Final Paper

Skier's Helpful Information Tracker – Team 52

Sam Knight, Jack Bay, Ryder Heit

Electrical and Computer Engineering

University of Illinois at Urbana-Champaign

May 2024

Contents

1 Introduction

Our project is a tracker designed to assist waterskiers in their goal of chasing the perfect slalom pass. Waterski coaching can be expensive and hard to find, especially given the nicheness of the sport as well as the financial barrier to entry. Our tracker provides a way to minimize this barrier to entry by making coaching something that can be provided at a low cost. Slalom is also a very precise sport, with the skier moving very quickly around 6 buoys, and quantitative data is currently impossible to get. Our tracker aims to change that.

The tracker provides two primary functions, collection and analysis. Data is taken directly on the ski by the device during a run, with sensors running and collecting data that is then saved to hard storage, an SD card. A skier can record multiple passes for analysis later. The analysis is done on a separate computer and shows a visualization of all the data taken to allow the skier to study their runs and improve.

Our well-rounded team of engineers each took responsibility for distinct tasks, allowing each member to contribute their top skills to the project and create a fleshed-out product with expertise applied in many areas. Jack Bay, an electrical engineer, was responsible for the power subsystem and PCB design. Ryder Heit, a computer engineer, was responsible for firmware creation and PCB design. Sam Knight, a computer engineer, was responsible for data processing and software design.

Together, all members of our team have combined their diverse specialties to create the Skier's Helpful Information Tracker. In this paper, we will talk about the successes, failures, challenges, and results over the semester and the design process.

2 Problem Analysis

2.1 High Level Design Goals

When formulating the initial design, we chose initial design goals to serve as the target criteria for the project's completion. Our design goals are as follows. Waterproofness: the device must be waterproof up to a submersion depth of 10ft. As the device is designed to be used in the lake, we need it to be waterproof up to the maximum depth we expect it to reach while attached to a waterski. Accuracy: the accuracy of the device must be within at least 3m for GPS systems, and within %5 for all other sensors. Without an acceptable accuracy requirement, we cannot say our device provides useful feedback. Multiple passes: the device must be able to record 10 passes consecutively without issues. The device wouldn't be useful if the skier needed to charge and reset it between each pass.

2.2 Tolerance Analysis

The team conducted tolerance analysis for the ski tracker after receiving feedback at the design review to analyze this component. We needed to make sure that we designed a board that can withstand the associated g-force of skiing, both staying attached to the ski and steady within the box. We did tolerance analysis work to determine the maximum g-force experience by a skier during a pass and determined that while the maximum g's may go as high as 6, this is only an instantaneous value and that the sustained g-force on the device for a meaningful period will only be 3.5g's. This was well within the force rating on the Velcro system we chose, but we added redundancy systems as well. The device had no problems staying attached to the ski during a run.

3 Project Design: Hardware

3.1 Design Overview

The main hardware components for the device are a microcontroller, GPS, accelerometer, gyroscope, IR receiver, SD card reader/writer and battery. The GPS, accelerometer, and gyroscope provide us with the location, speed, and orientation data to be stored on the SD card and analyzed on the external computer. The SD card reader/writer is the component that allows us to write this data onto a physical SD card after the microcontroller formats the input data to a usable format in a .CSV. The IR receiver changes the state of operation when it picks up an IR signal. The battery obviously supplied the entire board with a consistent 3.3V with a voltage regulator.

3.2 Considerations

KiCAD was used to lay out schematics for the PCB. The main worry when putting the board together was maintaining a small size without crowding components too closely together. This was both so the board would fit in the waterproof box that will be attached to the ski and so the microcontroller will be able to communicate with the accelerometer, gyroscope, and GPS chips efficiently.

Another large consideration was creating a 3.3V plate on the top of the board and a GND plate on the bottom. This decision was made based off the CAD assignment from earlier in the course where we created two GND plates, one above and one below, to make it easier to give components access to a GND port without crowding the board with wires. We added the 3.3V plate instead of two GND plates because all our chips operate at 3.3V, so it makes it much easier to connect them to power and GND if these sources are readily available at any spot on the board.

A few changes were made during the design process. This includes changing our MCU to an STM32 for higher processing power. With this change, we had to change the design of the board due to it having a different pin out from the previous MCU. Also, it was discovered that the pull-up resistors attached to the SD card reader blocked the SD card from being inserted and removed, so we decided to remove them from the design. Images of the final schematic and layout for the PCB are provided in the following figures.

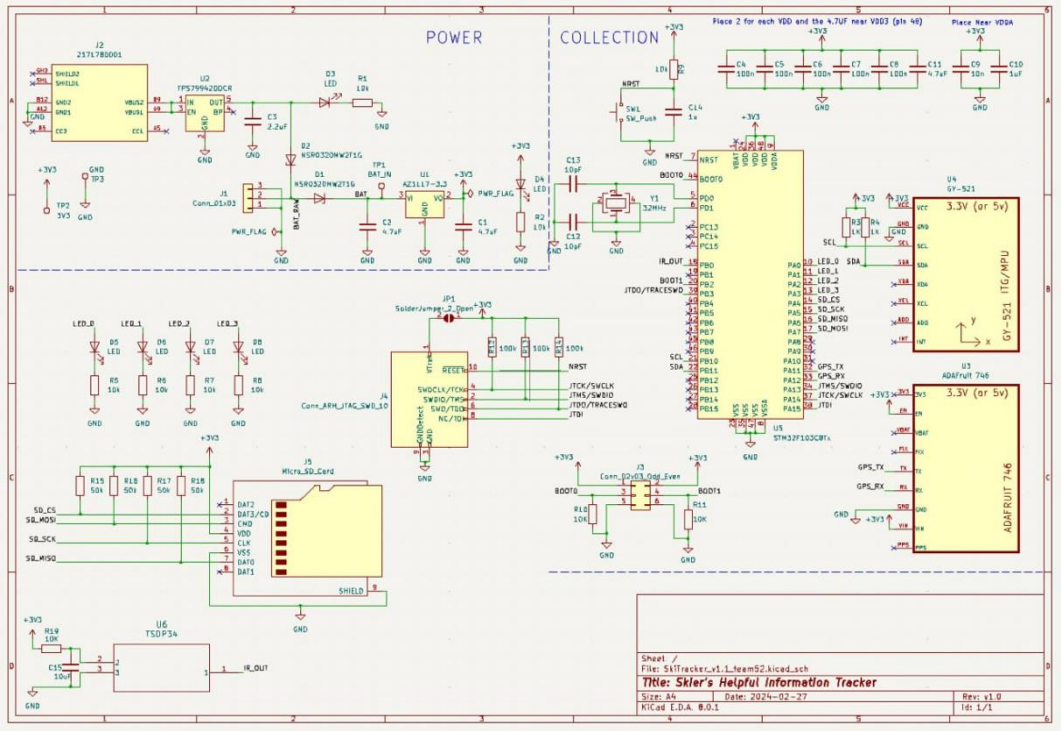

Figure: PCB Schematic

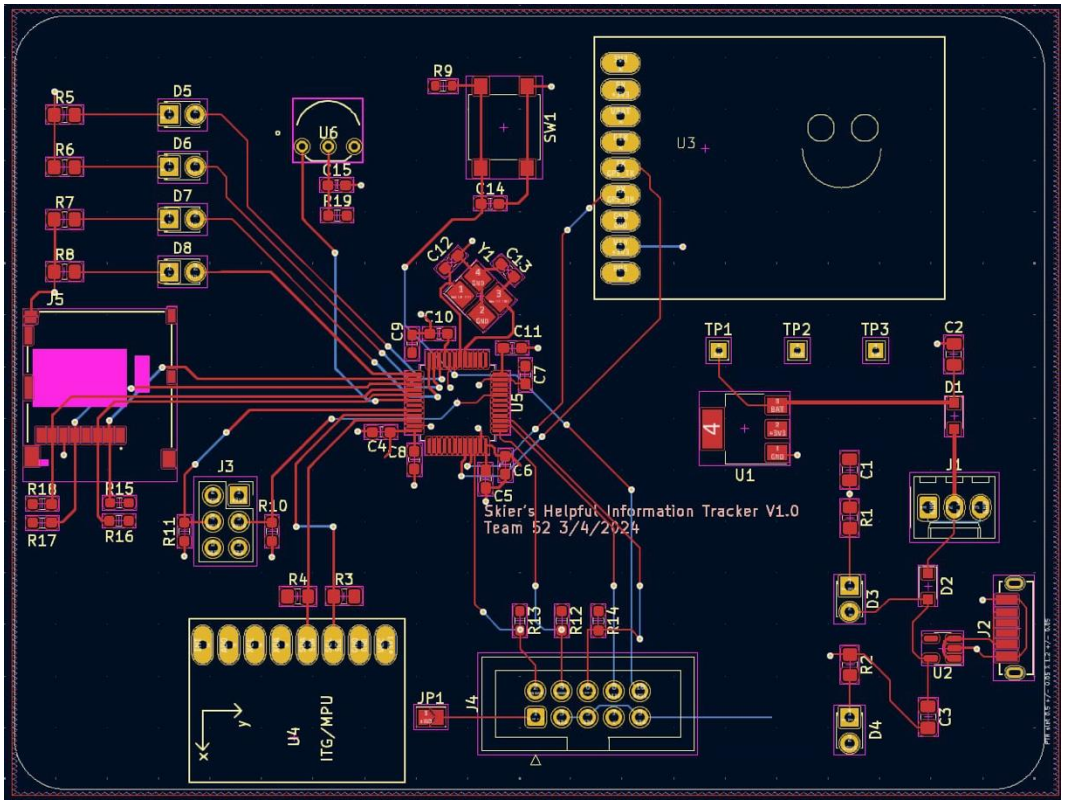

Figure: PCB Layout

3.3 Design Justification

Part	Reason Picked	Function
6V Battery	We originally picked out a 3.7V battery, but the 3.3V regulator's overhead voltage was higher than anticipated, so we jumped to a 6V battery.	The battery provides each chip on the PCB with power for enough time to successfully record data from several passes.
STM32F103 MCU	Originally picking the ATMEGA MCU, we switched over to the STM32 because it was both easier to obtain and has a higher processing power.	The MCU controls all our peripherals (gyroscope, GPS, accelerometer) and records data to the SD card.
ADAFRUIT 746 (GPS)	This GPS has a sufficient polling rate (~10MHz) and can operate at 3.3V like the rest of our peripherals.	The GPS records the skier's coordinates as they run through passes, giving an actual location to link gyroscope and accelerometer data to.
MPU-6050(Gyroscope & Accelerometer)	This chip has both an accelerometer and gyroscope on it, which was very advantageous since these are both sensors we wanted to use.	The accelerometer measures the skier's speed while the gyroscope measures angles of the skier with respect to the X, Y, and Z axes.
3.3V Regulator	This regulator was both small and cost effective.	This provides each component in the collection and storage subsystems with the 3.3V they require.
4.2V Regulator	This regulator was both small and cost effective.	This provides the battery with a maximum of 4.2V when charging off the USB-C port.
TSOP34 (IR Receiver)	This IR receiver was low-cost and can pick up the IR frequency of the remote we selected.	This IR receiver can detect IR light from a remote and close a circuit when that occurs. This is how we will control the state of operation when the box is sealed up.
USB-C Port	This port was cost effective and designed just for charging. USB-C is also found everywhere nowadays.	This port was selected to allow us to charge the battery without removing it from the device.
Buttons	These parts are cheap and easy to control.	One is a RESET button for the entire system while the other replaces our IR receiver in controlling the state of the device.
STM32F103	Originally picked an ATMEGA chip for our MCU. Switched to this for more efficiency and more documentation available	This is the MCU of the device. It provides all the logic and control for all the devices. It reads from the sensors and writes to the SD Card.

3.4 Hardware Conclusions

In conclusion, the hardware portion of this project was a bit more difficult than it was originally scoped out to me. This was mainly due to the changes that needed to be made when parts were swapped out and the limited time we had to make those changes. The design of the PCB is satisfactory but could still use a few tweaks to better fit the parts we selected. If given the opportunity in the future, we would choose a different IR receiver that would work with the polling rate of our MCU. This was a large oversight in our design process that inhibits a very critical part of the device's operation.

4 Project Design: Firmware

4.1 Design overview

The firmware for our device was written onto an STM32F103 Microcontroller. We used the HAL library and STM32CubeIDE for physical programming. The library allowed us to write low-level C code that could directly communicate with the sensors. The firmware was fairly simple for this device as it mainly acted as a go-between for the sensors and the sd card. It also controlled device state.

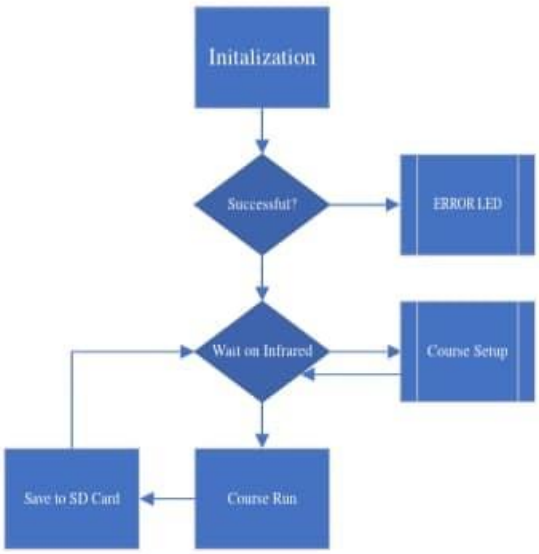

Figure: General Firmware Flowchart

4.2 Firmware Design

The first choice to be made before writing the firmware was what language to program it in. The two main choices were using the Arduino Library for the STM32 or the HAL Library

with STM32CubeIDE. Arduino is simpler for many purposes and is usually easier to start with; however, we chose to use the HAL Library because it provided more flexibility and control over the MCU. Throughout all states of operation, we use different LED states to give the skier information on skiing.

The first piece of the firmware was getting it to properly read the sensors. For the Gyroscope and Accelerometer, we have an MPU6050 chip that is connected to the MCU using I2C. The first step for the MPU6050 is turning it on and calibrating it. Register 0x75 is the "Who Am I" Register which allows the firmware to verify that the device is operating correctly. We then write to register 0x6B for power management as well as setting the modes for the MPU [1]. To read from the device, registers 0x3B to 0x48 contain all the gyroscope and accelerometer data. Once the device is initialized, we run a function that takes the running average of the gyroscope and accelerometer data that we save as offsets so future polls of the MPU will be correctly zeroed. Finally, the main while loop of our function constantly polls the MPU so that we get current data.

The next sensor to work with was the GPS. The GPS works as a UART interrupt device. That means, the code will be interrupted whenever the GPS has a new message. These messages are in the NMEA format, and we wrote simple code to decode these to get the helpful values, such as latitude and longitude [2]. This was stored in static variables so that it would still allow us to provide the SD Card with a constant stream of data.

Once I gathered this data, I had to format it and write it to the SD Card. To write to an SD Card from the MCU we used the SPI protocol and the Fat-FS. We had to create a custom driver for the Fat-FS file system using the SPI code of HAL. We first had to mount the SD card to the MCU file system, then write a csv file with the correct information. The first line of the CSV file is the labels for each column.

ms	GPStime	lat	lon	satelites	course	speed	yaw	pitch	roll
100	234451.00	40.09514236	-88.23922729	6	0.00	0.00	12.045801	2.893130	0.076336
200	234451.00	40.09514236	-88.23922729	6	0.00	0.00	12.244275	1.221374	-0.656489

Figure: Example CSV Table Values

We initially wanted to use an IR Remote to control the entire system, but we struggled to get it to work in time. Instead, we used a button to control the state of the system. The button acts as a recording latch. When you press the button, the system enters a "recording" state in which every 100ms, it writes the current values to the SD card file. When the button is pressed again, it closes the file and unmounts the file system.

4.3 Firmware Conclusions

With the firmware, I think that it was mostly successful. If we had more time, I would like to have gotten the IR receiver working. As far as other things to change, I would like to potentially do some on chip smoothing for the GPS to make the processing easier. Also, it would be nice to continue to add more useful data to the csv file. Finally, I would have liked to improve the status, such as adding an OLED display that has more information.

5 Project Design: Software

5.1 Design Overview

The software design for the project is somewhat demanding. At a high level, we need to accomplish a few things. First, we need to extrapolate a few pieces of information from the location data. We need to read the calibration data and use it to calculate the position of each body, then request satellite images to draw on to represent the skier path. This requires a lot of math using GPS coordinates, which is difficult due to the curvature of the Earth needing to be plotted on flat maps. Second, we need to extract and smooth the speed and orientation data from the gyro and accelerometer. The data is initially very noisy, so digital signal processing will need to be used to get useful info from the data. Lastly, we will need to create a tool to visualize this data. The goal of the project is to view all the data at the same time, so creating a tool to visualize will be imperative to the overall success of getting useful information for the skier from the data we took!

5.2 Design Considerations

To accomplish the above goals, we need to choose a programming language to support the tasks we are doing. For creating static map images, I chose to use the Google Maps Static Maps API. While panning and map interactivity would be a neat feature, embedded maps must be used in a webpage, and as this project is not web based, I chose the Static Maps API. This makes the live map animation simpler to construct. For processing sensor data and creating animations, I chose to use Python3. Python is advantageous for this for a few reasons. Firstly, the availability of easy-to-use libraries for math and data processing makes Python a strong choice, as well as the ease of creating animations through data streams. Secondly, the bundled Tk GUI framework was chosen to create the tool that will allow us to visualize the data. Lastly, Python has excellent integration with the tool FFmpeg, which is used for video encoding and output. I had considered using C++ for this project, but the high-level availability of the necessary tools in Python made it a cumbersome choice in comparison. Speed is not an important goal for our project, so a fast C++ implementation was not necessary.

5.3 Static Maps API

As the member in charge of the processing subsystem I was responsible for enrolling for and using the Google Maps Static Maps API. Static Maps are not particularly flexible, returning simply a still image, but I decided that most of the processing can be done using Python3 image processing libraries such as Pillow [6] and Matplotlib [7]. Using the satellite image, we need to be able to draw the skier's path, as well as relate some information about the path back to the skier using the visualization. I decided that this would be far

easier using Python to do the brunt of the processing as opposed to making complicated API calls to Google Maps.

5.4 Python Analysis

The bulk of the analysis of the data was done using Python3 to parse and extract information from the data. The CSV provided by the device contains many data fields, and we can use the python bundled CSV reader to get this data from the storage device. After prompting the user for the location of the data they wish to use, the CSV reader code reads the data as follows.

After reading the data into our own data structure, we can begin analysis. The first step is to calculate a series of map images. The coordinate points of the gate bouys are stored at the head of the CSV file, so the first thing we do is take those gate coordinates and use them to calculate the positions of the remaining balls. The ski course is always the same layout and dimensions, so this is achievable with only these two points of information. The difficulty involved in doing distance calculations and other complicated math was abstracted by converting the coordinates to XY coordinates relative to the image we already have. If we know the real size of the area the map captures, and we know where the bounds of our image line up with real coordinates, we can safely convert between XY and GPS coordinates. After computing all the course information, we create a map image of the course by requesting the API for an image of the lake, and then use the Pillow library to draw the computed points of interest onto the map. Because of the work done to abstract the conversion between GPS and XY, this is a simple task that is easily accomplished with the Pillow draw functionality. To create the live animation, we simply move the point representing the skier around the image, then stitch those images into an animation later. I will talk about animation creation in a later section.

For the other stats visible in the live animation, we can take a much simpler approach. Most of the data we are looking for is stored directly in the CSV file, such as pitch, yaw, roll, and timescale. All that is necessary to move on to the animation step is to read these stats from the CSV into arrays that we can later direct at an animation function. The speed requires some calculation, however. The GPS chip chosen did not return speed and course values correctly, so for the final product we needed to use the GPS readings to calculate the speed. This is trivial, all we need to do is to calculate the change in distance between each data point, and then divide by the change in time to get the speed over the course of those two data points being taken. We also used interpolation to fill in values where the GPS chip fell behind and failed to take data points. We found that this was a very accurate method of smoothing the data without making large assumptions or altering the overall content of the data itself. While the initial plan was to Kalman Filter our data, we found in

the end that this was unnecessary, and some simple DSP would clean and smooth our data very effectively.

5.5 Data Processing

The Data gathered from the board was initially very noisy. We noticed a lot of problems from jittering of the device and the ground under the bicycle was not smooth. The first issue was a high amount of high frequency noise in the Gyroscope data. To fix this, we initially tried to use a small, naive, box-car window that works as a moving average on the system. This got out some of the noise but there was still a lot present. To fix this, we applied a larger Hamming Window that got out the noise but kept the general trends of the data.

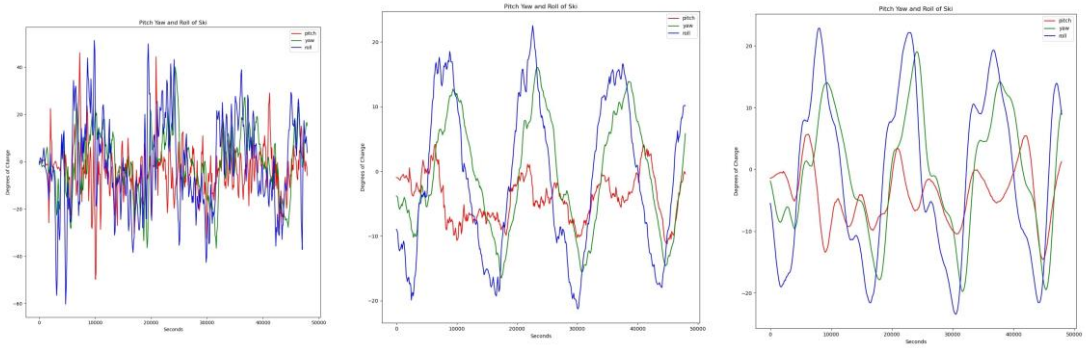

Figure: Progression of smoothing Gyroscope data

For the speed data, we noticed a bunch of spikes in the data. These spikes are outliers in the data. To fix this, we first thought that a median filter would work to reduce the outliers. This worked to a point, but the data still had some noise problems. To fix this, we applied a Kaiser window that further reduced the data.

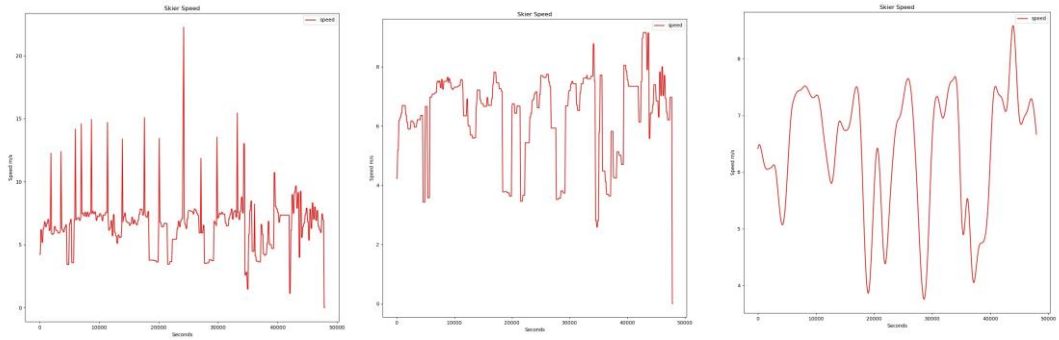

Figure: Progression of Smoothing Speed Data, not the change of axis scale

5.6 Animation Creation

Creating an animation is, in essence, creating a video file from a series of images. While the naive approach is to do just that – create a gif container of images from our precomputed image values from above, this is not a sufficient solution for our GUI to be able to smoothly play and seek later. I chose FFmpeg to be the video writer for our animations throughout the project. It is a highly flexible video encoder that can convert between all formats seamlessly and with many options. With this flexibility, we can create low-volume and efficient encodings with the options we require for the GUI customized to our liking. I chose AVC to ensure the videos are highly compatible and can be played on any consumer computer setup. The container was chosen to be MP4, again for high compatibility.

OpenCV provides a framework for a videocapture object [3] which creates a video object we can load our images into. For our other stats, it is unnecessary to compute a series of images beforehand as the Matplotlib library provides us with a method for creating animations already. It supports blitting, which is the technique that reuses the background graph and only draws the part of the animation that updates on each frame. So, to animate our remaining stats, we simply create animation functions to pass into the animation methods provided by the library and load our arrays into the calls. We again leverage FFmpeg to encode our videos and output them as MP4.

5.7 GUI

To create the GUI for visualizing the data, I chose to use Tk as I have experience writing Tk already, and the library comes bundled with many Python3 installations. As well as this, our goal is relatively simple – display precomputed animation files in sync. With little actual GUI based functionality, other popular frameworks such as QT or GTK are overkill for this task. The Tk GUI is simple – we leverage the TkVideoPlayer library to create a video player instance for each video we want to play, which is the live map, speed data, and orientation data. Then, we associate a lead video stream, in our case the live map, to the seek bar and various events provided by the class construction. Because all our videos are the same length, we have no problem using one video as the master of all streams. Whenever a video player event is generated, we apply the action to all video streams. This gives the functionality of playing all streams in sync! Also included in the GUI side of the tool are interactive file dialogs to ease user experience in using the tool, prompting the user to input the data they wish to process at each invocation of the tools.

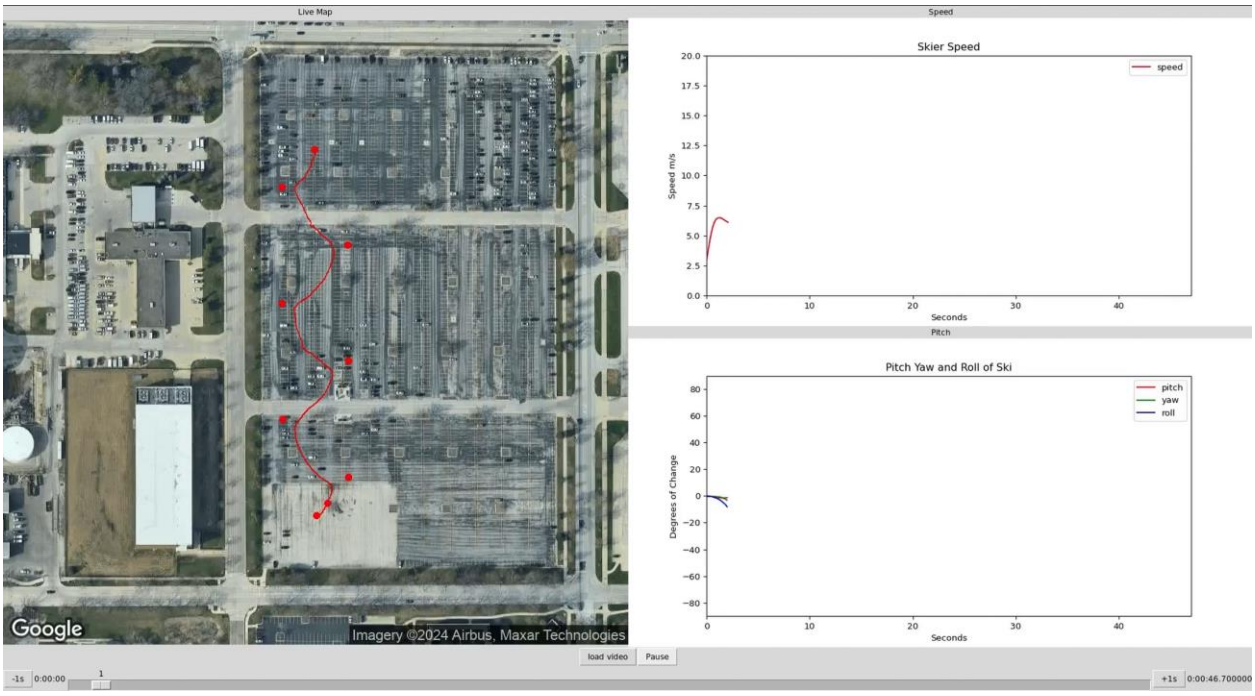

Figure: GUI serving to visualize the data

5.8 Verification

To verify the software portion of the design, we revisit the original verification goals of the project that we wanted to accomplish in the processing subsystem. The software system needed to process and filter the data such that we extract a complete and useful picture from even very noisy data. This was a success, with the filtering being highly effective at extracting features from the very noisy IMU data. The second goal was to create a live map of the skiing data using satellite imagery, which was also a success. The final goal was to display the processed data side-by-side in a visualization tool, which was accomplished via the Tk GUI. The processing subsystem meets all the goals we set for it.

5.9 Software Conclusions

Overall, I am very satisfied with the software component of this project. All the goals I had for the design were met, and no major changes were made that added significant complexity to the design. The data generation tool is flexible and user friendly, and the visualizer tool very neatly ties it all together. The only things I would change about the software design are minor implementation differences, such as re-using the same map instead of requesting many, or tweaking data smoothing to more intelligently remove noise from the data using AI or other advanced techniques.

6 Cost and Schedule

6.1 Cost

Component	Price
Gy-521 MPU-6050 (Gyro + Accelerometer)	$13.71
SD Card	$10
SD Card Reader	$9
TSOP93438 (IR Receiver)	$1.06
ADAFRUIT746 GPS	$12.99
Four AA Batteries	$6.79
GPS	$31.99
STM32F103 (MPU)	$6.49
Clear Waterproof Case	$30.00
USB-C Charging Port	$7.35
3.3V Regulator	$0.99
4.2V Regulator	$0.99
$88,000 salary for 3 people over 12 weeks	$60,900.00
Total	**$61,031.37**

Table: Cost Analysis

6.2 Schedule

Week	Everyone	Jack	Ryder	Sam
Week 1 of 02/26	Design Check	Board Schematic and finish part selection	Help with Board Schematic & Bread Board Testing	Start learning Maps API and create mock data for testing
Week 2 of 03/04	Finalize Design	Finish Layout	Help with Layout and Start Firmware	Use Maps API on mock data
Week 3 of 03/11	Spring Break	Spring Break	Spring Break	Spring Break

Week 4 of 03/18	Discuss hardware with machine shop	Solder and test board, revise board design	Work on simple firmware on board and help revise board	Begin Kalman Filtering on Mock Data
Week 5 of 03/25	Individual progress reports	Help with firmware, revise board again	Work on updating firmware to link all subsystems	Using Mock data, finalize software workflow
Week 6 of 04/01	Finalize	Finalize Board design and one last board run	Finalize Firmware for on board design	Apply Software to real data
Week 7 of 04/08	Finalize	Hardware	Firmware	Software
Week 8 of 04/15	Mock	Demo	Mock	Demo
Week 9 of 04/22	Final	Demo	Final	Demo
Week 10 of 04/29	Final	Presentation	Final	Presentation

Table: Team Project Schedule

7 Conclusions

7.1 Accomplishments

Our biggest accomplishment is that the entire workflow was successful. From the operation of the board to the firmware operation to the software processing and GUI. The product is one that correctly solved the problem of creating affordable water ski coaching. The Processing subsystem successfully smoothed the data to take the high noise out and show the curve of the skier. We are also happy with how the GUI turned out as it makes the data very readable.

7.2 Uncertainties

Our biggest uncertainties were in our sensor data. The GPS was only accurate up to 3m and had a low polling rate. We would have preferred to have higher GPS accuracy but that was not feasible with our time and budget. Another issue we had was with our IR receiver. Due to I/O polling issues on the STM chip, we were unable to get the IR receiver to work.

7.3 Future Work

If we had an infinite budget, the main improvement we would have made if we had more money would be to change to an RTK GPS System. The current system lacks precise GPS data due to the low precision of cheap consumer GPS modules, so switching to an RTK system will help. RTK uses a base station to provide centimeter level accuracy of the GPS. We would also like to fix the IR receiver issues we had and potentially add more user feedback such as an LCD screen or a phone app.

7.4 Ethical Considerations

The ACM Code of Ethics section 1.6 discusses respecting privacy. Our product will make sure to store no information for outside parties to access. While we must store location information for usage by our processing algorithms, we will not access other users' data, and all data will be held locally.

8 References and links

[1] “MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2 MPU-6000 MPU-6050 Register Map and Descriptions,” 2013. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[2] G. Baddeley, “GPS - NMEA sentence information,” Jul. 20, 2001. https://aprs.gids.nl/nmea/

[3] *Reading and writing images and video¶*. Reading and Writing Images and Video - OpenCV 2.4.13.7 documentation. (n.d.). https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html