attempt to navigate the menu with shivering fingers, before reversing the process and placing the phone back into their coat pocket. If someone is trying to reach them with their phone, it is unlikely that they will even notice. Calls and messages ring uselessly against the vibrations and noise of the slopes. If the skier is lucky enough to notice their phone, they must stop their run, quickly remove their gloves, and find their phones to respond, again exposing their fingers in unpleasantly cold temperatures.

The successful demonstration of HUDware is a stepping stone for AR-based technologies to become more realizable in the future. It aims to set an example that such products can be inexpensively fabricated and still maintain robust performance in tough environments. In a world where smartphones are becoming more prevalent among many people, there is a need for such a device to seamlessly integrate within a person’s daily schedule to keep up with notifications and events. Essentially, HUDware shows that there exists another medium for smartphones to interact with a user in a passive manner. The user will not have to divest much attention to the phone while

still be able to control it using simple gestures.

There are already devices that can send a display image based on information from a mobile device, or preset settings. These devices however are only one directional; information is sent to the user, but the user cannot change the type of information sent. Heavily interactive displays for certain AR devices are cumbersome and difficult to integrate into an active setting like skiing. Similar systems include Google Glass and the Microsoft HoloLens [1]. Both systems fail to be practical for a skiing HUD due to their price and impractical nature. Google Glass uses a touch bar and voice commands while the Microsoft HoloLens requires hand gestures, both which would be difficult to manage with gloves on, let alone while skiing. Vocal commands and hand gestures are inconvenient when the user’s voice is muffled by a scarf and their hands by gloves. In addition to being impractical to use, AR devices are expensive and do not lend themselves to the rigor of athletic activities. To be integrated into the skiing, or any active and quick paced environment, they need to be hands-free, durable, and relatively inexpensive.

At its core, our goal for a HUD for skiing would need to be able to take information from the user quickly and in real time, use that information to control the device and the information, and remain unobtrusive throughout the whole process. The device will need to display relevant and important information, have a reasonable battery life, and be easily integrated into the skier’s current gear. More specifically, this device must not add significantly more weight to the user’s helmet. It also must withstand typical skiing conditions, which has a temperature range of about -20 to 30 °C. The battery must provide enough power to run about 4 hours, which will allow it to be feasible for at least a half days’ worth of skiing. Sending control signals at a minimum of 2 frames per second (FPS) allows for smooth instantaneous updating of information on the goggles. Most importantly, for the sake of the user’s safety, the AR projection must not impede the user’s sight. Additional information regarding requirements for our device is provided in Table 1. By combining these features into a single device, we will be able to enhance the skiing experience as well as advance practical AR technology.

Table 1
GENERAL REQUIREMENTS

Requirement	Specification	Result
Lightweight attachment	< 100g	260g- but the casing does not feel heavy
Operable in reasonable temperatures	Temperature range -20 to 30 °C	
Reasonable battery life	> 4 hours	9.7 hours
Quick feedback	Control signals > 2FPS	3.51 fps
Unobtrusive	Unobstructed view	Met

II. DESIGN

A. Overview

HUDware will bridge the gap between the skier and the phone with a two-part system that displays the information from the mobile device and user interface that allows the skier to manage the phone. The display system begins at the phone application which collects kinematic data, messages alerts, phone calls, and music information. This information is transmitted through an open source PIC microcontroller-based board (IOIO, pronounced “yo-yo”) to a microcontroller (MCU), namely the Arduino Micro, which drives an Organic Light Emitting Diode (OLED) display. This display is reflected off a transparent material to give the illusion of a heads-up-display in the skier’s field of vision. The user can then control the interface by winking either the left or the right eye. This is picked up by a camera located in the center of the device above the field of view of the skier. The other MCU, the Raspberry Pi Zero, then processes the image to determine if the user closed only one eye. The result of this analysis is used to generate control signals, which are then sent back to the mobile device.

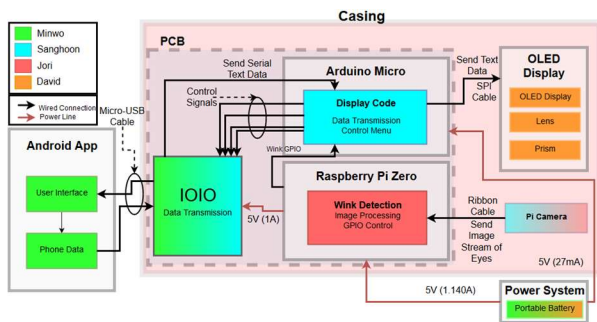


Figure 1. Block diagram of system components

Figure 1 displays the different subsystems that comprise HUDware. Each component is color coded to the engineer designing the subsystem. The Android application takes in control signals and feeds text data to the IOIO device. The IOIO handles communication between the microcontroller and the mobile application via a wired USB connection. An Arduino Micro drives the projector and maintains the menu states while the Raspberry Pi Zero analyzes the images captured by the camera. The wink detector is an image processing algorithm on the Zero that takes the images from the camera, determines if one eye is closed, and sends corresponding control signals to the Micro. The projector display uses an OLED screen and a prism-like structure to create an image of relevant information appear in the skier’s field of view. The power system feeds into the Zero and then to both the Micro and IOIO. By extension of being powered by the MCU, this power system will also handle the demands of both the projector display and the camera. Finally, all the components shown in the casing block, the OLED Display, Pi Camera, Zero, Micro, IOIO, and printed circuit board (PCB), will be held in the 3D printed plastic casing.

B. Wink Detector

The user control system is a wink detector system. This works in conjunction with the camera subsystem to detect if the user is closing one and only one eye as well as which eye is closed. The camera takes a picture of both eyes which is fed to the Zero. The MCU then crops the image to focus on the eyes, applies a Gaussian blurring filter to reduce the effect of noise, and applies a Gaussian adaptive threshold filter to detect dark and light portions of the image. The threshold filter compares each pixel to those in the surrounding region. If the pixel is darker than the weighted average of its local neighborhood, it is set to zero intensity, otherwise it is set to the maximum intensity. The Zero then uses an image processing technique called a Hough Circle Transform. This detects any circles in an image which would correspond to an iris or pupil. Each circle detected has defined center points and radii. If the circle center coordinates place it on the left side of the image, this would correspond to a right eye open, or a left wink due to mirroring. Likewise, a circle on the right side indicates a right wink. When only one circle is detected, the wink detector will turn on a corresponding output pin which is sent as a control signal to the IOIO device.

A Hough Circle Transform is a computer vision technique that was taught in ECE 597IP Image Processing at the University of Massachusetts. In principle, the parametric equations $x=a+R\cos(\theta)$ and $y=b+R\sin(\theta)$ describe a circle [2]. The transform works by describing each circle in terms of variables a and b instead of x and y and letting x and y become constants. To detect a circle in an image, an edge detector is used first, in this case we used a canny edge detector. The algorithm then cycles through the image and increments the corresponding location in the a and b coordinate map each time it meets an edge in an x and y location. The peaks of this coordinate map correspond to the x and y locations of the centers of the circles with radius R . By repeating this process, we can search for a range of radius values. By altering the sensitivity of the edge detector and the threshold for the peaks in the a and b coordinate map, the algorithm can be calibrated to accurately detect pupils and irises. In addition, the wink detector can be set to only accept one circle within a selected pixel distance. Other calibration parameters include the minimum and maximum radii allowed, and the amount of blurring before the edge detector. Blurring allows the algorithm to reduce noise at the cost of computational expense and loss of detail.

Table 2

WINK DETECTOR REQUIREMENTS AND SPECIFICATIONS		
Requirement	Specification	Results
Accurate	Reads 90% of the frames correctly	Reads 85% of frames correctly in optimal light
Quick	Minimum 2 FPS	3.51 FPS
Connects to the IOIO	Controls 2 binary output pins for control signals	Met

Table 2 demonstrates the individual requirements for which this subsystem is responsible. This system will be the control interface for the user, so to maximize ease of use, the system will need to be quick and accurate. The whole algorithm will need to run in under a half a second for easy control of the device. The speed of the program from image capture to signals out was timed over 50 images and averaged .285 seconds or 3.51 frames per second. This is an improvement over the speed at MDR where the time from image capture to analysis was .793 seconds, since that algorithm required saving the image to memory and then analyzing it, while the current code analyzes directly from the video stream. Accuracy of the system was determined by taking a series of pictures of each configuration (both eyes open, left wink, right wink, and both eyes closed) and averaging the accuracy for each configuration. Unfortunately, this turned out to be dependent on the light conditions as the Pi Camera had difficulty in low light situations. The binary GPIO signals were sent to the Micro instead of the IOIO, but the wink detector still connects to the IOIO indirectly and from there, to the phone application.

Alternative designs for this system were considered. Initially, we planned to use a sample image of a closed eye and run a similarity metric using a calibration algorithm; this however was computationally heavy and not accurate. An eye tracking system was also considered. While this would give us more control over the device, concerns rose over the computational costs and thus the approach was abandoned. Using our current system, we would be able to detect if both eyes are closed and we could use that as an additional user input, but that system runs the risk of detecting normal blinks instead of control winks. If we used two frames to ensure there were no accidental blinks, then the system would force the user to keep their eyes closed for longer which would quickly become dangerous. Given all these concerns, we decided to pursue only a wink detector algorithm using the Hough Circle Transform.

Future work on the wink detector should have a focus on improving accuracy and adjusting for changing light conditions. A histogram equalization function should help standardize the image intensities and was attempted, but it had adverse effects with the rest of the image processing algorithms and we had difficulty incorporating it. Other systems that may have more success would include a Haar scaling function or an image segmentation method. Unfortunately, we were unable to test the final algorithm until late in the design period since it was dependent on the final configuration of the whole system and we were unable to flesh out these alternative methods to compensate, but they may prove successful to future progress on the subsystem.

C. Camera

The camera subsystem will function as the input to the wink detector. It will be mounted within the goggles above the user's field of view. In this location, it will capture both user's eyes and send images at a fixed rate to the microcontroller. Requirements for this subsystem include capturing both eyes, and capturing an image within a quarter of a second. The last

requirement is necessary to achieve the two frames per second requirement of the user interface specified in the wink detector block.

The camera chosen is a Raspberry Pi Camera Board v2. This has a resolution of 8 megapixels, a light weight of 3.4g with the cable, and a small profile of 25mm x 23mm x 19mm [3]. The camera contains a high-resolution Sony IMX219 Image sensor [4]. This is a high-speed sensor that has a lens shading correction function and two exposure controls to support Binning Multiplexed Exposure HDR (BME HDR). The lens shading correction suppresses optical unevenness. The BME HDR is an image processing technique that synthesizes both dark and bright aspects in the photo between short and long exposure images. Using these two different exposures, the camera can minimize the effects of high contrast. Table 3 below shows the relationship between image size and frame rate as provided by the datasheet.

Table 3
RELATIONSHIP BETWEEN IMAGE SIZE AND MAX FRAME RATE

Image Size	Frame Rate	Notes
3280 x 2464	30 fps	
1640 x 1232	120 fps	
1408 x 792	180 fps	High-sensitivity mode
1280 x 720	198 fps	High-sensitivity mode
960 x 540	240 fps	High-sensitivity mode

Due to the proximity of the camera to the user's face, we are using a fisheye lens to widen the aperture to approximately 180 degrees so that we will be able to fully capture both eyes. Figures 2 and 3 demonstrate the effectiveness of the lens. Each picture was taken with the camera at the distance it will need to be to fit within the goggle attachment. The fisheye lens allows us to easily capture both eyes and fulfill our requirement.



Figure 2. Captured image without the fisheye lens

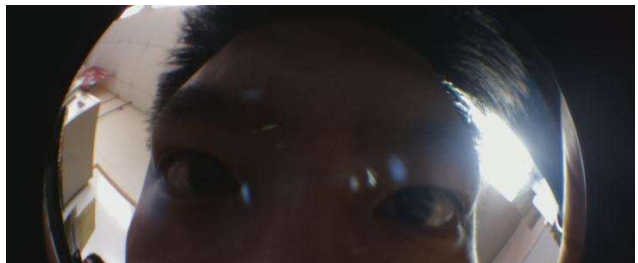


Figure 3. Captured image with the fisheye lens

D. Microcontroller

The microcontrollers will serve as the central hub between all the other subsystems. The MCUs drive the OLED for the

Team 6 Final Project Report

display, run the wink detector, power the camera, and communicate with the mobile device via the IOIO. These devices receive serial data from the IOIO and receive images as png files from the camera. In return, they output binary (GPIO) pin values to the IOIO and drive the display. The MCUs are able to manage the different tasks through asynchronously communicating with one another.

We are using a Raspberry Pi Zero to run the wink detection algorithm. This is a small microcontroller, only 65mm x 30mm x 5mm. This allows it to be easily used for a wearable device as it is not bulky or heavy (9g). The Zero has a BCM2835 processor with a 1 GHz ARM1176 core. It has 512 MB of RAM which will be enough to run both the wink detector and the OLED display code [5]. Several tests were performed on the Zero to ensure that it would meet general specifications listed for the project, namely power consumption tests and operational performance.

Table 4
ZERO PERFORMANCE STATISTICS

Performance Statistic	Value
Power Consumption	.7W (5V @ 140 mA)
Memory Usage (1 image)	9%-10% Max
Memory Usage (5 images)	12%-13% Max
Running Temperature	53 °C

As shown in Table 4 above, the Zero runs on low power consumption, while utilizing less than a fifth of the memory to run wink detection analysis. The different specifications represent how much memory is taken to analyze a certain number of frames (either one or five); as more frames are used the memory usage does not increase by more than 5% from the second row in the table. More importantly, the Zero does not heat up considerably, maintaining about 53°C while running. The temperature was measured using a Python script that accessed the internal temperature sensor inside the Broadcom processor chip; it provided reasonable accuracy after calibration. This will be a crucial aspect in our prototype design, since it is not desirable to have a very hot component located in proximity to the user (in fact, our specifications list that it cannot be more than 80°C, which it suffices).

To test the communication aspect of the microcontroller, first a serial data communication test was performed between the laptop terminal and the Zero. Text data was sent in both directions between the two devices, demonstrating the capability to communicate with the IOIO serially. However, a different option from using the Zero for both wink detection and OLED display was used because the Zero was unable to refresh the OLED display to show new information at around 2 FPS. As a result, all of the functions to power the OLED display and communicate with the IOIO serially were moved over to the Arduino Micro. This Micro does not have a lot of overhead (it is only running one program at a time, as opposed to the Zero which runs the whole Linux OS) and runs on C++, which is faster than Python. As a result, it was able to achieve our requirement to provide for fast OLED display refresh rates of at least 2FPS. In addition, it received data from the Android phone

like music information, text message data, and etc. serially to project it onto the display. This was validated by connecting the IOIO to the Micro, which was powering the OLED display, and visually inspecting that phone data was being printed on the screen.

With the introduction of a new microcontroller, some changes had to be made for communication protocols. As previously mentioned, the IOIO talks to the Micro serially, providing the phone data. The Micro also powers the OLED display using hardware Serial Peripheral Interface (SPI), which uses the dedicated SPI pins on the board to provide much faster data rates than serial transmission. The wink detection, which originally sent out the GPIO signals to the IOIO from the Zero, now sends that information to the Micro, which also houses a finite state machine to control what the user sees in his vision and what data to send from the IOIO. The GPIO signals from the wink detection drive into the interrupt pins on the Micro, which asynchronously interrupt the loop it is running to change the menu display and receive different kinds of information from the IOIO. A more precise explanation of the communication signals is shown in the block diagram (Figure 1) and denoted later in this section. The finite state machine is shown in Figure 4.

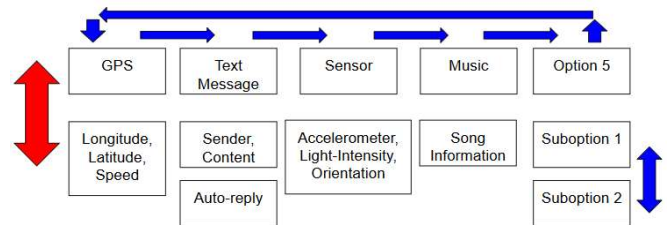


Figure 4. Finite state machine for control menu housed in the Micro. Blue signifies right winks, red signifies left winks

When the user wears the goggles, it will begin by displaying the top level menu, beginning with GPS. If the user would like to access the GPS, he winks with his left eye, which will bring him to a submenu of the GPS showing Longitude, Latitude, Speed data. To exit the submenu and return to the top level, the user winks left again (hence the bidirectional red arrow between the top level and the sub-functions). To traverse laterally through the menu, the user winks with his right eye. So from the GPS top level menu the user will go to the Text Message top level menu. The user can once again enter the Text Message menu through a left wink, which will first show information about the person who sent the message and what the message contents were. If there are multiple sub-functions within a menu, the user can right wink to traverse (it is a loop so at the end it will revert back to the first sub-function). Again, the user can exit at any subfunction to the top level menu by winking left. The top level menu is also in a loop, going from the last option to the first option (GPS). This menu can be scalable, as shown in Figure 4 that all that needs to be implemented are the sub-functions and then the top level menu option itself in the code. Depending on the state the Micro is in, it will request different data by signaling with two different GPIO pins (one

Team 6 Final Project Report

for each sub-function) to the IOIO and will receive that information through five other GPIO pins (one for each top level menu) from the IOIO. Our PCB has a maximum of three sub-function pins and six data pins; this can be further scaled for more different options and sub-functions by soldering more connections between the Micro and the IOIO.

Finally, we tested for whole product functionality in different environments, such as putting it in a refrigerator for 2.5 hours and running the code outside where it was at least 25°C. In Table 5 we list specifications required for our product.

Table 5
MICROCONTROLLER REQUIREMENTS

Requirement	Specification	Results
Maintain communication between Android application and OLED (support Wink Detector and OLED)	Constant link of sending GPIO signals and power OLED display	Met through running entire system for 4 hours without interruption
Drive Pi Camera	Minimum 2 FPS	Met
Does not overheat	Temperature remains less than 80°C	Zero: 53°C Micro: 23°C
Works within reasonable temperature	Operating range -20°C to 30°C	Met by putting both microcontrollers in refrigerator for ~2.5 hours
Low power consumption	<1W	Zero: .7W (5V @ 140mA) Micro: .135W (5V @ 27mA)
Memory constrained	<50% of RAM	Zero: ~18-20% Max for infinitely running code
Size	<80mm x 80mm x 10mm	Zero: 66mm x 30.5mm x 5mm Micro: 48mm x 18mm x 5mm

The requirements for this subsystem are shown above in Table 5 alongside test results. As shown in the table, the temperature requirements, memory constraints, and power consumptions have all been met with both the Micro and the Zero. More importantly, it is able to run the system and provide end-to-end feedback between namely the phone, wink detection, and the OLED display through asynchronous interaction between these systems.

E. Mobile Phone Application

The mobile phone application collects data from the phone and sends information through the MCU to display. It will receive input commands from the IOIO to adjust the information it is displaying or to control the phone. We used Android Studio to write the phone application for Android devices. The application is currently designed for the Huawei Honor 5x smartphone, which uses Google Android 5.1 OS [6]. The requirements of this subsystem are shown in Table 6.

Table 6

PHONE APPLICATION REQUIREMENTS

Requirement
User friendly interface
Gathers information from phone sensors
Sends messages and phone number data
Able to gather GPS data and calculate the speed
Allows IOIO to manipulate the phone
Able to manipulate music

The phone application can currently turn music on and off from I/O pin input, displays information from the phone sensors, and is accessible to users as seen in Figure 5.

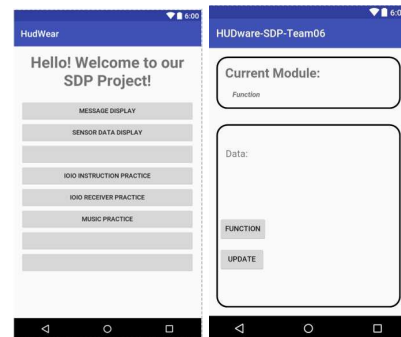


Figure 5. Main menu layout for testing (left), HUDware working module (right)

The layouts for the application have already been completed. We have implemented music, statistical data extraction, and part of the message display functionalities.

The music function allows the user to listen to songs and change the song sequence via I/O pins. First, we needed to add “MusicService” in “AndroidManifest.xml” which guarantees the application permissions it needs to do system level methods. We imported .mp3 files into the resource directory, allowing us to call music within the application. In the Java class, we created a media player to recall those songs. This media player can be controlled through GPI/O pins.

The sensor layout seen above shows the phone’s sensor sets, which we are using to display information about our surroundings. Using these sensors, we are able to detect light intensity, gravity, acceleration, and orientation. With GPS data and calculations, we expect to be able to determine speed as well. Finally, all these kinds of data have been extracted and send out to the MCU through IOIO serial data communication.

The message display function has been completed. We use an Android SMS Message API to help the application show messages when they arrive. “Broadcast Receiver” has been implemented for monitoring the status of Message service. When the message comes in, the sender’s phone number and message content will be reserved and then sent out to the MCU through IOIO serial data communication. At the beginning, we tested message function on virtual device. After it was completed, we moved to the real device with an SIM card.

The GPS function has been implemented by using Google-Play-Service. It will return the accurate longitude and latitude data once the phone is connected to the Internet and it is under GPS function. We are able to get new longitude and latitude by configuring time intervals and minimum distance of updating the data. In this case, updated longitude and latitude data can be used to calculate the distance by using Haversine Formula, as shown below in figure 6. The speed can then be calculated using distance over minimum time interval.

$$\begin{aligned} \text{hav}\left(\frac{d}{r}\right) &= \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1) \\ d &= r \cdot \text{hav}^{-1}(h) = 2r \cdot \arcsin\left(\sqrt{\text{hav}\left(\frac{d}{r}\right)}\right) \\ d &= 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \end{aligned}$$

d is the distance between the two points
r is the radius of the sphere
 φ_1, φ_2 : latitude of point 1 and latitude of point 2
 λ_1, λ_2 : longitude of point 1 and longitude of point 2

Figure 6. Haversine Formula

Figure 7 below shows the test circuit used for the application. This test circuit includes the IOIO device which will be described in more detail later. An external control circuit was used to simulate an I/O pin turning on and off. Using this system, we could control the phone via external switches.

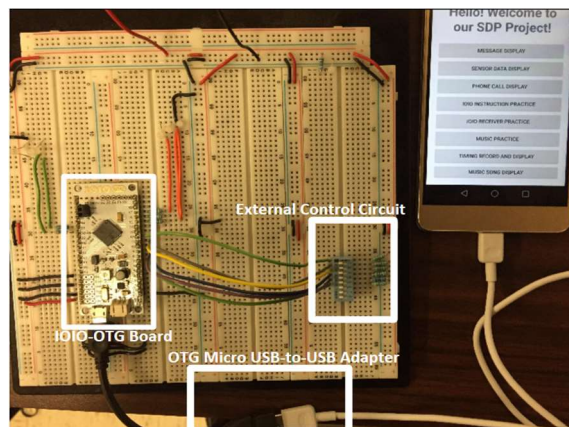


Figure 7. Phone Application, IOIO-OTG Board, and External Control Circuit

F. IOIO-OTG

The IOIO-OTG board connects the mobile phone to the MCU, namely the Micro. Using a wired USB connection, this device allows us to communicate at a fast rate. Usually with a mobile device, the phone is default set to slave in the master/slave paradigm. This device allows us to swap back and forth to fit the needs of the project. This subsystem needs to provide stable serial data transmission, read GPIO and analog inputs, and write GPIO outputs.

The Android IOIO-OTG board is a development board that uses a PIC microcontroller [7]. It provides two pairs of serial

data communication, 46 GPIO pins, analog input, SPI, and I2C interfaces to handle external hardware. This provides enough functionality to handle all the requirements needed from this subsystem.

Table 6
IOIO PROTOCOLS AND SPEED

Protocol	Throughput t	One-way Latency	Stability
Open Accessory	600 kB/S	1 ms	High
Android Debugging Bridge	300 kB/S	4 ms	Low

Table 6 shows the different speeds for the two protocols available on the IOIO. Clearly, the Android Open Accessory (AOA) protocol is more attractive. In addition to the speed and stability, with AOA we do not need to set the phone in USB debugging mode as we do for the Android Debugging Bridge (ADB). ADB is commonly used for downloading Android code from the desktop. We have implemented the IOIO transmitter by importing the IOIO API and the Utility library. The transmitter can control each GPIO pin and set them to either high or low from the phone application. It sends serial data at common baud rates, e.g. 9600, 19200, 38400, and 115200.

IOIO sends out the serial data through UART. For the input, we define two kinds of GPIO pins: state pins and control pin. State pins decide the Phone's functionality, such as Music, GPS, Message, and Sensor Data. Each function corresponds to one specific pin. Apart from that, we have control pin to enter the function or exit the function. For example, if user wants to enter the message function and see message information on the display. Message function's state pin will be on first, and then control pin will be on then to enter the function and message content will be presented on the OLED display.

Alternative approaches to the IOIO device were considered for communicating between the application and the microcontroller. These options are shown in Table 7 with their respective bandwidth, range, and complexity. Bluetooth offers the lowest bandwidth but is straightforward to implement. Wi-Fi has the highest bandwidth, but would be difficult to implement due to protocols. Both Wi-Fi's and Bluetooth's data transmission can be unstable for dynamic situations like skiing. Since wireless connectivity was not necessary, the IOIO device proved to be the best option due to its stability and low complexity.

Table 7
COMMUNICATION LINK DESIGN ALTERNATIVES

Communication	Bandwidth h	Range	Complexity
Bluetooth (v4.0)	800 kB/s	30 ft	Pairing Devices
Wi-Fi (802.11n)	11 MB/s	300 ft	Accessing Internet
IOIO-OTG	5 MB/s	Wired	Configuring Board

G. Display

The display for this device will be generated by an organic light emitting diode (OLED) display that reflects off a transparent material. An OLED works by having a series of thin organic films between two conductors. By applying current,

light is emitted from the device [8]. The device we are using is driven by a SSD1351 chip. To provide necessary setup and functionality for the display, we are using a library from Github, `py-guagette_master_2` [9]. Using this library, we can display png images as well as text and ASCII characters.

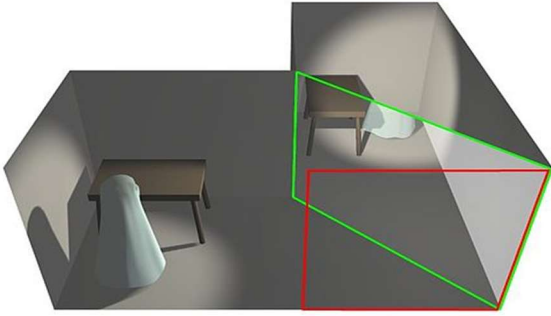


Figure 8. Pepper's Ghost Diagram

The AR display works on a principle called Pepper's Ghost shown in Figure 8 above [10]. This effect uses two light sources, in our case the natural light of the environment and the light from the OLED [11]. By placing a film in the green plane, some of the light from the OLED will appear to be directly in front of the user. This allows us to hide the OLED on the side of the goggles outside of the user's field of view and still create a transparent display. For this subsystem, we are using a sheet of LEXAN clear acrylic. The optimal angle for this sheet, or angle where the user sees total reflection, is 42.2° from the red plane [12].

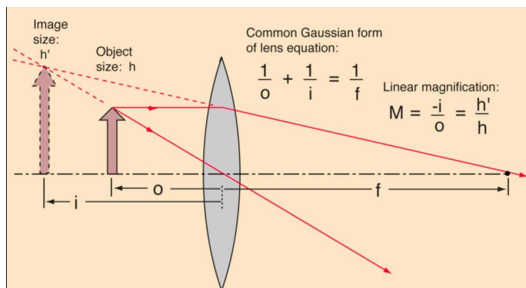


Figure 9. Virtual Image

For purposes of focusing the image for a general user, we added a 10cm focal length BiConvex lens in the optical path. This allows us to achieve a focused image within the user's vision. A normal human eye cannot perceive an image that is within 25cm of the eye, we needed to create a virtual image that appears to be coming from a distance greater than 25cm. With our 10cm focal length lens, we are able to achieve an image that appears to come from about 40cm away which is sufficient to meet the 25cm criteria.

Table 8
DISPLAY REQUIREMENTS

Requirement	Specification
Focused and Legible Display	
Brightness Minimum	90 nits
Operable at reasonable temperature	$-20^\circ\text{C} - 30^\circ\text{C}$
Low power consumption	<1W

The requirements for this subsystem are shown above in Table 8. This display needs to be bright enough to be legible while skiing in the middle of the day and needs to be able to work under cold temperatures. Currently we are able to meet the specification on brightness although just barely with 91 nits as measured from a light meter.

As previously mentioned, between MDR and FDR a switch was made from a Raspberry Pi Zero running the OLED display, to an Arduino Micro. This change was done in order to increase the refresh rate of the OLED. With the Zero, we were getting refresh rate of a frame every 3 seconds (.333FPS), which does not make sense to have a refresh rate so low that it does not allow for fast enough display of the incoming data. With the Micro, we are able to achieve a much faster refresh rate of roughly at least 2FPS.

A few different design alternatives were considered for this display. Texas Instruments has a Pico projector chipset that would have fit the requirements of the projector well, but the chipset itself was difficult to assemble due to their format as ball grid arrays. A transparent OLED was also considered, as it would provide the light required without needing a reflective film, but these proved impossible to procure. Given the complexity and ease of manipulation, an opaque OLED was chosen for the display.

H. Power System

The power system will support the HUDware device for its outdoor setting. This system will need to supply power to the IOIO and MCU, and by extension, the camera and OLED. This subsystem will supply enough power and energy to run the device for four hours. Table 9 shows the power drain for each system.

Table 9
POWER BUDGET

Device	Current Draw
Zero	140mA
Micro	27mA
OLED	0.4mA
IOIO	60mA
Total	227mA

We decided to use an external phone battery to power the device. This provides 2200mAh which will prove more than enough to reach our goal of four hours of battery life [13]. It is worth noting that the battery is not located within the casing as previously intended, but is instead attached by a USB cable and designed to rest in the pocket with the user's phone.

III. PROJECT MANAGEMENT

Table 10
MDR GOALS

Goal	Specification	Achieved
Projector Display Assembled	Prototype Prism Constructed	Yes
	OLED Display Setup	Yes
Application Layout	Partial Functionality	Yes
Wink Detector	Wink Detection Algorithm	Yes
	Moved into microcontroller	Yes
MCU communication setup	Serial to MCU	Yes
	Serial from MCU	Yes

Table 10 lists the goals our team aimed to accomplish by the midway design review presentation date. Each subsystem had separate objectives for the MDR presentation. The projector display needed to have a basic system in place so that an image was visible to the user. This was not intended to be a final configuration so much as a proof of concept. To this end, a makeshift prototype was created that showed the OLED display image across the field of view of the user. Further work in this subsystem includes optimizing reflection, finalizing OLED placement, and converting text data into a display that can be shown to the user. While there is plenty of work to be done in this area, the display has been created to the point specified at the preliminary design review.

The mobile phone application subsystem has all required functions and it is integrated with other subsystems. At the time of the presentation, it would take in sensor data, GPS data, speed data, and message content from the phone itself and display the information on the OLED Display. Sensor data is extracted from phone's own sensors. Message information includes both sender's phone number and the content of message. Speed data is calculated from the Haversine formula based on the time interval and the changed distance. Apart from that, Music function is able to skip the song and restart back to the first song. It is controlled through microcontroller's control signals.

There were two milestones in the wink detector subsystem. The first was a functional algorithm that would detect if one eye was closed in a picture of the user's face. The second portion of this subsystem was moving it from a computer into the Zero. Currently, both milestones have been reached and there is a functional wink detector inside of the MCU. In order to complete these tasks, the wink detector code first was designed on a computer. For ease of use, this was done in MATLAB. An equivalent program was developed in Python so that the microcontroller would be able to run the code. Finally, this had to be imported into the microcontroller which involved calibration and importing the correct libraries. Like the mobile phone application and the IOIO device, the camera sensor subsystem falls closely in line with the functionality of the wink detector. Although no deliverables were specified, the camera is currently able to take a picture of both eyes from a distance that would fit comfortably within the goggles. Further work in both subsystems would include calibrating the wink detector, improving the speed of the whole algorithm, testing the system to ensure requirements have been met, and permanently fixing the camera to the prototype attachment.

Since MDR, the wink detector was moved from the Raspberry Pi B+ to the Zero. In addition, filters and calibration were added to improve accuracy of the live stream. The overall latency of the algorithm was drastically reduced thanks to a change in the handling of the video stream.

While there are several components to the microcontroller subsystem, for the MDR presentation date, we specified that we would have communication to and from the MCU. As promised, we sent serial data from the microcontroller to a desktop computer and from the computer to the microcontroller, just using two different MCUs. This past semester, the Micro became the dedicated MCU to drive the OLED display, while the Zero ran the wink detection algorithm. Otherwise, the overall system is still able to send information bi-directionally; it is just through a slightly different way for the reasons explained earlier in the MCU section of the report.

In the past semester, two new subsystems were incorporated into the overall design; the power system and the casing. The power system is driven by an external phone battery which rests outside of the main casing. This supplies enough energy to power the system for 9.69 hours. The power is connected directly to the Zero which in turn supplies it to the IOIO, phone, Micro, OLED display, and the Pi Camera. The casing was designed in a 3D modeling software and printed out in two parts. Each part was designed to hold the three components of the OLED display subsystem as well as the PCB. After printing, the pi camera was attached using sheets of plastic, and a foam cushion added to line the perimeter of the headset. Tabs on both the top and bottom halves allowed the casing to be connected with screws and nuts.

Since we have complex system, including one Arduino Micro, one Raspberry Pi Zero, IOIO Communication Board, and OLED display, and all these things have to be fitted into the Casing prototype, which has limited space. We decided to design our Printed Circuit Board (PCB) with Altium Circuit Maker to reduce the complexity of wiring. The PCB was sent to Advanced Circuits for fabrication. When the board is received, it was soldered with male and female header pins. After finishing soldering, ground continuity was checked before plugging in the components. The PCB was then placed in the goggles.

The team worked well together. For the first semester, most of the work was done with individual subsystems, requiring each member to work independently. While the bulk of the work was done individually, when a team member had difficulty achieving a goal, other team members stepped in to assist. Most commonly, this duty fell to Lee, whose subsystem impacted everyone's, putting him in the unique position of being involved slightly in every subsystem. The second semester work required each of the subsystems to join together. This required significantly more communication, but we met this challenge head on. The team has been able to communicate with each other through both a group messaging platform as well as weekly meetings. The group meets once a week as a team to work on current action items, and again later in the week to discuss progress and goals with our faculty advisor.

Team 6 Final Project Report

This allowed us to achieve our goals for the project and resulted in a functioning prototype.

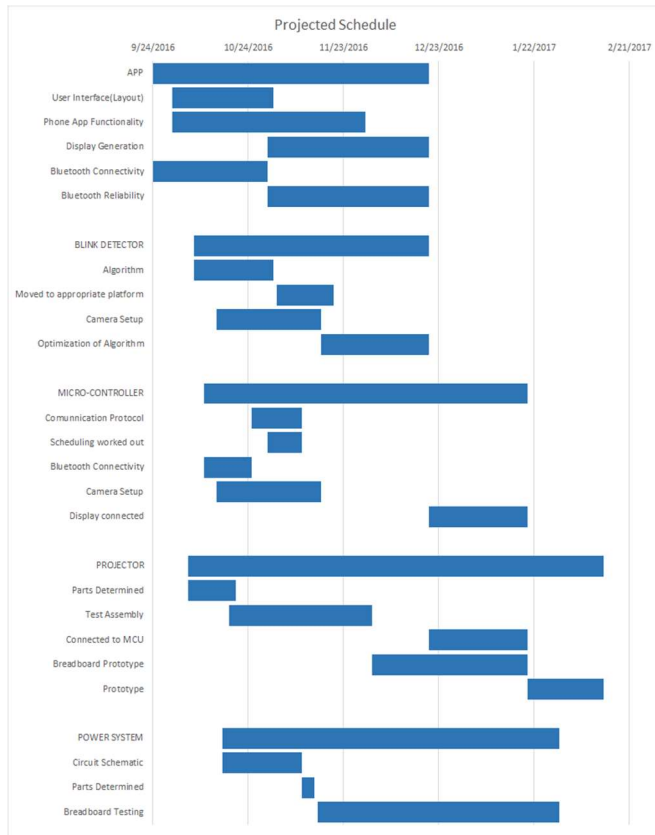


Figure 10: Schedule for the first semester

Figure 10 demonstrates the goals that we have set for the fall semester, with the deadline of the midway design review. This chart was created early in the project and understandably represents somewhat different goals than what we decided to pursue. For instance, the original design includes Bluetooth connectivity, a feature we decided to scrap in favor of a wired USB Connection. Some of the work we anticipated being able to achieve was not accomplished, such as scheduling on the microcontroller and any work on the power system. These were conscious decisions aided by the advice of our evaluators to focus our attention on more important priorities.



Figure 11: Revised schedule for second semester

Figure 11 is the proposed schedule for the second semester. This aimed to have all tasks completed by CDR. Unfortunately, this proved erroneously optimistic. While a prototype was developed by CDR, it has been drastically reworked since that presentation to improve ergonomics, display quality, and control accuracy. By FDR and Demo Day, all tasks were completed, all subsystems combined, and a functioning HUDware was displayed.

IV. CONCLUSION

HUDware has been completed and will function as we planned at the beginning of the project. All the various subsystems have been combined and communicate together to form a cohesive heads-up-display. That being said, there is room for improvement in many of the subsystems to improve clarity, ease of use, and quality increases. In addition to simply improving the quality of HUDware, we as a group would like to encourage future work on the project to include finding new applications to a portable display. We selected a skiing application due to personal familiarity with the sport, but we would love to see another group take this project in a new direction.

ACKNOWLEDGMENT

We would like to acknowledge the assistance of several members of the University of Massachusetts faculty. Professor Parente assisted us in the direction of the wink detector algorithm. Professor Holcomb aided with the microcontroller section. Professor Wolf provided much needed direction for the project as a whole given his experience in working with another Senior Design Project group that had similar ambitions as we have. In addition to these professors, we would like to thank John D’Errico of Eastman Chemical Company who provided us with a sample laminate for the projector medium and Bruce Platt for the suggestion of the weatherproofing adhesive for the cushion on the casing.

REFERENCES

- [1] "Microsoft HoloLens: Mixed Reality Blends Holograms with the Real World," Feb. 29, 2016. [Online]. Available: https://www.youtube.com/watch?v=Ic_M6WoRZ7k. [Accessed: Dec. 13, 2016].
- [2] H. Rhody, "Lecture 10: Hough Circle Transform," Rochester Institute of Technology, 2005.
- [3] "Raspberry Pi Camera Board v2 – 8 Megapixels," *adafruit.com*, [Online]. Available: <https://www.adafruit.com/products/3099>. [Accessed: Dec. 13, 2016].
- [4] "IMX219PQ," *sony-semicon.co*, [Online]. Available: http://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html. [Accessed: Dec. 12, 2016].
- [5] "Raspberry Pi Zero - Raspberry Pi". *Raspberry Pi*. N.p., 2016. Web. [Accessed: 4 Oct. 2016].
- [6] A. Dobie, "Honor 5X Specs," *androidcentral.com*, Jan. 6, 2016. [Online]. Available: <http://www.androidcentral.com/honor-5x-specs>. [Accessed: Dec. 12, 2016].
- [7] Ytai. "IOIO Documentation," *github.com*, Oct. 14, 2016 [Online]. Available: <https://github.com/ytai/ioio/wiki>. [Accessed: Dec. 12, 2016].
- [8] "OLED Introduction and Basic OLED Information," *oled-info.com*, [Online]. Available: <http://www.oled-info.com/introduction>. [Accessed: Dec. 8, 2016].
- [9] Guyc, "Py-Gaugette," *github.com*, Nov. 5, 2016. [Online]. Available: <https://github.com/guyc/py-gaugette>. [Accessed: Dec. 8, 2016].
- [10] "Pepper's Ghost," *wikipedia.com*, [Online]. Available: https://en.wikipedia.org/wiki/Pepper's_ghost. [Accessed: Nov. 12, 2016]
- [11] B. Costa, "Explaining the Pepper's Ghost Illusion with Ray Optics," Jan. 11, 2016. *comsol.com*, [Online]. Available: <https://www.comsol.com/blogs/explaining-the-peppers-ghost-illusion-with-ray-optics/>. [Accessed: Nov 20, 2016]
- [12] "Optical & Transmission Characteristics," *plexiglas.com*, 2000. [Online]. Available: <http://www.plexiglas.com/export/sites/plexiglas/.content/medias/downloads/sheet-docs/plexiglas-optical-and-transmission-characteristics.pdf>. [Accessed: Dec. 13, 2016].
- [13] "Power Pack," *southerntelecom.com*, [Online]. Available: <http://www.southerntelecom.com/polariodsupport/downloads/PPP2255.pdf>. [Accessed: Feb. 28, 2017].