Near-Eye Heads Up Display for Action Sports

Baechler, Jeremy
M.S. Computer Engineering
Colorado State University
Fort Collins, United States
jeremy.baechler@colostate.edu

Garcia Zamora, Isai
B.S. Computer Engineering
Colorado State University
Fort Collins, United States
isaig7@colostate.edu

McPhearson, Teague
B.S. Computer Engineering
Colorado State University
Fort Collins, United States
teague.mcphearson@colostate.edu

***Abstract*—Performance skydivers and wingsuiters often lack real-time feedback on their flight metrics and safety information during jumps. To address this issue, we present the development of a Heads-Up Display (HUD) system designed to provide comprehensive real-time data integration for skydivers. Leveraging components such as the Raspberry Pi 4, u-blox SAM M8Q GPS module, and MPL3115A2/BMP280 Altimeter Modules, our system aims to enhance flight awareness, safety, and performance optimization. The HUD interface offers customizable display options and standard configurations, allowing users to access pertinent flight data tailored to their needs. This paper details the system architecture, implementation challenges, hardware, and software evaluations, along with the achieved results and implications for the field of performance skydiving and wingsuiting.**

***Index Terms*—HUD, Heads-Up Display, Skydiving, Wingsuiting, Real-time Feedback, Raspberry Pi, GPS Module, Altimeter, Custom PCB Design, Printed Circuit Board, User Interface, Qt Creator.**

I. Introduction

Performance skydivers and wingsuiters operate in environments where split-second decisions and precise control are paramount for safety and success; however, the lack of real-time feedback on critical performance metrics during flight poses a significant challenge to these athletes. Traditional methods rely on post-jump data analysis, providing limited insights into flight dynamics and safety considerations. To address this, we present the development of a HUD system tailored for skydivers and wingsuiters, aiming to provide comprehensive real-time feedback and enhance flight awareness.

The HUD system is designed to integrate seamlessly into existing safety gear, offering easy access to essential flight metrics and safety information mid-flight. Leveraging cutting-edge technology such as the Raspberry Pi 4, u-blox SAM-M8Q GPS module, and MPL3115A2/BMP280 Altimeter Module, our system aims to revolutionize the way skydivers interact with their flight environment. By providing instant access to metrics such as altitude, glide ratio, GPS tracking, and more, our HUD empowers users to make informed decisions and optimize their performance in real time.

In this paper, we outline the system architecture, detailing the hardware components, custom PCB design, and application interface. We also discuss the challenges encountered during the implementation phase, including software development and hardware integration. Furthermore, we present the results of our hardware and software evaluations, highlighting the effectiveness of the HUD system in providing real-time feedback and enhancing flight safety and performance.

Overall, our research and development represents a significant step forward in the field of performance skydiving and wingsuiting, offering a practical solution to the longstanding challenge of real-time feedback during flight. By combining state of the-art technology with user-centered design principles, we aim to empower skydivers and wingsuiters with the tools they need to push the boundaries of human flight while ensuring their safety and success.

II. Related Work

Several existing solutions address the need for performance feedback and safety information for skydivers and wingsuiters, albeit with varying degrees of effectiveness and functionality. Two prominent systems in this domain are the Flysight telemetry tracker and the AON2 Project Obsidian Heads-Up Display.

Flysight Telemetry Tracker: Flysight is a telemetry tracker designed to aid performance wingsuit fliers and parachute canopy flight professionals in improving flight awareness and honing skills using trackable metrics. While Flysight offers valuable insights into flight performance, its data accessibility during flight is limited, with most information available only for post-jump analysis. The device provides real-time data on metrics such as horizontal speed, vertical speed, and glide ratio primarily through auditory cues. These cues are simple rising and falling sonic beeps, which can inform the user whether the metric they are focused on is going up or down. Additionally, the device lacks a comprehensive heads-up display interface, making it challenging for users to access critical data mid-flight.

AON2 Project Obsidian Heads-Up Display: The AON2 Project Obsidian represents another approach to providing real-time feedback for skydivers through a heads-up display worn as a set of glasses. While the Obsidian HUD offers the advantage of hands-free access to information, its functionality is limited compared to the Flysight telemetry tracker. The Obsidian HUD lacks integrated GPS tracking and performance metrics such as glide ratio, limiting its effectiveness in providing comprehensive flight feedback. The Obsidian project has been in development for a number of years now but still is not present in the market, providing an entry point for other specialized real time data display modules such as ours.

While existing solutions such as the Flysight telemetry tracker and the AON2 Project Obsidian HUD provide valuable insights into flight performance, they each have limitations in terms of data accessibility and functionality. Our proposed HUD system represents a significant advancement in this field, offering comprehensive real-time feedback and enhanced user customization to improve flight awareness, safety, and performance for skydivers and wingsuiters.

III. Hardware Components

The design of our HUD depends on both the mechanical hardware and optics suite to enable the near-eye display, as well as integration and functionality of our sensor suite for data collection. The most important piece of the mechanical hardware used in this project is the optical display housing, which facilitates proper focusing of the micro-display for our near-eye implementation. This is a critical feature of the project because improperly applied near-eye displays can cause significant eye strain and appear unfocused or blurry. Our optical setup ensures a clear image during use while reducing eye strain as much as possible.

On the data collection side, we utilized two different sensors in our final design: a u-blox SAM-M8Q GPS module to collect velocity and position telemetry and a BMP280 pressure and temperature sensor which collects pressure data to calculate altitude above sea level. The integration of the sensor suite was done with custom drivers written in C++ and built on top of the functionality of the Raspberry Pi. The most important challenge of our heads up display implementation was the optics, and we were excited to have developed a solution to the near-eye optical challenge for this project.

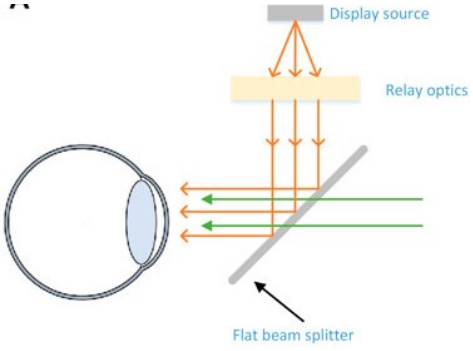

Fig. 1. Augmented Reality Beam Splitter Implementation (Adapted from [1])

A. Optical Hardware

In an effort to overcome the challenge associated with near-eye augmented reality displays, two different implementations were researched. The first implementation focused on a ferroelectric liquid crystal on silicon (FLCoS) based optical driver/lens combination [8], and used an analog driver board as an input signal to drive the display. The second implementation was a custom optical assembly with an OLED micro-display. We did initial testing with an FLCoS based implementation, but found that the image produced was not bright enough to be used as an AR display platform. After determining that the FLCoS display would not suit the needs of the project, we switched our focus to the OLED micro-display [7]. After some testing, this display proved to be bright enough to view a clear image.

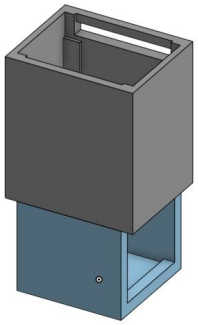

Fig. 2. Optical Display Housing 3D Model

While the display driver was the backbone of our visual heads up display, it was imperative to design an optical assembly capable of magnifying the image to a focal length that could be focused by the human eye. In order to do this, we used the optical magnifier that had been present in the FLCoS display. Since the two display sizes were comparable, we were able to piggyback off the prior optical solution without much additional adjustment. The challenge with this implementation was determining the proper distance the micro-display should sit away from the lens element to allow for full field of view of the display from the user's eye. This was accomplished with the help of 3D print prototyping.

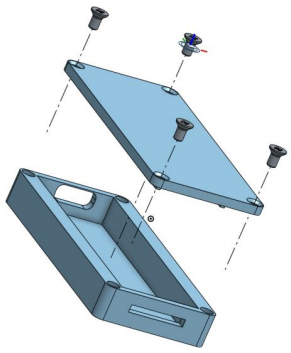

Fig. 3. Optical Driver Housing 3D Model

After some initial observations, we determined a ballpark distance we would need the display to sit from the lens and used this information to iterate through housing designs until we ended up with a distance that aligned the focal point the best. The final piece of the optical assembly is the most important in giving our user the augmented reality feeling.

Using a beam-splitter, we were able to combine the display output with the incoming environmental light, providing a combination of the two at the same time. This type of design is not novel, and has been implemented on a number of different augmented reality designs in the past (discussed further in [1] and [3]). Figures 1, 2, and 3 illustrate the implementations and optical hardware designed to achieve an augmented reality interface.

B. Electronic Hardware

The main functionality of our software lies in the development of sensor drivers for data collection. We used a number of different electronic sensors and components in our design to collect the data required for our project. There are five key hardware components that allow us to operate our heads up display. The most important component is the Raspberry Pi 4B which handles on board communication with sensors, data processing, and application display. Without the addition of the SAM-M8Q GNSS Receiver and BMP280 temperature and pressure sensor we would not be able to collect the data we need to display the necessary performance metrics for our user. After taking care of the basic functionality, we were also interested in powering our Raspberry Pi with a battery, as we designed the system to be small and portable enough to strap to the user's helmet. To enable the use of battery power, we have included a voltage boost converter [9] to drive the Pi as well as a battery charge controller [12] to handle battery recharge using wall power from a micro-usb port.

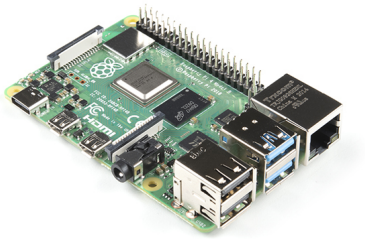

Fig. 4. Raspberry Pi 4B (Adapted from [5])

Raspberry Pi: The Raspberry Pi (see figure 4) is crucial to the implementation of our heads up display. The Pi handles all of the operating system fundamentals runs the UI platform while also allowing the use of general purpose input and output (GPIO) pins and inter-device communication. The Pi was chosen for two main reasons: it is capable of interfacing with our sensors through digital communication busses such as I2C [11] and UART and is capable of driving the micro-display via an hdmi to usb-c bridge. The display bridge finalized the connection between the image output from the Pi to the usb-c video input on the display's driver board and completed the prototype optical display for our project.

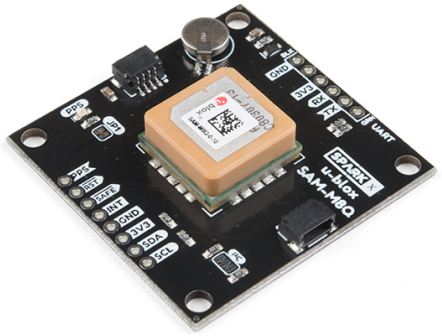

Fig. 5. u-blox SAM-M8Q (Adapted from [4])

u-blox SAM-M8Q GNSS Receiver: The u-blox SAM-M8Q is a GNSS receiver which pulls satellite data from various different GPS and GNSS satellite constellations to return position and velocity data of the sensor (see figure 5). This specific GNSS receiver was chosen due to its small form factor, I2C and UART communication interfaces, and most importantly its onboard antenna. The integrated antenna on this chip is incredibly valuable for us as it means we do not have to break out a separate antenna to receive GPS signals, reducing our platform's size. To interface with this module, we used the Pi's UART serial port to read messages from the device. The default serial communication baud rate for the receiver is 9600 baud, which is quite slow. One of the configurations we adjusted in the GPS driver includes the alteration of the message baud rate to 38400 baud. This was quick enough to enable multiple messages to be sent every second without throughput loss due to slow signal timing. We also adjusted the communication configuration on the module to send data at a 2Hz rate. These configuration changes have to be done every power cycle of the module, as it does not contain non-volatile configuration memory and is reset to the default configuration when powered down.

The most important message for our initial functionality was a GNVTG NMEA message, which contains information on course over ground and ground speed. This message structure was parsed in our driver to return the instantaneous ground speed, which was fed into and updated on our user interface.

BMP280 Pressure and Temperature Sensor: This pressure and temperature sensor is responsible for handling our altitude determinations at a nominal refresh rate. Initially, we designed drivers and implemented the MPL3115A2 altimeter into our hardware suite, but shifted gears when we realized we were limited to a 1Hz refresh rate, a rate much too slow for our needs. We pivoted to the BMP280 sensor (see figure 6) as it had high precision and up to a 200Hz data refresh rate. Writing the drivers for the BMP280 proved quite a bit

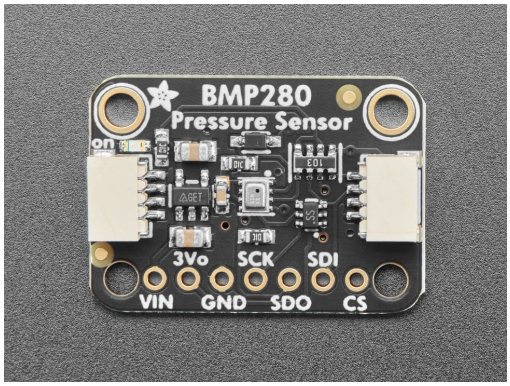

Fig. 6. BMP280 Temperature and Pressure Sensor (Adapted from [6])

more involved than the MPL, as the MPL performed internal altitude calculations based on the pressure readout, and the BMP280 did not.

Each BMP280 sensor is calibrated before shipping with the device-specific temperature and pressure calibration values stored in internal read-only registers. Calculating the pressure readout required us to write the proper message to the control register to enable high frequency data collection via the oversampling rate, storing all the calibration coefficients in our driver object, and then using the specific calibration values to perform output compensation to calculate the real pressure value. The equations for the pressure and temperature calculations done were given by the manufacturers (see figure 7).

```
// Returns temperature in DegC, resolution is 0.01 DegC. Output value of "5123" equals 51.23
DegC.
// t_fine carries fine temperature as global value
BMP280_S32_t t_fine;
BMP280_S32_t bmp280_compensate_T_int32(BMP280_S32_t adc_T)
{
    BMP280_S32_t var1, var2, T;
    var1  = ((((adc_T>>3) - ((BMP280_S32_t)dig_T1<<1))) * ((BMP280_S32_t)dig_T2)) >> 11;
    var2  = (((((adc_T>>4) - ((BMP280_S32_t)dig_T1)) * ((adc_T>>4) - ((BMP280_S32_t)dig_T1)))
>> 12) *
         ((BMP280_S32_t)dig_T3)) >> 14;
    t_fine = var1 + var2;
    T  = (t_fine * 5 + 128) >> 8;
    return T;
}
// Returns pressure in Pa as unsigned 32 bit integer in Q24.8 format (24 integer bits and 8
fractional bits).
// Output value of "24674867" represents 24674867/256 = 96386.2 Pa = 963.862 hPa
BMP280_U32_t bmp280_compensate_P_int64(BMP280_S32_t adc_P)
{
    BMP280_S64_t var1, var2, p;
    var1 = ((BMP280_S64_t)t_fine) - 128000;
    var2 = var1 * var1 * (BMP280_S64_t)dig_P6;
    var2 = var2 + ((var1*(BMP280_S64_t)dig_P5)<<17);
    var2 = var2 + (((BMP280_S64_t)dig_P4)<<35);
    var1 = ((var1 * var1 * (BMP280_S64_t)dig_P3)>>8) + ((var1 * (BMP280_S64_t)dig_P2)<<12);
    var1 = (((((BMP280_S64_t)1)<<47)+var1))*((BMP280_S64_t)dig_P1)>>33;
    if (var1 == 0)
    {
        return 0; // avoid exception caused by division by zero
    }
    p = 1048576-adc_P;
    p = (((p<<31)-var2)*3125)/var1;
    var1 = (((BMP280_S64_t)dig_P9) * (p>>13) * (p>>13)) >> 25;
    var2 = (((BMP280_S64_t)dig_P8) * p) >> 19;
    p = ((p + var1 + var2) >> 8) + (((BMP280_S64_t)dig_P7)<<4);
    return (BMP280_U32_t)p;
```

Fig. 7. Temperature and Pressure Output Compensation Algorithms (Adapted from [6])

Battery Management: The final components in our hardware design include the battery charge module as well as the voltage converter. These two components allow us to facilitate module charging without having to remove the battery from the housing, as well as convert the 3.7V lithium ion battery voltage up to 5V in order to power the Pi. Unfortunately, due to a lack of stock with our PCB manufacturer we were unable to realize a printed circuit board to implement the functionality of these battery management strategies. For our minimum viable product we opted to use off the shelf prefabricated versions of these modules to demonstrate proper functionality.

IV. Custom PCB Design

A. Initial Approach

A significant portion of this project is meeting spatial requirements for an ideal user experience. Our minimum viable product (MVP) uses the relevant modules wired together to demonstrate functionality of the system, however this design is not at all conducive to a real-time implementation. The MVP design is bulky and space inefficient, and our goal is to create a final product that is compact, light, and easy to mount directly onto a helmet for wingsuiting or skydiving purposes. To solve the problem of this bulky initial design, we decided to adapt the schematics of each component (minus the GPS module due to ground plane spatial requirements) into a single printed circuit board.

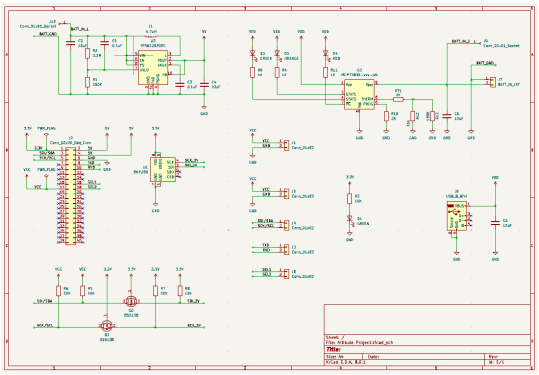

Fig. 8. Custom PCB Schematic (First Iteration)

Our ideal PCB design combines the altimeter module, the battery charger, and the voltage converter (to convert the 3.7V output to a 5V power supply for the Raspberry Pi) into a single PCB component capable of mounting directly to the Raspberry Pi via a 2x20 pin connector. Additionally, we breakout 3.3V lines, the I^2C pins, and the UART pins to JST connectors for external access which provides a modular and flexible board for further iteration and testing.

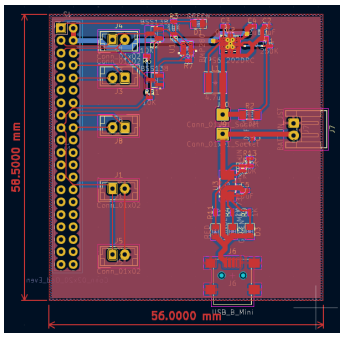

Fig. 9. Custom PCB Layout (First Iteration)

Despite finalizing the layout for this design (see figures 8, 9, and 10 for the PCB schematic, layout, and 3D view respectively), we realized that further tweaks may need to be made as we iterate through various modules to achieve better implementations (for example, we switched from the MPL3115A2 barometric sensor to the BMP280 altimeter mid-design due to the data rate limitations of the MPL3115A2). This, along with the fact that some of our chosen components were unavailable to order within a short time-frame persuaded us to hold off on ordering the PCB. To accommodate this decision we pivoted to a simpler PCB implementation for modeling and iterative purposes.

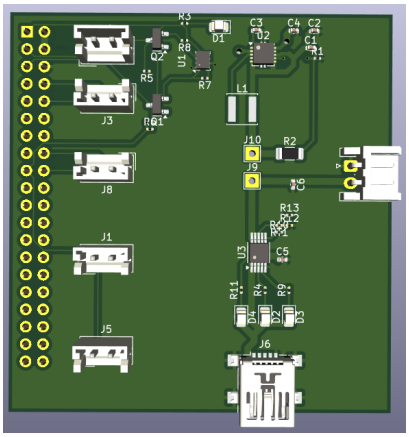

Fig. 10. Custom PCB 3D View (First Iteration)

B. Small PCB Implementation

The first iteration of our design allowed us to familiarize ourselves with the KiCad software [10]. It was straightforward to reduce our initial design to a "Small PCB" implementation, which included the JST breakouts mentioned above, the BMP280 altimeter module, and the pin connector, but eliminated the voltage converter and battery charger schematics entirely (see figures 11, 12, and 13 for the Small PCB schematic, layout, and 3D view respectively). After significant rewiring of the design for a highly compact layout, we were able to finalize, verify, and order our Small PCB design from JLCPCB.

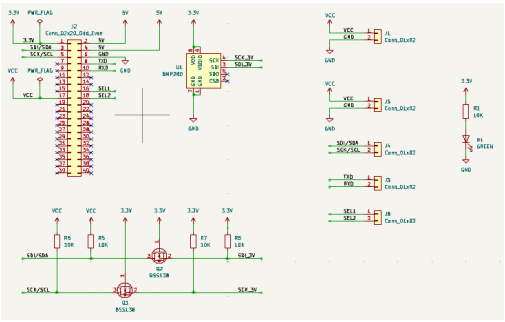

Fig. 11. Custom PCB Schematic (Small PCB Adaptation)

This is an ongoing project, and the Small PCB design will allow us to test our pin connector orientation and begin designing housings for the heads-up display hardware. A long-term goal for this project is to implement the GPS module into the PCB design as well such that all components can be housed together in a space-efficient manner. We are still in the process of iterating through design ideas for how we might incorporate the GPS block but eventually hope to achieve a highly compact physical housing for all components.

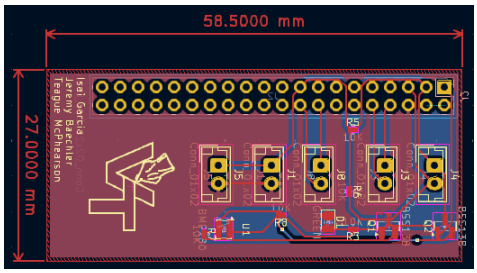

Fig. 12. Custom PCB Layout (Small PCB Adaptation)

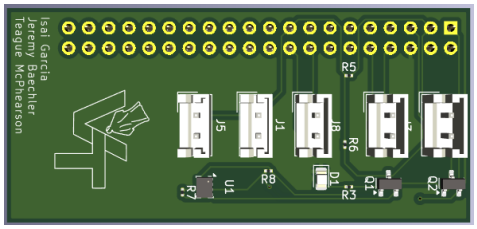

Fig. 13. Custom PCB 3D View (Small PCB Adaptation)

V. Software Implementation

The software implementation of our HUD system primarily focuses on developing a user-friendly interface for accessing and displaying real-time flight data. Leveraging the power of the Qt framework [2] for front end app design, we aimed to create a seamless user experience while ensuring efficient data processing and visualization. Below, we outline the key steps and challenges encountered during the software development phase.

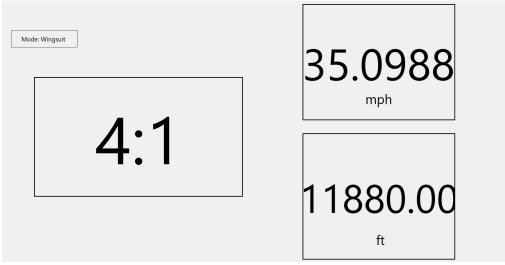

Fig. 14. User Interface Design

User Interface Development: We began by designing the user interface (UI) to accommodate various flight metrics including speed, altitude relative to ground and glide ratio (see figure 14). Qt Creator provided a versatile platform for

creating customizable widgets and layouts, allowing us to tailor the HUD display according to user preferences. We implemented features such as dynamic data visualization to enhance usability.

Cross-Compilation Challenges: Initially, we explored cross-compilation techniques to deploy the software on the Raspberry Pi. However, we encountered challenges in setting up the host environment and configuring dependencies. As a workaround, we opted to flash a Raspberry Pi OS image with Boot2Qt. This platform enables remote cross-compilation on Windows while enabling easy upload of the new Linux image wirelessly. Boot2Qt enabled access to the necessary I2C ports for reading data from the altimeter module. After completing driver development based on our Boot2Qt, we realized that there was no access given for the UART serial port. After discovering this lack of functionality in the Boot2Qt deployment platform, we pivoted again to writing and compiling the application software natively in the Qt Creator IDE on the Pi. This implementation is not suitable for long term development due to its inability to compile more complex user interfaces we would like to implement in the future but suited our needs while developing our minimum viable product.

Driver Development: With access to the ports secured, we proceeded to develop drivers for interfacing with the altimeter and GPS modules. We implemented protocols for reading and parsing data streams from these modules, ensuring accurate retrieval of altitude, ground speed, and other relevant metrics. We tested the initial functionality of our drivers by SSH-ing to the Raspberry Pi. We wrote main scripts to print out GPS and altimeter values to the terminal for validation. Once the initial testing of these drivers were completed on their own, we integrated these drivers into the Qt UI design.

Integration and Testing: Once the drivers were developed, we integrated them into the main HUD software framework and conducted testing to validate functionality and performance. We utilized simulation values and NMEA sentences as examples while monitoring the HUD display for real-time updates and accuracy of displayed metrics while we waited for the hardware drivers to be finalized. Once the drivers were validated, we wrote the data collected from the sensor drivers to the display at a 1Hz refresh rate. This update rate is entirely configurable so we will be able to speed up or slow this rate down depending on the desired implementation in the future. To test the final configuration of our project, we took the electronics on a drive and verified total functionality of our UI and sensor integration.

VI. Experiments and Results

To demonstrate the functionality and effectiveness of our HUD system, we conducted experiments to validate the accuracy and reliability of the real-time flight data obtained from the device. Initially, we intended to showcase the successful display of the Widget Application on the OLED micro display (see fig. 15). However, a sudden malfunction in the display prevented us from proceeding with this demonstration. As a solution, we redirected our focus towards validating the data output from the device by recording and analyzing the flight metrics during a controlled test scenario.

Fig. 15. OLED Micro Display (Adapted from [7])

Experiment Design: We conducted a test drive up Horsetooth Reservoir, during which we connected the battery and voltage divider module to the Raspberry Pi to power the device, along with the altimeter and GPS module to collect data on simulated flight conditions. We captured real-time altitude and speed data, which were logged into a CSV file for analysis within the Pi. The vehicle's speedometer was used as a reference for validating the accuracy of speed measurements recorded by the HUD device. We also compared the expected altitude gain from google maps for our specific drive profile and found that the altimeter did a good job measuring the difference in altitude between our start and end points. Our experimentation aimed to assess the device's performance in accurately capturing flight metrics under controlled conditions, and after our testing we can say the data captures the simulated conditions well.

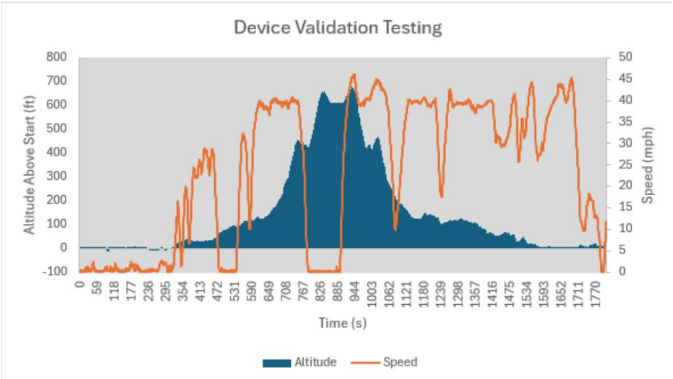

Fig. 16. Experimental Validation Data

Data Analysis: The CSV file of our test data was formatted into three columns: Time (s), Altitude (ft), and Speed (mph). We plotted the altitude and speed data against time to visualize the device's performance during the test drive. The altitude data is represented by blue lines, while the speed data is depicted in orange. The x-axis represented time in seconds,

while the y-axis had two scales for altitude (ft) and speed (mph) (see fig. 16).

Results: The device functioned as intended during the test drive, capturing real-time altitude and speed measurements. The plotted data showed a consistent trend in altitude and speed changes over time. It is important to mention that there were some points in which the altitude drops below 0, or ground level. This makes sense as we could very well have spend some time at an elevation slightly lower than the point where the initial ground altitude measurement was taken. During our testing, the overall performance of the system was satisfactory, with the recorded data aligning closely with the expected flight metrics.

Validation Process: To validate the accuracy of the speed measurements recorded by the Altimeter and GPS modules, we compared them with the readings from the vehicle's speedometer. We compared the time stamps from both sources and cross-referenced the corresponding speed values. This validation process helped ensure the reliability and integrity of the data, despite the challenges encountered during the test.

VII. Conclusion

Our project aimed to address the need for real-time feedback and performance metrics for performance skydivers and wingsuiters through the development of a near-eye HUD system. Throughout the course of this project, we encountered various challenges and setbacks related to both hardware and software, but despite these obstacles we succeeded in developing a functional product and emerged with valuable insights and lessons learned. Our project required seamless integration of hardware and software components, which taught us valuable lessons in adaptability and versatility. Through collaborative efforts, we were able to implement and learn from different perspectives, enriching our understanding of both hardware and software aspects of the project. Despite the unexpected malfunction of the display, we learned to adapt and resolve issues, demonstrating our ability to overcome obstacles and find innovative solutions. The testing phase provided a steep learning curve in troubleshooting and validation, enhancing our skills in rigorous experimentation and data analysis. Looking towards future implementations, we envision adding features such as glide ratio calculations to provide more comprehensive flight feedback. Developing a mobile application for accessing and analyzing historical flight data would enhance user experience and data management capabilities. Integrating a custom PCB would further optimize the design for compactness and efficiency. Further exploration of Boot2Qt functionality could provide insights into alternative deployment methods and enhance system performance and UI functionality. Additionally, exploring options for enhancing the physical appearance of the HUD system could improve user appeal and marketability. We enjoyed the challenges involved with this project, and are looking forward to continue development on this in the future.

VIII. Video Links

We have included the presentation and demo video links here in this paper for reference.

Presentation: https://youtu.be/gTYhT-rIvSc

Demo: https://youtu.be/XWEDh5C_IR4

References

[1] Xia, X., Guan, F. Y., Cai, Y., Magnenat Thalmann, N. (2022, March 14). Challenges and Advancements for AR Optical See-Through Near-Eye Displays: A Review. Frontiers. https://www.frontiersin.org/articles/10.3389/frvir.2022.838237/full

[2] Ame. (2023, February 26). How to install qt and designer or creator on the PI - Raspberry Pi Forums. Raspberry Pi. https://forums.raspberrypi.com/viewtopic.php?t=348064

[3] Kim, M., Choi, S. H., Park, K.-B., Lee, J. Y. (2019, August 4). User Interactions for Augmented Reality Smart Glasses: A Comparative Evaluation of Visual Contexts and Interaction Gestures. MDPI. https://www.mdpi.com/2076-3417/9/15/3171

[4] Sparkfun. (2017). GPS Breakout Ublox SAM-M8Q (Qwiic). Sparkfun Start Something. Sparkfun. Retrieved May 3, 2024, from https://www.sparkfun.com/products/retired/15106.

[5] PiShop. (2019). Raspberry Pi 4 Model B/4GB. PiShop.us. PiShop.us. Retrieved May 3, 2024, from https://www.pishop.us/product/raspberry-pi-4-model-b-4gb/.

[6] Adafruit. (2015). Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor - STEMMA QT. Adafruit. Adafruit. Retrieved May 3, 2024, from https://www.adafruit.com/product/2651.

[7] Display Anox. (n.d.). 0.39 inch Micro OLED For AR Type-c Board. Display Anox. Display Anox. Retrieved May 3, 2024, from https://www.panoxdisplay.com/micro-display/0-39-inch-micro-amoled-1920x1080-mipi.html.

[8] AliExpress. (n.d.). Ferroelectric Liquid Crystal Micro Display Module Night Vision Thermal Imaging Display 960x540 FPV Display. AliExpress. Retrieved May 3, 2024, from https://www.aliexpress.us/item/3256803182439424.html?

[9] Sparkfun. (2011). LiPower - Boost Converter. Sparkfun Start Something . Sparkfun . Retrieved May 3, 2024, from https://www.sparkfun.com/products/10255.

[10] Getting started in KiCad: 6.0: English: Documentation. KiCad. (n.d.).https://docs.kicad.org/6.0/en/gettingstartedinkicad

[11] Kickstart Embedded. (2021, October 29). [eng] raspberry pi! EP:6 - I2C usage - C (i2cdev). YouTube. https://www.youtube.com/watch?v=znaxhjvDS-Y&t=2s

[12] Adafruit. (n.d.). USB LiIon/LiPoly charger - v1.2. Adafruit. Adafruit. Retrieved May 3, 2024, from https://www.adafruit.com/product/259.