Low Cost Rowing Tracker

By
Nathaniel Zurcher
Jai Agrawal
Kerem Gurpinar

Final Report for ECE445, Spring 2018
TA: Nick Ratajczyk

April 2018

Project No. 47

Abstract

In this report, we detail the design and building of our Rowing Tracker. In implementing our design, we used our knowledge of rowing physics as well as electrical engineering concepts to create a low-cost product that provides rowers with statistical analysis of their rowing technique. In this report we preface with our project's objectives. Following the objectives, are the design process and requirements and verifications. We end with a cost breakdown, our conclusions, and plans for the future of this project.

Table of Contents

1. Introduction

1.1 Objective

Olympic style rowing has existed since the 1700's [1]. It's popularity has fluctuated over time, but has been on the rise since the 1970's[2]. Traditionally an outdoor sport, rowers work in typically eight man boats and work in unison to achieve the highest rowing efficiency possible. During winter months the sport moves indoors where rowing machines are used to keep the rowers in top shape through rigorous training and distances tests. These indoor machines provide detailed insight into the power of the rower given in watts and make setting and achieving goals trivial. This brings us to the problem we want to solve. While on the water there is no proficient method for a coach to measure how the individuals in the boat are contributing. While some solutions exist, such as the Peach from Peachinovations, they are prohibitively expensive due to their oarlock design [3].

Our objective with this project is to design an affordable system that will allow us to measure the force output of individual rowers, and use that data to calculate the power output as well as provide additional information that would provide rowers with a better understanding of areas that they need to improve. This data will be saved on the device for the coach to upload to his computer after a rowing session. Currently there are affordable devices that measure the distance and the pace of a boat based on a GPS reading, but there are not any affordable ways to determine each person's individual contribution in boats with multiple rowers. We will use force sensors planted under the feet of rowers, which will measure how much pressure they make against the boat as they row. The data, along with GPS readings, will paint a clear picture about how much each person contributes.

1.2 High Level Requirements

1. Accurately record force output of rower and correlate it to watts and pace.
2. Store at least 80 hours of rowing data that can be accessed after the rowing session.
3. Record time, position, and speed every second from GPS and record it to the SD card.

2. Design

2.1 Block Diagram

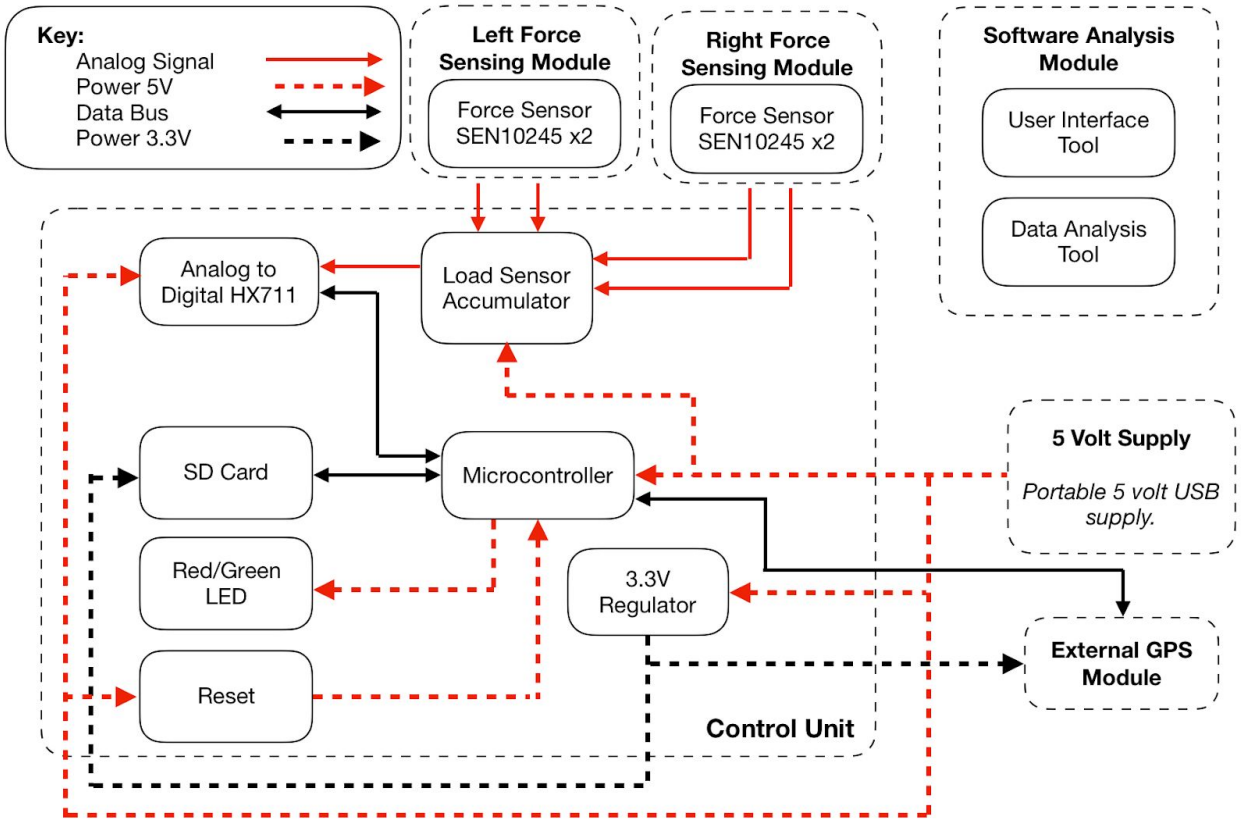

Figure 1: Block Diagram for Rowing Tracker

Left/Right Force Sensing Module: Each module consists of two very thin plates sandwiching force sensors to be inserted in existing boat footwear. Each force module contains two strain gauges, for a total of four, which recieve force input from the user's feet. The four analog signals are sent to the control unit where they are combined to one using a wheatstone bridge. The combined signal is converted by an analog to digital converter and relayed to the microcontroller for recording.

5 volt Supply: Standard USB portable charger with internal lithium ion battery and booster circuit. Supplies 5.0 volts at a maximum of 1.0 amp to our Control Unit. Indicator bar lights when powered and shows current charge level.

External GPS Module: The XA1110 GPS is a low power unit supplied 3.3 volts from the linear regulator located on the control unit. Communicates via I2C to the microcontroller relaying timestamps, position, and speed to be recorded.

Control Unit: Houses ATMega328P microcontroller, HX711 analog to digital converter, and SanDisk Ultra SD card. The microcontroller is programmed to read data from the GPS Module every second and analog to digital converter at least ten times per second. The microcontroller then transmits that data to the SanDisk Ultra SD card over the serial peripheral interface, recording it as a CSV for the software analysis tool.

Software Analysis Tool: Python script run on user's computer, responsible for interpreting the CSV generated by the control unit. Easy to use interface allows user to select data for analysis. Smooths selected data and extracts pertinent information from it, such as average watts, average strokes per minute.

2.2 Left/Right Force Sensing Module

2.2.1 Force Sensor SEN10245

We used four 3-wire strain gauges to measure the force exerted by the rower during each stroke. 4-wire strain gauges would have garnered greater accuracy, but at a higher unit cost and a more difficult conversion from four units to one output. At 1 cm tall the sensors allow for a thinner final package making the module easier to fit inside a rowing shoe. Research told us that an olympic rower's max force output was 183 kg [4]. With a max load of 50kg(110.2lbs) per sensor our system has a maximum load of 200kg(440.8lbs) making these perfect for our needs.

2.3 5 Volt Supply

Our original design had us picking our own lithium polymer battery, designing the charging circuit, and designing the 5 volt booster circuit to power the Control Unit. In order to save time and money we decided to use a device we already had on hand, a portable phone charger. With a 5 volt 1 amp output this device was more than adequate to power our unit.

2.4 External GPS Module XA1110

An external GPS module was used because we did not want it to be directly attached to the Control Unit. The externality gave us the flexibility to place the module wherever it received the best satellite reception. The unit also included a backup RTC battery that enabled it to generate a quicker fix upon power up. This was a vital feature to us as all the time stamps, required for recording the tracking data, come from the GPS module. The device supports I2C communication protocol which is compatible with our chosen ATMega328P microcontroller.

2.5 Control Unit

2.5.1 Microcontroller ATMega328P

We chose the ATMega328P because it is utilized on the Arduino Uno hardware. This combined with with it being a PDIP allowed for easy programming of the part using the Arduino Uno board and granted us access to the Arduino IDE[5]. The Uno also gave us a control environment to test from, simplifying the debugging process. With 18 digital I/O pins available, IC2 support, and SPI support, this microcontroller will be more than capable of communicating with all our peripheral devices. To make transferring the device from the Arduino to our control unit simple we matched the external clock configuration used by the Uno board (a 16MHz clock) [6].

The microcontroller code, see Appendix D, consists of a startup sequence followed by a constantly running loop. A majority of the processes take advantage of Arduino libraries, see Appendix D for list of libraries used. The microcontroller's startup sequence sets the status LEDs to red, indicating it is powered but not yet recording. After communication with the GPS unit and microSD card are established a new unique training file is created, and the green LED lights indicating recording has begun.

Next, the device moves into the loop phase where it polls for new input from the analog to digital converter. When new data is detected, it records it with the latest GPS data in CSV format to the microSD card.

If an error with the microSD card is detected at any time the red LED alone is illuminated. An error with the GPS module results in illumination of the red and green LEDs.

2.5.2 SanDisk Ultra 8GB MicroSD Card

The main criteria for the microSD card were easily satisfied given modern technology. We estimated our device would generate at most 1 KB/second (though in most cases much less). This is a very small amount of data compared to the size of widely available microSD cards. With an SPI write speed of 100 Mbits/second this card would greatly exceed our needs [7]. The capacity of the microSD card was also considered a non-issue as this 8GB card was the smallest size readily available and would supply over 2,000 hours of recording.

2.5.3 Load Sensor Accumulator

Originally our design included four analog to digital converters, one for each force sensor, but we realized this was a waste of power and money because we had no use for the individual force outputs, only the sum total of the sensors. Therefore, we implemented a wheatstone bridge, as shown in Figure 2 [8]. In Figure 2, each resistor (R1, R2, R3, and R4) is replaced with a variable resistor in the form of one of the 3-wire strain gauges. R1 and R3 are wired in a way such that as force increases resistance increases, while R2 and R4 are wired in a way such that as force increases resistance decreases. This results in the VOUT voltage corresponding with the total force on all four

3-wire strain gauges. Therefore, only one analog to digital converter is required to process the analog input from VOUT simplifying our original design.

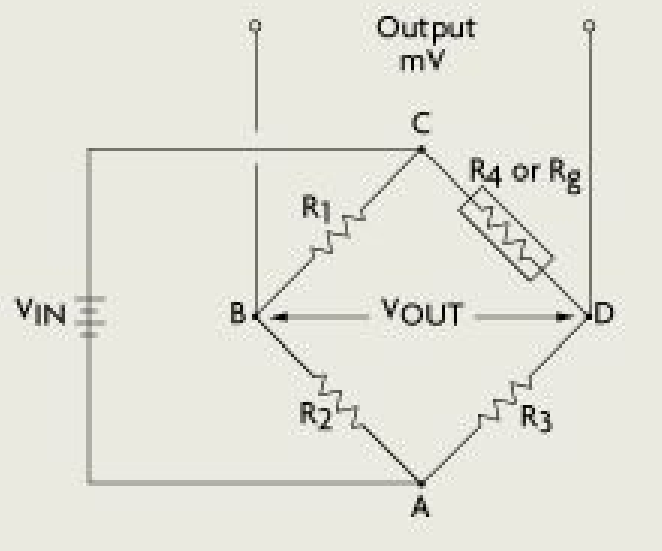

Figure 2: Wheatstone Bridge Schematic

2.5.4 Analog to Digital Converter HX711

Our group looked at several different analog to digital converters. We started by considering the A to D's built into the ATMega328 but ruled them out due their lack of granularity. We needed the converter to read a range from 0 to 200 kg (440 lbs). At only 10 bits the microcontroller's A to D would have only provided half pound increments. We decided to go with the HX711 because it was a 24 bit converter, giving us a granularity of .00002, far smaller than the noise floor we would be working with. In addition to this it ran on 5 volts, the same as our microcontroller, meaning it would require no additional regulators [9]. The final reason we selected this A to D was its 10 Hz sampling rate would be more than adequate to read a stroke that would take, at the shortest, 2.5 seconds to complete.

2.5.5 Red/Green LED

As the device will be used mostly outdoors the LEDs needed to be bright enough to be seen in direct sunlight. We selected the brightest LEDs available to us while staying within the 40mA absolute max current draw of the ATMega328P's digital I/O pin. The LEDs we picked had a running current draw of 20 mA, a safe margin from the absolute max. We placed each of these LEDs on a digital I/O pin on the ATMega in series with a 1,500 Ohm resistor. This resistance value was calculated using Ohm's law and the 2.0 volt forward voltage of the LED, Equation 1 [10].

$$Resistance = \frac{Supply\ Voltage - Diode\ Forward\ Voltage}{Current} = \frac{5.0\ Volts - 2.0 Volts}{.002\ Amps} = 1,500 Ohms \qquad (1)$$

We wired the LEDs such that their anode faced the ATMega and the cathode faced the resistor. A 5 volt supply on the other side of the resistor meant setting the corresponding digital I/O pin to low, 0 volts, would light the LED while setting it to high, 5 volts, would disable the LED.

2.5.6 Reset Button

A simple switch, in the unpressed state it acts as an infinite resistor, in the pressed state it acts as a short. The ATMega328 requires a 5 volt "high" reading on the RESET pin in order to operate, when a 0 volt "low" reading is detected a reset is triggered. We wired the button such that one end was connected to ground and the other to 5 volts and the RESET pin on the microcontroller. The 5 volts first passes through a 10,000 Ohm resistor, chosen for its large resistance to minimize current draw during button press. When the button is pressed the 5 volt line goes to zero volts and using a variation of Ohm's law a, $\frac{5.0Volt}{10{,}000Ohm} = 0.5mA$ load passes through the resistor. This triggers a reset on the ATMega. The configuration of the ATMega causes a new file to be generated upon reset allowing the button to be used to start new training session files.

2.5.7 Linear Voltage Regulator

Two of the devices employed by our Rowing Tracker run on 3.3 volts, the GPS Module and microSD card. Referencing the documentation of each the GPS [11] and microSD [7] draw a maximum of 35mA and 100mA respectively. We therefore needed a 3.3 volt regulator capable of supplying 135mA from a 5 volt source. A brief search led us to the SOT-23-5 a very widely trusted linear regulator due to its low cost and use in many of Arduino's boards. It is capable of converting 150mA continuously from 5 volts to 3.3 volts [12].

2.6 Software Analysis Module

2.6.1 User Interface Tool

The user interface (UI) tool was designed so rowers could easily select a region of their rowing session to analysis in the Data Analysis tool. The UI tool is run through python and utilizes a graphics library to visually represent the data received by the SD card as well as allows users to select an interval by dragging bounds on the a graph [13]. The reason we chose to implement this is was because rowers may start the rowing tracker but not necessarily start rowing immediately leaving areas where the data is not useable. The UI tool also allows flexibility for rowers to pick and choose which sections of their rowing session they would like to measure.

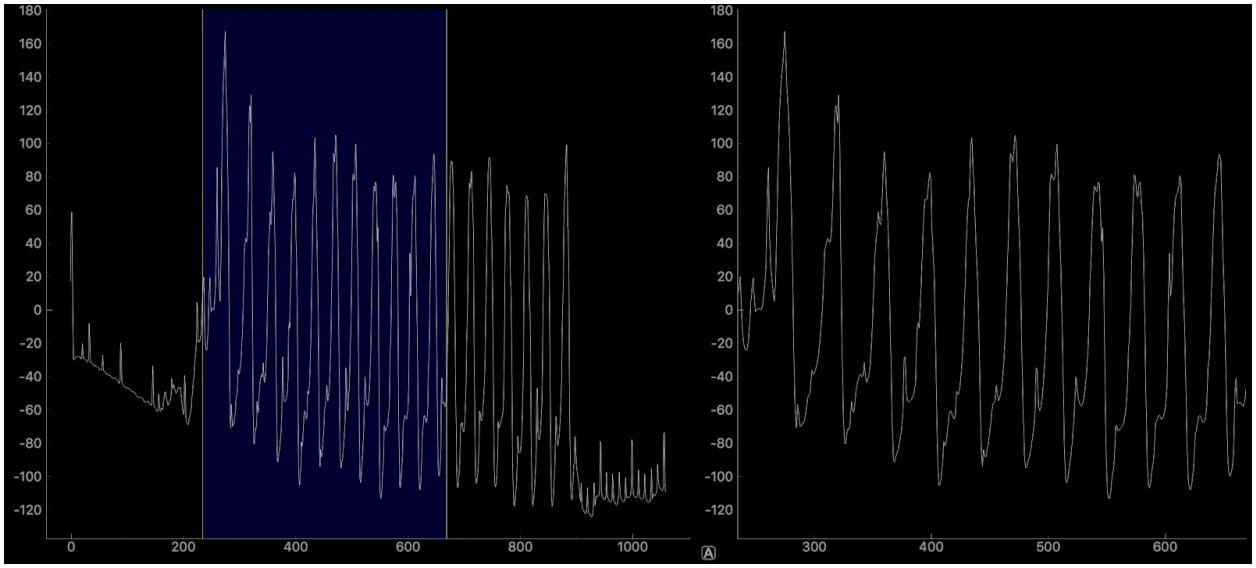

Figure 3: UI Tool with interval selection on left and zoomed-in version of interval on right

2.6.2 Data Analysis Tool

The data analysis tool consists of a python script , see Appendix D, that does the computation required to provide pace, wattage, and strokes per minute (as well provides all other useful information that the tracker records). The data analysis tool is also responsible for smoothing out the data received from the tracker. Since the force sensors are sampling at a high rate (receiving approximately 12 samples per second), the data receives a lot of noise and requires smoothing to more accurately depict a rowers force curve. To do this we implemented a Savitzky–Golay filter which smoothed the force curves [14]. Using these force curves we calculated the average force and from there used the data from the GPS along side the average force to calculate the wattage. The method for calculating the average
force is given by Equation 1 and the resulting wattage calculation is given in Equation 2.

$$average\ force = \frac{1}{number\ of\ samples} \int_{left\ bound\ sample}^{right\ bound\ sample} force(t)\ dt \qquad (2)$$

$$watts = average\ force\ *\ average\ speed \qquad (3)$$

2.7 Mechanical Design

Since the rowing tracker force sensors would be dealing with tremendous amounts of force we had to design plates that would protect them and yet still be able to transmit the force from the rowers feet to the sensors accurately. After consulting the machine shop, we decided to use 0.07 inch steel plates which would allow for transmission of force while still being robust. Another aspect of

designing the foot plates was that it had to be able to fit inside existing rowing footwear easily. As a result we designed the plates to be as thin as possible and fit within a 3.5 by 10 inch rectangle.

Figure 4: Top view of steel foot plate

3. Design Verifications

3.1 Left/Right Force Sensing Module

3.1.1 Force Sensor SEN10245

Force sensors needed to be capable of registering an entire rowing stroke. We ran the device with the force plates mounted to an indoor rowing machine. We had a rower row as hard as possible for four strokes and viewed the data produced, see Figure 5. Clearly at no point in the recording was there aliasing of the data, all 4 strokes taken are represented, and at no point were the sensors maxed. The peaks are clearly smooth and not clipped.

3.2 5 Volt Supply

Required a 5 volt supply capable of supplying 200mA continuous load with peaks of 250mA. We tested this using a voltmeter, ammeter, and resistors. We wanted to simulate an extreme load of 500mA that our system would never actually generate. Setting up a simple test rig we placed $\frac{5\ Volts}{.500\ Amps} = 5\ Ohms$ of resistance at the end of the USB connecting our external battery pack to our Control Unit. We ran this test for 15 minutes monitoring the temperature of components and voltage. The battery packs temperature remained constant throughout and the voltage at the Control Unit never dipped below 4.94 volts. This was within our design parameters thus the battery pack met our requirements.

Required the battery life of the Rowing Tracker to be between 6 and 10 hours. We ran the tracker for 6 hours straight and the battery indicator light on the battery module did not indicate any loss in charge. This requirement was not only met, but exceeded as the pack still had a considerable charge remaining.

3.3 External GPS Module XA1110

The GPS Module must be able to provide the date, time, position, and speed at least once a second. We confirmed this by running the microcontroller on our test platform, the Arduino Uno. We connected the Uno to a laptop and had the microcontroller display the GPS output over the serial interface to the computer. We clearly received updates too all the requested parameters every second confirming the unit met our requirements.

3.4 Control Unit

3.4.1 Microcontroller ATMega328P

To confirm the ATMega328 operated correctly on our control unit we uploaded it with a simple program to change the color of the LEDs from red to green ever second. After powering the device we immediately saw the lights changing colors verifying the microcontroller operated on our printed circuit board.

To confirm the ATMega328 was fast enough to record at least 10 samples per second from the analog to digital converter and GPS data to the microSD card we set up the device regularly. We then ran the device for 2 minutes. Afterwards we looked at the time stamps and correlated the number of samples in every second. We averaged about 12 force samples/second which is slightly faster than our required 10 samples/second.

3.4.2 SanDisk Ultra 8GB MicroSD Card

SD card had to be large enough to store hundreds of 12 hour rowing sessions. This was tested by running the device normally. The data was then checked on the computer for its file size and length of test. For the test we ran a 47KB file was generated in 138 seconds. This implied a write speed of 340.5 bytes/second. At that rate it would take $\frac{8{,}000{,}000{,}000\ bytes}{340.5\ bytes/second} = 23494860.5\ seconds\ or\ 272\ days$ of continuous recording. Clearly the 8GB card could store far more than hundreds of 12 hour sessions.

We also required that the microSD card be capable of writing all the samples produced by the analog to digital converter and GPS without slowing down the system. We confirmed this by comparing the previously tested for value of 340.5 bytes/second to the datasheet max SPI speed of 100 Mbit/second (12,500,000 bytes/second) we were well within the specs of the card, proving it was not a limiting factor in the speed of our recording [7].

3.4.3 Load Sensor Accumulator

The load sensor accumulator needed to accurately sum each of the force sensors and report it to the analog to digital converter. This was tested on our testing platform, the Arduino Uno, by placing a constant mass of 10 lbs on each of the four force sensors and monitoring their output on the serial interface. We verified that the results on each of the four sensors was equal. We then tested the system by placing a 10 lbs load on each of the sensors and verified the output was four times greater than the previous results. Confirming this proved to us that the system was indeed summing the load of the four force sensors.

3.4.4 Analog to Digital Converter HX711

Required to produce at least 10 samples per second to produce enough data to properly analyze each stroke. Tested using our testing rig, the Arduino Uno, with the serial interface active we printed new results from the HX711 analog to digital converter and counted how many samples were being

received each second. On average the system output approximately 12 samples/second verifying our design.

3.4.5 Red/Green LED

LED must light up and be visible in direct sunlight. Placed the control unit outside in direct sunlight and supplied 5 volts from the portable battery pack to the opposite side of the resistor of the LED. We then grounded the anode side of the LED illuminating it. This process was repeated for both the red and green LED and both were visibly lit in direct sunlight.

3.4.6 Reset Button

The reset button must be capable of triggering a reset on the ATMega328P. This was tested by uploading the ATMega with our standard rowing tracker code. We then ran the device for 20 seconds and pressed the reset button. We allowed the device to run for another 20 seconds. Upon analyzing the data recorded to the microSD card we confirmed two seperate tracking files were generated each with 20 seconds of data. This proves the reset button behaves as designed.

3.4.7 Linear Voltage Regulator

The linear regulator must be capable of supplying a maximum of 133mA at 3.3 volts to the SD card and GPS Module. Tested with current load of 140 mA using resistors in series to generate $\frac{3.3\ Volts}{.14\ Amps} = 23.4\ Ohms$ of resistance. Voltage was then tested and at various 3.3 volt sites on the board. Voltage sage amounted to 3.24 volts on the multimeter. This is within the lower bounds of the more sensitive GPS Module[11]. After 15 minutes there was no noticeable change in temperature on the regulator or any of the vias. The regulator passed the verification.

3.5 Software Analysis Module

3.5.1 User Interface Tool

The UI tool worked as intended, providing the users with an easy to use interface. Users were able to view their rowing force curve and set the interval for which they wanted to run the data analysis tool. See Fig. 3 above to view the UI tool interval selection.

3.5.2 Data Analysis Tool

The data analysis tool was able to calculate the same metrics as the rowing machine and therefore meets the requirements we set out for it. It provides: the watts, pace, and strokes per minute (as well as several other extra metrics). The analysis tool exceeds our requirements set for it, however the accuracy of the metrics could be improved further with access to better rowing equipment. The tool also accurately smoothed data as we wished it to. This allowed for better readings and for our calculations to be more representing of a rowers stroke.

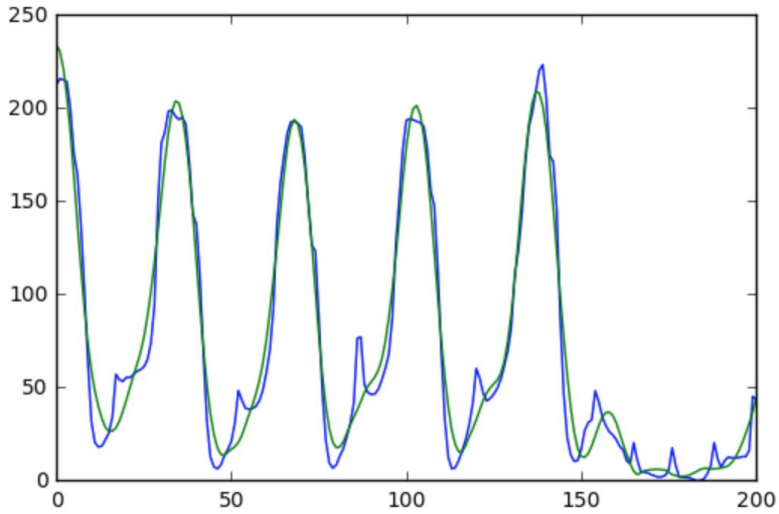

Figure 5: Output from data analysis tool. Data with noise graphed in blue and data smoothed graphed in green.

4. Cost

4.1 Parts

Total cost of the parts we used for a single unit of our design is $128.34 One rowing boat will have 8 rowers, which brings it to a total of $1,540.08. Table *** in Appendix C shows a detailed list of the parts we used.

4.2 Labor

Labor cost for the project is set to $35/hr, which is average hourly salary for an engineer. Each member worked approximately 10 hours per week on the project for 16 weeks over the semester. This brings the total labor cost to

$$\$35/hr \times 10\ hr/week \times 16\ week \times 2.5 \times 3\ partners = \$42{,}000.$$

Total cost to produce a single unit sums up to $42,128.34.

5. Conclusion

5.1 Accomplishments

Although, our tracker had was not completely functional during the final demo we were able to successfully achieve all of our objectives. The force sensing modules were able to sample and record the force generated by the rower without maxing out. The power module was able to successfully power the tracker for more than 6 hours. The control unit was able to read in all values and transmit them to the SD card with no flaws. The software unit was able to interpret the data and provide the rowing machine value equivalent. Although, our tracker prototype does not yet resemble a finished product, the functionality of the tracker is all there.

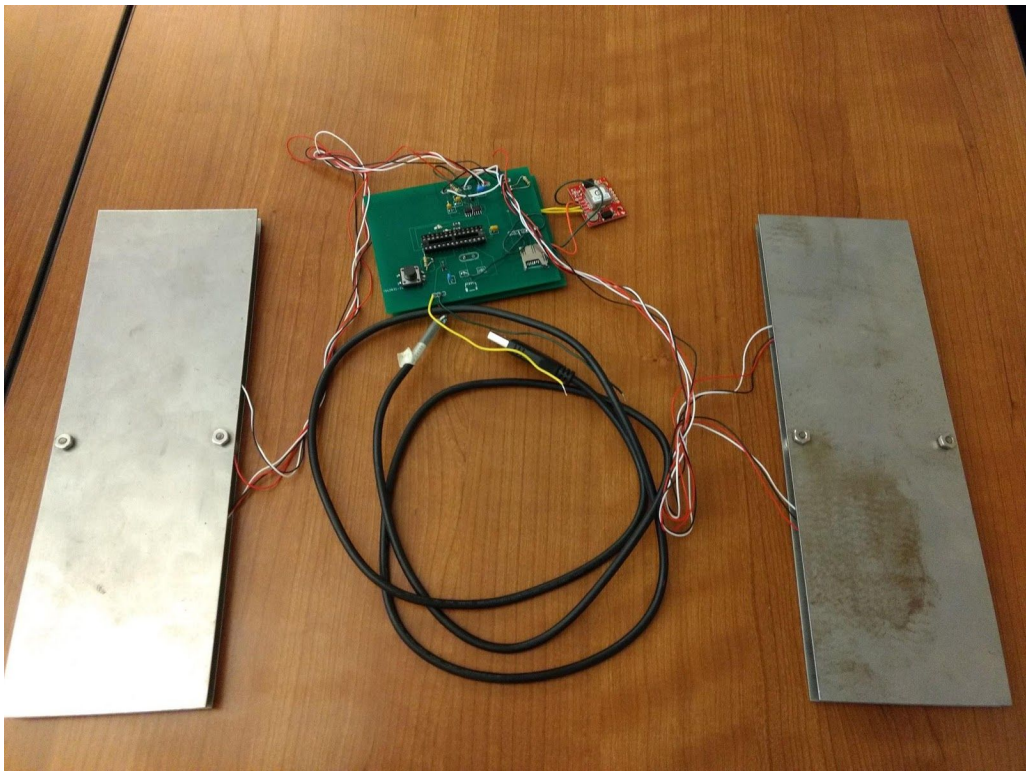

Figure 6: Naked view of Rowing Tracker device.

5.2 Uncertainties

While our device gives precise readings regarding force curves and speed calculations, it was difficult to correlate these readings to wattage accurately. To do so we would require special equipment (that we did not have access to on campus) and therefore our correlations may not be as accurate as we

would like them to be. However, as we will discuss in the Future Work section, this problem can easily be fixed with access to the proper equipment to test with.

5.3 Safety and Ethical Considerations

Control unit (PCB) will be placed under every rower, so any malfunctions related to power supply may cause PCB to heat up. It can be very dangerous for the rowers if the voltage regulator fails to operate correctly, and overheat the system.

System has to be durable in wet environment. A waterproof box is necessary for the control unit, or else any splashes of water may cause safety hazards. In addition to the control unit, the sensor module needs to be waterproof as well for the accuracy of the data collected.

The primary goal of our project will be to correlate force input on sensors to a more usable unit, power. This conversion will be based off testing and data correlation from a rowing machine. As this is uncharted territory we will have to develop a correlation algorithm on our own. We could very easily overstate the accuracy of our device at displaying power output. This would be a breach of IEEE standard #3 [15]. We cannot make claims off data that are not realistic.

Some aspects of the project will beyond our ability to accurately deem safe. Specifically, we will need to find someone with more knowledge and expertise in the area of splash resistant electronics. If our group members attempted to certify this as splash resistant it would violate IEEE #6 as we do not have the requisite knowledge to make such a claim[15].

5.4 Future Work

At this moment our design is a single unit, and one of our future goals is to expand the system to multiple rowers on a single boat. In order to do so we need to do more tests and quantitative analysis to improve the accuracy of our collected data. Moreover, we need to make the system waterproof since it is placed inside the boat. Control unit and foot sensing modules have to be durable in wet environment. We need to make a waterproof box to fit our modules, so that we can minimize the risk of water splashing and causing any safety hazards for the rowers. Finally, we will enhance our connections by finding better ways to connect/disconnect foot sensing module to control unit. This way it will be easier to change or replace any malfunctioning components of the system.

References

[1] Nauright, John; Parrish, Charles, eds. (2012). Sports around the world history, culture, and practice. Santa Barbara, Calif.: ABC-CLIO. p. 169.

[2] Brown, NCAA.com Gary. "Rowing increases in popularity." NCAA.com, 27 May 2012, www.ncaa.com/news/rowing/article/2012-05-23/rowing-increases-popularity.

[3] "PowerLine Rowing Instrumentation and Telemetry." Peach Innovations - Rowing Telemetry and Instrumentation, www.peachinnovations.com/.

[4] Strength Goals for Rowers." PEAK CENTRE, 20 Jan. 2009, peakcentre.wordpress.com/2009/01/20/strength-goals-for-rowers/.

[5] Arduino - Software, www.arduino.cc/en/Main/Software.

[6] Arduino Uno Rev3, store.arduino.cc/usa/arduino-uno-rev3.

[7] Debbie.werbeloff. "OEM Product Manual." SanDisk MicroSD, www.alliedelec.com/m/d/04db416b291011446889dbd6129e2644.pdf.

[8] "OMEGA Engineering." Omega Engineering, www.omega.co.uk/literature/transactions/volume3/strain2.html.

[9] AVIA semiconductor. 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales, Datasheet. www.bing.com/cr?IG=71673AD79FAF43DABADB8657470854E4&CID=06FEB1CB418C69EE3881BA2F402368EF&rd=1&h=zhG__aLmjs7fmdMJloSI4CErUQlIy6NXOQis_bbl_vw&v=1&r=https%3a%2f%2fcdn.sparkfun.com%2fdatasheets%2fSensors%2fForceFlex%2fhx711_english.pdf&p=DevEx.LB.1,5528.1.

[10]"LTST-C230KGKT." Lite-On Inc. | Optoelectronics | DigiKey, www.digikey.com/product-detail/en/lite-on-inc/LTST-C230KGKT/160-1456-1-ND/386854.

[11] Datasheet, Titan X1, and Version: V0A. "Titan X1." Titan X1.

[12] [Slvsbc1D, Texas Instruments Incorporated, and]. "1.5-A High Efficiency Step-Down Converters in SOT-23 5-Pin Package Datasheet (Rev. D)." 1.5-A High Efficiency Step-Down Converters in SOT-23 5-Pin Package Datasheet (Rev. D).

[13] "PyQtGraph - Scientific Graphics and GUI Library for Python." PyQtGraph - Scientific Graphics and GUI Library for Python, www.pyqtgraph.org/.

[14] "Savitzky Golay Filtering¶." Savitzky Golay Filtering - SciPy Cookbook Documentation, scipy-cookbook.readthedocs.io/items/SavitzkyGolay.html.

[15] IEEE.org. (2017). "IEEE Code of Ethics." [Online] http://www.ieee.org/about/corporate/governance/p7- 8.html [Accessed: 21 Sep. 2017].

[16]Bogde. "Bogde/HX711." GitHub, 11 Jan. 2017, github.com/bogde/HX711.

[17]Greiman. "Greiman/SdFat." GitHub, 23 Dec. 2017, github.com/greiman/SdFat.

[18]"SparkFun GPS Breakout - XA1110 (Qwiic)." Learn at SparkFun Electronics, learn.sparkfun.com/tutorials/sparkfun-gps-breakout---xa1110-qwiic-hookup-guide.

Appendix B Requirement and Verification Table

B.1 Power Unit

B.1.1 Battery Pack

Requirements	Verification	Status
Battery Pack Supplies 5 volts to PCB	*Check output voltage of battery pack.*	*Y*
Capable of storing 2500mAh to maintain 6-10 hour operating window	*Check the mAh rating of the battery.*	*Y*
Produce sustained load of 250mA for the voltage regulator with spikes of 300mA	*Check on multimeter if the battery produces sustained load of 250mA.*	*Y*

B.1.2 Voltage Regulator

Requirements	Verification	Status
Supply 3.3 volts ± 9%	*Check for 3.3V output from the linear regulator*	*Y*
3.3V linear regulator needs to supply an average load of 67mA with peaks of 100mA	*Place 100mA load on the regulator and make sure voltage drop is within 2.7 volt minimum of sd card and gps module.*	*Y*

B.2 Force Sensing Unit

B.2.1 Force Sensors

Requirements	Verification	Status
Force sensors able to detect an entire rowing stroke without maxing out	*Set up system with force sensors installed under feet and have rower pull as hard as possible. Check the outputs for clipping.*	*Y*

B.3 Control Unit

B.3.1 Microcontroller

Requirements	Verification	Status
Receive and process GPS coordinate inputs in real time and digital signals from 4 force sensors simultaneously	*A. Set up simulation code for microcontroller* *B. Have microcontroller print A to D and GPS data to serial output and read on computer* *C. Verify the data appears un-corrupted and that the minimum required samples per second are being met*	*Y*
Red LED signifies device is booting and not yet recording. Green LED signifies device is active and recording. Red and Green LED on simultaneously signify GPS issue. Red on after short flash of green signifies SD card missing, improperly formatted, or broken.	*Power device and verify red LED comes on first. Unplug device before green light comes on and verify no data was stored to SD card. Disconnect the GPS unit, power the device and verify both the green and red LEDS light.* *Remove the SD card, power the device, and verify after termporarily flashing green, the red light alone illuminates.*	*Y*

B.3.2 Analog to Digital Converter

Requirements	Verification	Status
At least 1 pound granularity	*Place marginally heavier items on the force plates and see what the system is accurately able to differentiate.*	*Y*
A to D must be compatible with chosen microcontroller	*Connect the A to D to a digital in/out pin the ATMega328 and print its value to the Serial port. Display the value on a computer and verify the outputs of the A to D are being accurately interpreted by the microcontroller.*	*Y*

B.3.3 SD Card

Requirements	Verification	Status
SD card must be capable of writing at least 10 samples per second from the A to D and 1 sample per second from the GPS	*After wiring the system together verify with a computer that the required data was accurately recorded to the SD card*	*Y*
Data stored to SD card must be formatted in a readable csv for the computer running script.	*Write code in Arduino for the ATMega328 that prints your formatted output planned for the SD card to the serial monitor. Read the outputted data on the computer and ensure the csv is formatted as intended. Date(DDMMYY),hours, minutes, seconds, force(lbs), latitude, longitude, speed(km/h)*	*Y*
Max storage capable of saving hundreds of 12 hour of rowing sessions without issue.	*Place the SD card in a computer and verify it is capable of storing a file at least as big as: 8*	*Y*

	*bytes/second from force sensor + 32 byte/second from GPS module; 40 bytes/second * 3600 seconds/hour*12 hours = 1728 KB in 12 hours of running * 500 rowing sessions = 864,000 KB.*	

B.3.4 GPS IC

Requirements	**Verification**	**Status**
GPS must be able to sample at least at 1Hz	*Wire the GPS to the I2C connections on the ATMega328. After booting the GPS module query it see if valid responses are received every second.*	*Y*

B.4 Software Unit

B.4.1 Software

Requirements	**Verification**	**Status**
Software allows user to easily select what data to analyze.	*Ask someone uninvolved with the design process to attempt to use the software.*	*Y*
Data analysis tool records and displays tracker data in the forms that rower would understand and benefit from (i.e. number of strokes, force, pace etc.).	*Check printed data with raw data inputted from the SD to see if values are accurate and on par with rowers needs.*	*Y*

Appendix C Parts List

Part	Part Number (SparkFun)	Quantity	Price	Total
Crystal 16MHz	COM-00536	1	$0.95	$0.95
Tactile Button - SMD (12mm)	COM-12993	1	$0.50	$0.50
Arduino Uno - R3	DEV-11021	1	$24.95	$24.95
SparkFun GPS Breakout - XA1110 (Qwiic)	GPS-14414	1	$44.95	$44.95
microSD Socket for Transflash	PRT-00127	1	$3.95	$3.95
Qwiic Cable - 100mm	PRT-14427	1	$1.50	$1.50
Load Sensor - 50kg	SEN-10245	4	$9.95	$39.80
SparkFun Load Cell Amplifier - HX711	SEN-13879	1	$9.95	$9.95
Total				**$126.55**

Part	Part Number (Digi-Key)	Quantity	Price	Total
100 Ohms ±1% 0.05W, 1/20W Chip Resistor 0201 (0603 Metric) Thick Film	541-100AABCT-ND	2	$0.24	$0.48
10 kOhms ±5% 0.05W, 1/20W Chip Resistor 0201 (0603	311-10KNCT-ND	1	$0.10	$0.10

Metric) Moisture Resistant Thick Film				
1.5 kOhms ±1% 0.5W, 1/2W Chip Resistor 1206 (3216 Metric) Anti-Sulfur, Moisture Resistant Thin Film	RNCP1206FTD1K50CT-ND	2	$0.10	$0.20
20 kOhms ±1% 0.05W, 1/20W Chip Resistor 0201 (0603 Metric) Moisture Resistant Thick Film	YAG2276CT-ND	1	$0.10	$0.10
8.2 kOhms ±1% 0.05W, 1/20W Chip Resistor 0201 (0603 Metric) Moisture Resistant Thick Film	YAG2353CT-ND	1	$0.10	$0.10
Bipolar (BJT) Transistor PNP 25V 1.5A 100MHz 625mW Surface Mount SOT-23	MMSS8550-H-TPMSCT-ND	1	$0.21	$0.21
Linear Voltage Regulator IC Positive Fixed 1 Output 3.3V 150mA SOT-23-5	296-18476-1-ND	1	$0.60	$0.60
Total				**$1.79**

Appendix D Code

D.1 User Interface Tool Code:

```
from pyqtgraph.Qt import QtGui, QtCore
import numpy as np
import pyqtgraph as pg
import csv

#QtGui.QApplication.setGraphicsSystem('raster')
app = QtGui.QApplication([])
#mw = QtGui.QMainWindow()
#mw.resize(800,800)

f = open('TEST9.CSV')
csv_f = csv.reader(f)

force = []

for row in csv_f:
    (force.append(float(row[4])))
#print(force)



win = pg.GraphicsWindow(title="Basic plotting examples")
win.resize(1250,600)
win.setWindowTitle('pyqtgraph example: Plotting')

# Enable antialiasing for prettier plots
pg.setConfigOptions(antialias=True)



x2 = np.linspace(-100, 100, 1000)
data2 = force
p8 = win.addPlot(title="Region Selection")
```

```
p8.plot(data2, pen=(255,255,255,200))
lr = pg.LinearRegionItem([400,700])
lr.setZValue(-10)
p8.addItem(lr)

text = ''
black = (0,0,0)
white = (255,255,255)
red = (255,0,0)
p9 = win.addPlot(title="Zoom on selected region")

p9.plot(data2)
text_display = pg.TextItem()
def updatePlot(text_display):

    p9.setXRange(*lr.getRegion(), padding=0)

def updateRegion():
    lr.setRegion(p9.getViewBox().viewRange()[0])
    axX = p9.getAxis('bottom')
    left_bound = (format(axX.range[0]//1))
    right_bound = (format(axX.range[1]//1))
    print('The left bound is: ',left_bound)
    print('The right bound is: ',right_bound)
lr.sigRegionChanged.connect(updatePlot)
p9.sigXRangeChanged.connect(updateRegion)
p9.setXRange(*lr.getRegion(), padding=0)

updatePlot(text_display)
updateRegion()


if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
```

D.1.1 References for User Interface Tool Code:

[13]

D.2 Data Analysis Tool Code:

```
#from savitzky golay cookbook cited below
def savitzky_golay(y, window_size, order, deriv=0, rate=1):
    import numpy as np
    from math import factorial

    try:
        window_size = np.abs(np.int(window_size))
        order = np.abs(np.int(order))
    except ValueError, msg:
        raise ValueError("window_size and order have to be of type int")
    if window_size % 2 != 1 or window_size < 1:
        raise TypeError("window_size size must be a positive odd number")
    if window_size < order + 2:
        raise TypeError("window_size is too small for the polynomials order")
    order_range = range(order+1)
    half_window = (window_size -1) // 2
    # precompute coefficients
    b = np.mat([[k**i for i in order_range] for k in range(-half_window, half_window+1)])
    m = np.linalg.pinv(b).A[deriv] * rate**deriv * factorial(deriv)
    # pad the signal at the extremes with
    # values taken from the signal itself
    firstvals = y[0] - np.abs( y[1:half_window+1][::-1] - y[0] )
    lastvals = y[-1] + np.abs(y[-half_window-1:-1][::-1] - y[-1])
    y = np.concatenate((firstvals, y, lastvals))
    return np.convolve( m[::-1], y, mode='valid')


my_data = genfromtxt('TEST9.csv', delimiter=',')
my_datav = pd.read_csv("TEST9.csv")
#print my_data
#print my_datav
fig = plt.figure()
x = my_data[:,4]
hour = my_data[:,1]
minute = my_data[:,2]
second = my_data[:,3]
```

```
speed = my_data[:,7]
x_plot = plt.figure()
#print x
#plt.plot(x)
l = 0
#plt.plot(x)
#plt.plot(x)
#print x
y = []
velocity = []
m = 0
count = 0
ui1 = 743
ui2 = 945
samples = ui2 - ui1
bound1 = 3600*hour[ui1] + 60*minute[ui1] + second[ui1]
bound2 = 3600*hour[ui2] + 60*minute[ui2] + second[ui2]
time = bound2 - bound1
#print bound1
#print bound2
#print time
v = len(x)
for l in range(v):
    if m > ui1:
        if m < ui2:
            y.append(x[m])
            velocity.append(speed[m])
    m = m + 1
#print y
#print velocity
low = y[0]
#print low
for o in y:
    if o < low:
        low = o
#print low
low = low * (-1)

for p in y:
```

```
    y[count] = y[count] + low
    count = count + 1

#print y
area = np.trapz(y)
#print area
#print len(y)
average = (area/samples) * 4.44822
new = savitzky_golay(y,51, 8)
smooth = np.array(new)
maxInd = argrelextrema(smooth, np.greater)
r = smooth[maxInd]
final = []
for value in r:
    if value > 100:
        final.append(value)
#print r
#print final

sumspeed = sum(velocity)
avgspeed = sumspeed/samples
#print avgspeed
meterspersecond = avgspeed*0.277778
pace = 500/meterspersecond
minpace = int(pace/60)
secpace = int(pace - 60*minpace)

distance = meterspersecond * time

watts = meterspersecond*average
newpace = (2.80/watts)**(1./3)*500
minnewpace = int(newpace/60)
secnewpace = int(newpace - 60*minnewpace)
print newpace


#print meterspersecond

#print pace
```

```
#print minpace
#print secpace

#print distance
#remove data that is you do not need printed by adding a comma
#table
print "Data table: position, mm/dd/yy,h,m,s,N,latitude,longitude,speed"
print my_datav
#break
print ""

#time in window
print "time:", time , "seconds"
#average force in window
print "average force:",average, "N"
print "number of strokes:", len(final),"strokes"
print "number of strokes per minute:", ((len(final))/time)*60, "strokes/min"
print "average speed of boat:", avgspeed, "km/hr"
print "approx distance of boat :", distance, "m"
print "approx pace of boat:", minpace, "minute(s)", secpace, "second(s)"
print "indvidual rowers pace:", minnewpace, "minute(s)", secnewpace, "second(s)"
print "watts:", watts, "W"
#print speed

plt.plot(y)
plt.plot(smooth)
```

D.2.1 References for Data Analysis Tool Code:

[14]

D.3 ATMega328 Microcontroller Code Using Arduino IDE:

```
#include <SPI.h>
#include <SD.h>
#include <SdFat.h>


#include <SparkFun_I2C_GPS_Arduino_Library.h>
#include <TinyGPS++.h>
```

```
#include "HX711.h"



I2CGPS myI2CGPS; //Hook object to the library
TinyGPSPlus gps; //Declare gps object


HX711 scale(3, 2);
//HX711 scale(6, 5);
String filename;

float calibration_factor = -7050;
//float scale_val;



void dateTime(uint16_t* date, uint16_t* time) {

  // return date using FAT_DATE macro to format fields
  *date = FAT_DATE(gps.date.year(), gps.date.month(), gps.date.day());

  // return time using FAT_TIME macro to format fields
  *time = FAT_TIME(gps.time.hour(), gps.time.minute(), gps.time.second());
}

void setup() {
  File myFile;
  // Open serial communications and wait for port to open:
  //Serial.begin(115200);

  pinMode(A0,OUTPUT);      //sets up green LED pin
  pinMode(A1,OUTPUT);      //sets up red LED pin
  digitalWrite(A0,HIGH);    //turns off green LED
  digitalWrite(A1,LOW);   //truns on red LED
  delay(5000);            //wait 1 second for everything to boot
  digitalWrite(A0,LOW);    //turns on green LED
```

```
  digitalWrite(A1,HIGH);   //truns off red LED
  //while (!Serial) {
  //  ; // wait for serial port to connect. Needed for native USB port only
  //}
if (myI2CGPS.begin() == false)
  {
     //Module failed to respond.
     //truns on red LED
    while (1){
     digitalWrite(A1,LOW); //failure red light on!
    }

  }
  //Serial.println(F("GPS module found!"));

  //Serial.print(F("Initializing SD card..."));

  if (!SD.begin(10)) {
    //initialization failed!


     digitalWrite(A1,LOW);   //truns on red LED
     digitalWrite(A0,HIGH);    //turns off green LED
    return;
  }

     scale.set_scale();
     scale.tare(); //Reset the scale to 0


boolean t = false;
while(t==false){         //waits for gps time to be accurate before generating file
   while (myI2CGPS.available()) //available() returns the number of new bytes available from the
GPS module
  {

    gps.encode(myI2CGPS.read()); //Feed the GPS parser
  }
```

```
   if (gps.time.isValid())
  {
     if(gps.date.year()!=2080){
       t=true;
       filename = F("test.csv");
       int counter = 0;
       while(SD.exists(filename)){
         filename = (String)F("test") + (String)counter + (String)F((".csv"));
         counter++;
       }

       SdFile::dateTimeCallback(dateTime);
       myFile=SD.open(filename,FILE_WRITE);
       myFile.close();

     }


  }
}









}

void loop() {
          scale.set_scale(calibration_factor);

  writeInfo();

}
void writeInfo()
{
  File myFile;
```

```
  while (myI2CGPS.available()) //available() returns the number of new bytes available from the
GPS module
 {
  gps.encode(myI2CGPS.read()); //Feed the GPS parser
 }
  float scaleval;
  scaleval = scale.get_units();

 if (gps.time.isValid())
 {
      myFile = SD.open(filename,FILE_WRITE);
   if (myFile) {
      myFile.print(gps.date.value());
      myFile.print(F(","));
      myFile.print(gps.time.hour());
      myFile.print(F(","));
      myFile.print(gps.time.minute());
      myFile.print(F(","));
      myFile.print(gps.time.second());
      myFile.print(F(","));
      myFile.print(scaleval);
      myFile.print(F(","));
         if (gps.location.isValid())
         {
            myFile.print(gps.location.lat(), 6);
            myFile.print(F(", "));
            myFile.print(gps.location.lng(), 6);
            myFile.print(F(", "));
            myFile.print(gps.speed.kmph(), 6);
         }
         else
         {
            myFile.print(F("0"));
            myFile.print(F(", "));
            myFile.print(F("0"));
            myFile.print(F(", "));
            myFile.print(F("0"));
         }
      myFile.println();
```

```
  // close the file:
     myFile.close();

    }
    }
}
```

D.3.1 References for Arduino Code Libraries Used:

[5][16][17][18]