**Fall**

12

**University of Michigan**

Glasses

Russ Bielawski, Joe Romeo, Justin Paupore

The following document provides all the information needed to build and modify a minimally intrusive and affordable gaze tracking system.

08

**Fall**

Chapter 1 - Introduction and Overview

Chapter 2 – Architecture Details

In this chapter we will look at the implementation details of both the software and hardware used in the gaze tracking system. This chapter will give you a good idea of why we made certain design choices, both software and hardware, and will also offer suggestions for future improvements to the overall design.

# Section 1 Hardware

In this section, we will discuss the hardware used in the gaze tracking system. It will be useful to have the eagle design files open, as we will be discussing the hardware used in the design. We will however provide pictures of each part as we discuss them so it is not necessary. We will also use this section to suggest some design changes for future revisions of the hardware.

INSERT HIGH LEVEL HARDWARE DIAGRAM HERE

## Teensy

Lets start by taking a look at the teensy, the heart of our hardware. The teensy is a small form factor arm development board that allows access to the Arduino IDE and libraries. This makes the teensy an ideal platform for our application. There are many small form factor, Arduino compatible boards, but this one has a lot of extra power necessary for our application. There are also many small form factor arm development boards, but none offer Arduino compatibility.

In the spirit of Arduino, we decided to make a teensy “shield”. For those unfamiliar with Arduino shields, they are boards that can be plugged into the Arduino to allow easy access to peripherals. We designed a board that can easily be plugged into the teensy, that gives teensy access to two cameras, a battery/battery charging circuit and wireless Bluetooth communication. The following sections will outline each of the hardware components used in the teensy shield.

Note: for future iterations of the gaze tracking system, we suggest combining the teensy and teensy shield into one system. At this point, the teensy is being used purely as a way to upload code over USB. This means there is a lot of wasted space to unused pins that are pinned out on the teensy unnecessarily. The teensy bootloader used to upload code via USB could easily be replaced with a cheap JTAG arm debugger. This would add both debugging capabilities, and reduce the size of the overall system.

## Cameras

Currently, both cameras are connected to 8-pin JST connectors. Each camera shares 5 digital control lines, a power line and a ground line. Both cameras have dedicated analog lines. The digital lines are connected to 5 digital outputs on the teensy. The teensy sends commands across these digital lines, and the cameras respond with the requested pixel value on the analog line. Each analog line feeds into a different ADC on the teensy, where it is then converted into a usable pixel value. The exact details of the commands sent, and the analog data received will be discussed more in the software section of this document. After the pixel data is gathered, it is sent wirelessly via Bluetooth to a host for processing.

Note: The cameras we used were great! The company that we bought them from, however, is currently working on making a smaller version of the breakout board for the camera. We suggest keeping an eye on the centeye website for updates on that product, as it would greatly reduce the obtrusiveness of the gaze tracking system.

## Bluetooth Module

The Bluetooth module we chose is the SPBT2632C2A. We chose the SPBT2632C2A because it is very small, low power, and offers high bandwidth compared to other Bluetooth modules of similar price. We have the reset pin of the Bluetooth module hooked up to a N-channel MOSFET that is hooked up to a GPIO pin on the teensy. This allows us to easily reset the Bluetooth module, and get into a known and configurable state in case something out of the ordinary happens to the Bluetooth module. One of the GPIO lines of the Bluetooth module is hooked up to an LED. This was meant to serve as a “Bluetooth connected” indicator, but we never added that functionality into the Bluetooth module firmware. We also have the UART and SPI lines of the Bluetooth module hooked up to UART and SPI lines on the teensy. As you can see from the schematic, RTS and CTS are tied together and that was a mistake. See design suggestions below for how to fix this mistake.

Note: We originally chose the SPBT2632C2A as our Bluetooth module of choice because it had SPI, and we were worried UART would not be fast enough to transmit data fast enough for us to do gaze tracking. It turns out that UART with flow control will reliably max out the Bluetooth bandwidth, at a baud rate of 921600. We therefore recommend that future iterations of the board remove the SPI connections to the teensy, and route RTS and CTS instead. We also recommend adding in a connected status indicator LED into the Bluetooth module firmware.

## Batteries, Voltage Regulators, Charging Circuit, and Fuel Gauge

We chose 600 mAh, 3.7V, lithium ion battery with a form factor similar to a AA battery. This proved to be a great choice in a battery because the form factor fit perfectly on the gaze tracking system. Lithium ion also adds charging capabilities, and doesn’t have issues with memory effects of other batteries, allowing for many more recharges than other batteries. Because all of our circuitry runs at 3.3V, this made 3.7V a very nice feature of the battery. We are able to run the gaze tracking system off of 1 battery with a simple low dropout regulator that regulates the 3.7V of the battery to 3.3V. Although we lose a bit of power efficiency by using a low dropout regulator, our system is very low power, and typically runs at less than 200 mA in full blast. This makes losses to linear regulation inconsequential.

To charge the battery, we use a lithium ion charging IC. To charge, the switch on the board should be switched to “off”. If the IC is successfully charging the battery a red LED will turn on. To check the battery life, we put a fuel gauge on board. We can send a read command to the fuel gauge over I2C, and it will respond with the percent of battery left.

NOTE: We never got the fuel gauge to work. The software side of things was very simple, but the form factor of the fuel gauge was extremely small and difficult to solder. The pads on the IC are also hidden which makes debugging and examining for solder bridges very difficult. We recommend future board iterations use a larger form factor fuel gauge.

# Section 2 Software

In this section we will discuss the software used in the gaze tracking system. There are two main parts of code: the code on the embedded side that takes video and transfers it to a host for processing, and the host processing code that takes the video stream and does real-time pupil tracking. We will start with a look at the embedded side of the code, and then we will look at the host processing side of the code and look more at the algorithms used in gaze tracking. It will be useful to download the source code using git as we will be referencing the code throughout this section.

INSERT SOFTWARE ORGANIZATION CHART HERE

## Embedded Software

As discussed before, the teensy has the Arduino libraries available. This should make the embedded side of the software familiar for Arduino veterans. We did, however, remove the Arduino software from the picture. Our code uses the Arduino libraries, but all of the code is written in C and C++. We replaced the Arduino IDE with our own makefiles. We decided to get away from the Arduino IDE because it was sluggish as an editor and required us to rebuild the entire Arduino library suite everytime we wanted to compile.

As we are not using sketches, and you can’t just open up our programs in the Arduino IDE and hit play, let’s look at the general organization of the embedded software, and how a program can be built and loaded onto the teensy. Start by navigating to the git root directory. From the root directory, navigate to sw/embedded/impl. In this folder, you will see many other directories. The important ones to note are common, calibrationcalculation, and dualcameracapture.

common contains all of the necessary libraries, including the Arduino libraries, and the stonyman libraries (the libraries that allow us to capture data from the stonyman cameras). calibrationcalculation is a program that must be run once for each camera. We did not integrate this in with the rest of the software due to program size constraints. dualcameracapture is where all of the camera data is captured and sent to the Bluetooth module for wireless transmission.

BREAKDOWN OF EACH PIECE OF SOFTWARE (will leave libraries out and mention that there are very few modifications of the original arduino and ardueye libraries).

## Host Processing Software

DISCUSS PROCESSING SOFTWARE HERE, TALK ABOUT RFCOMM0, PIPING OF FILES, AND MACHINE LEARNING ALGORITHMS USED

Chapter 3 – Milestones, Schedule and Budget

# Section 1 Milestones

# Section 2 Schedule

# Section 3 Budget

Chapter 4 – Things Learned

* Stranded wire handles mechanical stresses much better than solid core wires. When we used solid core wires to interface our cameras, they were very inflexible and the wires easily broke.
* Flow control is great! Without flow control we were only reliably sending data at baud rates up to 230400 bps. With hardware flow control, we were able to easily send data at a baud rate of 921600, maxing out the speed of our Bluetooth module.
* PTCs (positive thermal coefficient) are your friend. If you know certain parts of your board will blow if you exceed a certain amp rating, it is useful to use a PTC. The PTC acts like an open circuit above its amp rating, but will become a short after it cools down and the circuit returns to normal current levels. This saved us from frying many boards during testing phases.
* Machine learning is amazing black magic! It can turn very complicated problems that would require months of algorithm development in matlab into very simple problems that turn into just a few lines of code.
* Sometime smaller isn’t better! Although using 0402 smd components and a very small TDFN package fuel gauge made routing easier and our board slightly smaller, it made soldering very difficult.
* When creating small components in Eagle, in can help to make the solder mask aperture smaller. When spreading paste on top of a PCB stencil, a smaller aperture will cause less solder paste to get in, and will be less likely to create shorts. We found that a smaller solder stencil aperture may have helped us to successfully solder the tiny TDFN package fuel gauge.
* Don’t bury traces in testing phase. A few times we were unable to rework boards because the traces that needed to be cut or green wired were covered by other components (we realize sometimes this may not be possible, but it’s something to keep in mind).
* If making small boards, covered vias can be important. At least once, we added a bit too much solder to a pad, and the pad shorted to a via it wasn’t supposed to be connected to. This leads to the next lesson learned, which is if space can be afforded, keep vias away from pads.
* Getting a board out early is essential, especially when some components must remain untested until the board arrives. In our case, we had SMD connectors that made poor connections to our cameras. If we had time to rework, we would have replaced them with 0.1 inch holes to solder the camera wires directly to. This leads us to our next less learned which is…
* …If you have untested components on the board in a first revision, add a special section to the board for alternate options. In our case, we could have quickly ditched the poor connectors and soldered the cameras directly into the board. This would have saved us time trying to make the connectors work, and would have given us time to get a second revision on the board.

Chapter 5 – Team Contributions

|  |  |  |
| --- | --- | --- |
| Team Member | Contribution | Effort |
| Russ Bielawski | Prototype, software development and algorithm development | 33% |
| Joe Romeo | PCB design and algorithm development | 33% |
| Justin Paupore | PCB design and hardware assembly and debug | 33% |
| Others | Miscellaneous help | 1% |

Chapter 6 – Parts and Budget

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part Name | Description | Qty | Total Cost | Link |
| Teensy | Microcontroller development board | 1 | $19.00 | [link](http://www.pjrc.com/store/teensy3.html) |
| SPBT2632C2A | Bluetooth Module | 1 | $32.00 | [link](http://www.digikey.com/scripts/dksearch/dksus.dll?vendor=0&keywords=spbt2632c2a) |
| Stonyman | Camera | 1 | $109.98 | [link](http://centeye.com/products/stonyman-vision-chip-breakout-board/) |
| Max17040 | Fuel Guage | 1 | $2.27 | [link](http://www.digikey.com/product-detail/en/MAX17040G%2BT/MAX17040G%2BTCT-ND/2699346) |
| MPC7383IT | Charging IC | 1 | $0.68 | [link](http://www.digikey.com/product-detail/en/MCP73831T-2ACI%2FOT/MCP73831T-2ACI%2FOTCT-ND/1979802) |
| TC1262-3.3V | 3.3V LDO | 1 | $0.56 | [link](http://www.digikey.com/product-detail/en/TC1262-3.3VDB/TC1262-3.3VDB-ND/1980071) |
| EVP-AA202K | Buttons | 2 | $1.02 | [link](http://www.digikey.com/product-detail/en/EVP-AA202K/P13348SCT-ND/822285) |
| JS202011SCQN | Switch | 1 | $0.44 | [link](http://www.digikey.com/product-detail/en/JS202011SCQN/401-2002-1-ND/1640098) |
| SM08B-ZESS | Camera Connectors | 2 | $1.86 | [link](http://www.digikey.com/product-detail/en/SM08B-ZESS-TB(LF)(SN)/455-2401-1-ND/1887054) |
| M02-JST | Battery Connector | 1 | $0.95 | [link](https://www.sparkfun.com/products/8612) |
| Miscallaneous | LED: 3 (< 3v drop, 0603)  PTC: 1 (500 mA, 0603)  0.1uF: 3 (ceramic, 0402)  1uF: 4 (ceramic, 0402)  10uF: 3 (ceramic, 0402)  56Ω: 3 (0402)  150Ω: 1 (0402)  4.7k Ω: 1 (0402)  NMOS: 3  Header, crimp pins, housings | N/A | ~$5 |  |