OarTracker

Group Members: Braeden Benedict, Joe McGrath, Nick Rasdon,

Brooks Meadowcroft, Abby Greentree

Final Report

May 9th 2019

Table of Contents

1 Introduction

A. Background and Motivation

Rowing is one of the most popular Olympic sports in the United States[1], with over 75,000 active participants and the third largest U.S. delegation to the Olympic Games[2]. Success in the sport requires a unique mix of technique and physical conditioning that is often cultivated over the course of many years. Training for the sport is therefore extremely demanding both physically as endurance is built and mentally as technique is perfected. The aim of this project was to create a device that will improve and simplify the process of perfecting rowing technique.

One of the more important technical aspects of rowing is the coordination of movements between rowers: feathering, roll ups, and catches should all be in sync. Traditionally, technique analysis has been done visually by a coach, but it can be difficult and expensive to find a coach with enough expertise to extract the maximum performance from a boat. Additionally, it is unlikely that a coach would be able to provide useful feedback to multiple people at once without the use of an external device. While there are commercial devices that help with technique analysis by measuring the forces at the oarlocks[3], there is a need for low cost device that can be easily added to an existing setup.

B. Solution

Our solution to the above problem is a system that collects data from the oar and transmits the signals to a bluetooth enabled device. The device then processes the data and gives feedback to coaches via an application with a visual representation of the boat as a whole. Smaller and cheaper than other rowing technique analysis implementations, the OarTracker

device allows rowers to improve their technique without spending large amounts of money or performing extensive modification to existing boats. Additionally, the OarTracker provides information about timing while other devices tend to focus on the forces present at the oarlock.

The OarTracker device met the design goals that were associated with the minimum viable product. It contains an accelerometer and a gyroscope that is able to collect oar orientation data in 3 dimensions. This is connected to a microcontroller, mounted onto the oar, and covered in plastic so that it is waterproof; a preliminary version of waterproofing in the form of plastic wrap and duct tape was used in this version, but in the future a waterproof enclosure may be created.. The device is small enough to enable data collection without interfering with the motion of the stroke. The microcontroller is bluetooth enabled with a radius of roughly 30 feet, allowing data to be sent to a coxswain who is also in the boat.

Data analysis to detect a catch - the part of the stroke where the blade enters the water - is performed on the bluetooth enabled microcontroller. Once a catch is detected, a catch notification is sent over bluetooth to a smartphone, where a custom application determines the relative timing of the rowers. The GUI shows a visual representation of each rower so that the coxswain can get real-time feedback on catch timing, allowing the coxswain to immediately correct stroke timing.

Overall, the Row Tracker system is able to collect and analyze stroke data that can lead to improvements of the overall efficiency of the boat. It does so in a way that is less intrusive and less expensive than currently existing commercial implementations while still providing useful feedback to the rowers.

2 Detailed System Requirements

In order for the system to function properly, numerous requirements were met. The devices attached to the oars must be light enough so as to not hinder or alter a rower's natural stroke; these devices must also be waterproof and capable of lasting through a typical session (2 hours). To measure the rower's technique, the devices must be able to reliably assess when a catch occurs. This required that the devices contain an inertial measurement unit (IMU), which provides data related to the orientation of the oar. The IMU must be able to provide accurate data regarding both the acceleration of the oar in three dimensions and the orientation of the oar in three dimensions. This data then must be collected and processed by a Bluetooth low-energy (BLE) microcontroller.

The BLE microcontroller must be capable of receiving accelerometer and gyroscope data from the IMU via I2C at a rate fast enough to discern the timing of the stroke - at least 500 Hz, which is the maximum sample rate for the IMU accelerometer data. The BLE microcontroller must be able to analyze this data and notify the cell phone application when a catch occurs via a bluetooth characteristic. The connection should function over a range of 30 feet so that a coxswain in the boat can access the feedback from the system.

The cell phone application must be capable of receiving data from at least four oar devices and analyzing the data to determine the difference in catch timing between rowers. A GUI should show the results of each stroke, indicating whether a rower's catch is on time, late, or early when compared to the lead rower.

The devices should be powered by lithium-polymer batteries which can be recharged via standard Qi wireless charging to accommodate their waterproofing. This requires that the devices

contain a Qi-receiver which can reliably communicate with any Qi-transmitter. This Qi-receiver must also work alongside a charge controller in order to regulate the charging of the battery. In addition to this charge controller, the battery must be connected to a DC/DC converter to guarantee a 3.3V output to the rest of the device.

3 Detailed Project Description

A. System Theory of Operation

The OarTracker system is comprised of an ensemble of subsystems that work together to provide stroke feedback. These systems are the data collection system, the data analysis system, and the wireless charging system. The inertial measurement system and wireless charging system are entirely contained on the OarTracker device. Each OarTracker device is comprised of one PCB connected to a lithium-polymer battery and a charging coil. The schematic and board designs are shown in Appendix A.

The inertial measurement system is composed of an inertial measurement unit, the Hillcrest BNO080, and a bluetooth enabled microcontroller, the TI CC2640R2F. The BNO080 collects accelerometer (m/s^2) and gyroscope (rad/s) data in the x, y, and z directions and sends it over I2C to the CC2640R2F. The CC2640R2F parses the data sent by the IMU and uses the values to control a state machine. This state machine looks at various thresholds in the accelerometer and gyroscope values to determine parameters of the stroke such as the orientation of the oar, either feathered (parallel to the water) or squared (perpendicular to the water), whether the rowers are using feathered or square blades, and whether a catch has recently occurred or not. If the accelerometer or gyroscope values meet the threshold requirements for a catch, the state machine will also update a binary bluetooth GATT characteristic that is set to notify.

The iPhone application continuously looks at the catch characteristic from multiple devices. One OarTracker device is designated the lead; all other catches are determined to be early or late in comparison to the catch seen on this device. A threshold is set at 50 ms: to be

early or late, a catch must come either 50 ms before or 50 ms after the catch seen by the lead device. The results of each catch are shown to the user of the application, allowing for real time feedback to be given to each rower.

B. System Block Diagram

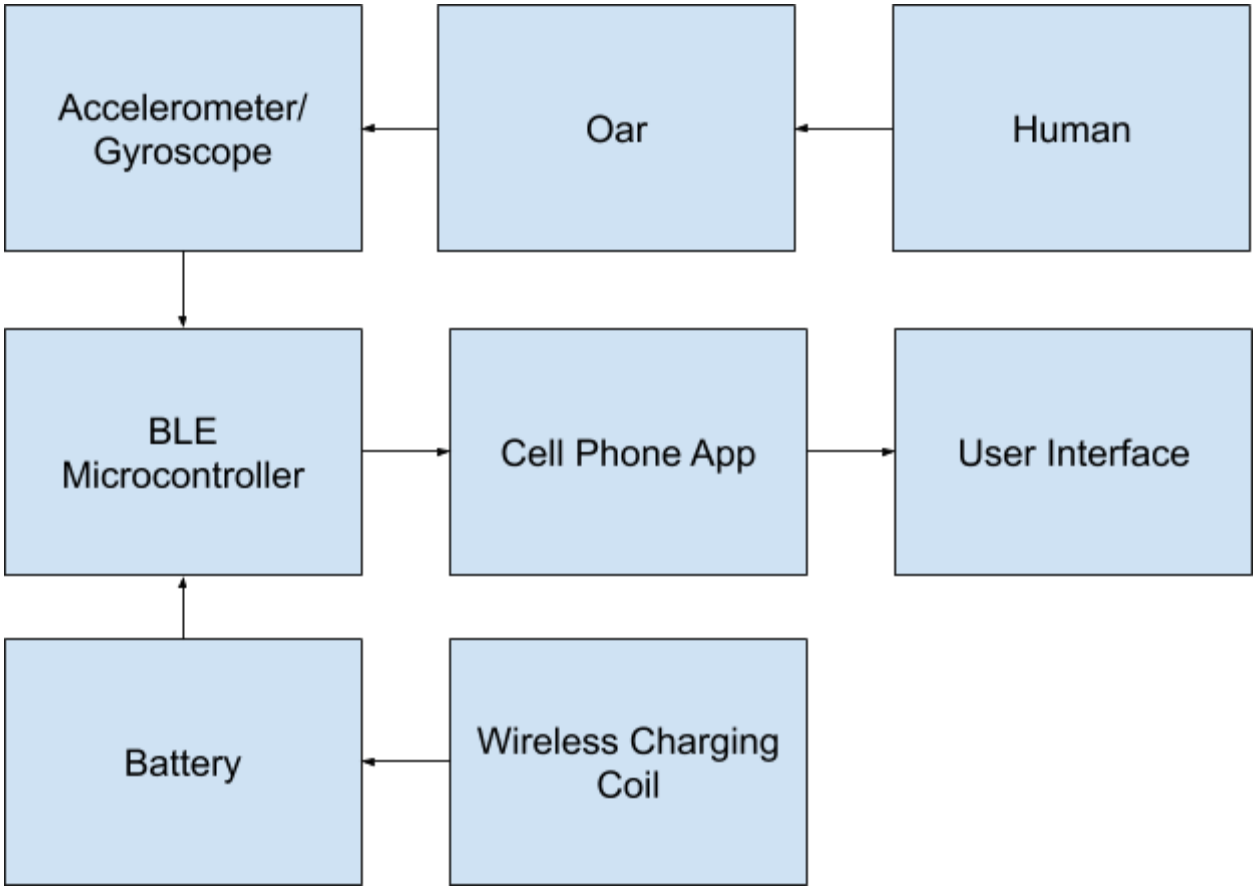


Figure 1: System Block Diagram

C. BLE Microcontroller and IMU

The Hillcrest BNO080 collects accelerometer (m/s^2) and gyroscope (rad/s) data in the x, y, and z directions at a rate of 500 Hz and 400 Hz respectively. This inertial measurement unit

was chosen because it is able to communicate over I2C and has low power consumption, the latter of which is extremely important for a device that should last the entire length of a rowing practice. Data is sent over I2C to the CC2640R2F at a rate of 400 kHz, accommodating the maximum IMU sample speeds.

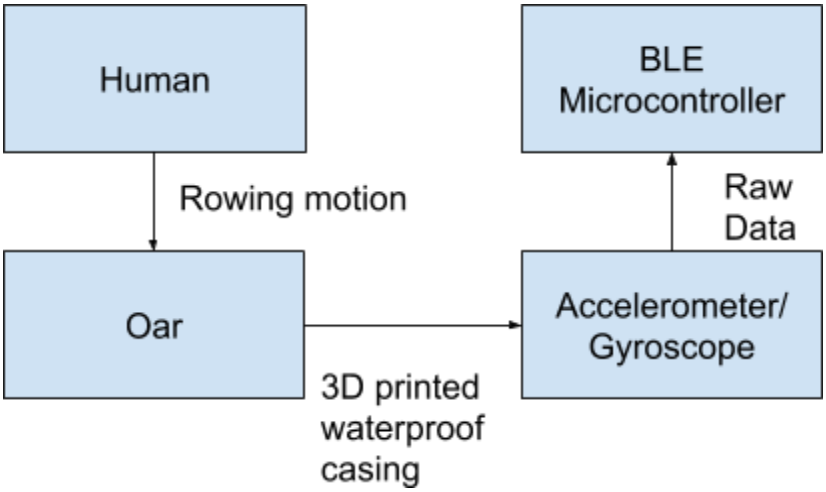


Figure 2: Flow Diagram for iPhone Application

The CC2640R2F serves as the I2C master and the BNO080 as the I2C slave, with the address 0x4A. Communication occurs using Hillcrest's Sensor Hub Transfer Protocol (SHTP), which can be used with any of the standard serial communication protocols and consists of a four byte header followed by the data. The header contains two bytes corresponding to the length of the data being sent followed by a byte designating the communication channel and a byte designating the sequence number. There are six available communication channels, including the command channel (0) and a sensor report channel (3) that are used by the OarTracker device.

Communication with the IMU begins with a setup process that allows the master to specify the desired measurements and sample rate by sending a command over the command

channel. In this step, the master device checks to determine if the IMU is active by requesting the device identification number and comparing this the accelerometer and gyroscope are enabled with the maximum sample rate. To read the sensor reports from the sensor report channel, the CC2640R2F continually polls the IMU for four bytes of information (the SHTP header). If no data is available, the IMU will return zero for all bytes, but if a report is available, the IMU will return the header information. The system supports partial reads, so once the header information is acquired and the CC2640R2F knows the number of incoming bytes, the read can be performed again for the number of bytes available. Once the read is complete, the CC2640R2F can parse the report and update the accelerometer and gyroscope values. Values from the IMU are in the Q number format, which is a fixed point format that is converted into a floating point format when the report is parsed.

The library[4] used for communication with the BNO080 device was originally written to work with Arduino. It was updated to work with the functions provided by TI for performing I2C on the CC2640R2F by modifying the API used to access I2C communication. Additionally, modifications were made to the library to improve performance in order to avoid processing delays. One such modification was the hard-coding of the conversion factor between the Q number format and floating point, replacing the costly exponentiation used previously.

The CC2640R2F has been set up to perform I2C and state machine updates as two separate periodic tasks, which are scheduled by the RTOS included on the chip. The I2C task occurs as fast as possible, with a periodic clock set to 0.01 ms. This task polls the IMU for data and updates the accelerometer and gyroscope values as they become available. The state machine

update task runs every 2.5 ms and is responsible for updating the state machine based on various thresholds that will be discussed in the data processing subsection.

The CC2640R2F updates a bluetooth GATT characteristic with any changes to the catch state, allowing the iPhone application to view the results of the data analysis. Another important task given to the CC2640R2F was to output a 32kHz clock signal for the IMU to use in order to limit the number of different crystals present on the board. A built in TI function was used to output the clock signal to the desired pin.

D. Data Processing

Accelerometer and gyroscope data for development of the analysis algorithm was collected both from a static rowing simulator and from multiple devices on the water. This was accomplished using an Arduino Uno with two IMU development boards connected over a single I2C bus. Using a single microcontroller eliminated issues of data synchronization, but this required I2C to function over several meters at its maximum frequency. Fortunately, I2C exceeded its specifications and did work at this distance. After being received by the Arduino, data was transmitted to a laptop over a serial terminal and saved. Accelerometer and gyroscope data was parsed, graphed, and analyzed in MATLAB. An algorithm to identify the catch, feather, and square-up was empirically developed from this data and fine-tuned. Finally, the algorithm was translated to C and implemented on the BLE microcontroller.

The analysis program is controlled by a number of state machine variables. Transitions between states are controlled both by timing and by data from the three most relevant IMU sensors: X gyro, Y accelerometer, and Z gyro.

At any point in time, the rower can either be rowing ‘on the feather’ which means they are rotating the oar 90 degrees and back when they remove it from the water, or they can be rowing ‘on the square’ which means they are not rotating the oar when it is removed from the water. If rowing on the square, there will be no feathering or roll up to identify, and the accelerometer/gyro data is significantly different. Rowing on the square is not done in competition, but it is a common drill during practice and we wanted to ensure our device would function under that condition. This state is tracked using the variable “feathering_state.” Transition to the feathering state occurs when a feather or a roll up motion is detected. Transition to the square state occurs if a feather motion has not occurred for 1.4 times the length of the last stroke.

In addition to keeping track of if the rower is rowing on the feather or on the square, the program also has a state machine variable “feather_or_square_blades” to keep track of the current orientation of the oar, which can either be feathered (parallel to the water) or squared (perpendicular to the water). This can be a confusing distinction, but it is necessary. For example, a rower can be ‘rowing on the feather’ while simultaneously having an oar which is ‘squared up.’ Transitions between the feathered and squared states is controlled by multiple values, but the most obvious indicator is a sharp negative X gyro peak for transition to the feather and a more gradual positive X gyro peak for transition to the square.

The state variable “ready_for_new_catch” is controlled by a counter which begins when a previous catch is identified. It ensures the program only identifies another catch after a specified length of time, 0.5 times the length of the last stroke. This ensures that only one catch event is

identified per actual catch event and it reduces the possibility that a false catch could be identified.

If rowing on the feather, a catch is identified only by looking at data from the Z gyro. The program marks a catch when it sees a negative Z gyro value below a threshold, a sharp dip occurs in the Z gyro value while it is below that threshold, and the Z gyro value subsequently increases for four consecutive readings. If rowing on the square, the same technique is used, with the addition of a requirement that there be a perturbation in the Y accelerometer values. For the sake of clarity, the Y accelerometer data has been smoothed using a moving average filter in the figure below, but this is not used in the data analysis program and the perturbation in the Y acceleration while catching on the square is obvious.

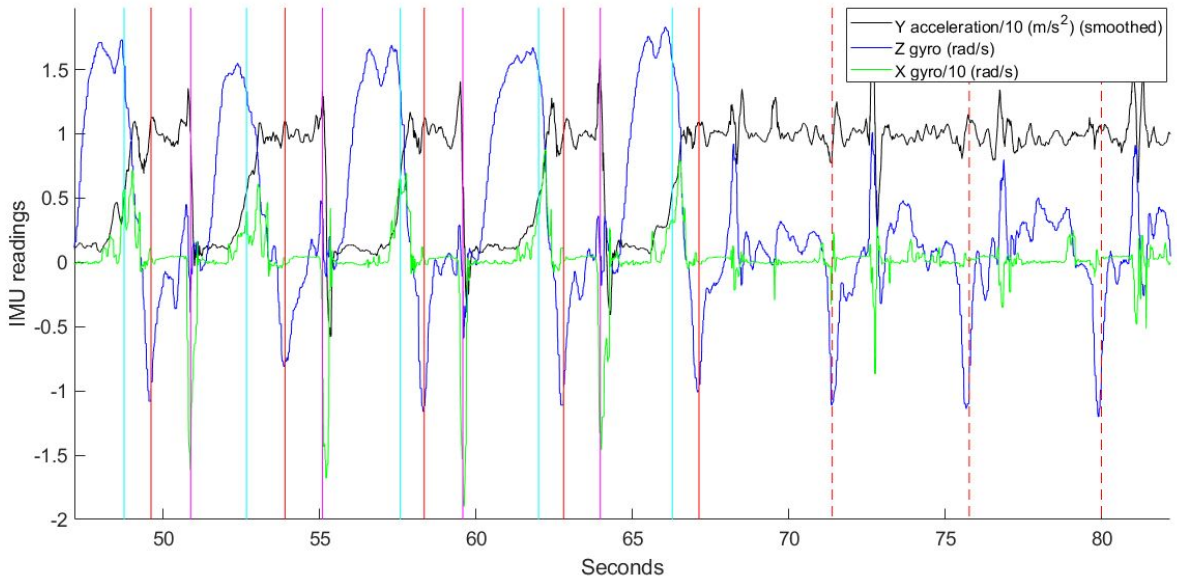


Figure 3: Plot of data output from MATLAB catch identification program. Roll ups are represented by cyan vertical lines, feathers by magenta, and catches by red. Solid red lines indicate a catch while rowing on the feather, while dotted red lines indicate a catch while rowing on the square.

While our catch identification algorithm has worked well, we believe there are likely some changes which could be made. We currently have determined the hard-coded threshold values for state transitions based on recordings from several rowers on the water, and these values do work for the data from each one. However, these rowers are all of similar experience and row with similar technique. These values may be different for different rowers, and it would be useful to incorporate some form of learning into the devices such that they could automatically change the threshold values depending on the style of the current rower.

E. Cell Phone App

The key user interface for the OarTracker product is a cell phone application. The mobile application was written in Swift 4 on Apple's XCode IDE and it uses Apple's Core Bluetooth Library. Below is a flow diagram for the OarTracker mobile application.

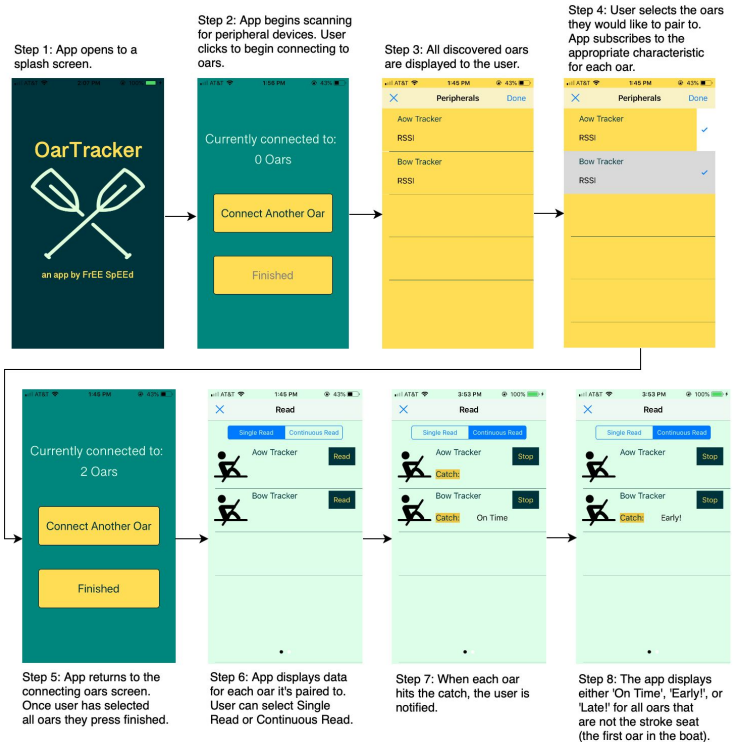


Figure 4: Flow Diagram for iPhone Application

When the user opens the mobile application, the program begins to search for peripheral devices in the area. In order to only provide relevant peripherals to the user, the program will only scan for devices that are advertising a specific, pre-determined UUID that all OarTracker peripheral devices advertise. The user is prompted to 'Connect to Oars'. Clicking this option progresses that application to the next page that displays a list of all peripheral devices in the area, identified by their device name. The user taps to select the devices they are interested in and then clicks 'Done' to return to the previous screen. The user should the device they want to set as the stroke seat (the first oar in the boat) first. The devices the user wishes to connect to are stored in an array of CBPeripheral data type. Once the user is done with their selection, they click 'Finished' and the application progressed to the communications pages, which lists each connected device along with options for reading information from the device. The first device listed is the stroke seat and will set the timing. All other oar devices will be compared to the state of the first device to determine if they are on-time, early, or late.

When 'Finished' is selected, the program will discover the specific characteristics that are available to read and write from on the peripheral. The application has been program to search for and subscribe to a specific characteristic that is transmitting the catch. The communications page of the application currently has a interface to read from the peripherals, as the project was focused solely on retrieving information from the peripherals. Writing to the devices is equally possible and could be included in a future iteration of the application if it would provide added value.

The user can choose from two types of read modes: single-read and continuous-read. Single-read mode allows the user to click the 'Read' button and obtain the latest value of the

Catch characteristic. Continuous-read mode prompts the user with ‘Start’ and ‘Stop’ button to begin or end reading on each individual device. In continuous-read mode the program with cycle through the devices, reading from them consecutively beginning with the first oar. When the catch characteristic is updated from 0 to 1, the app will display a yellow ‘Catch’ label for that oar. If that catch received before the stroke seat has hit the catch, then the app displays a ‘Early!’ label, it similarly will display an ‘On-Time’, or ‘Late!’ label when appropriate. There is a 50 millisecond threshold for being early or late, meaning that any oar can be up to 50 milliseconds late or early before being labeled as such. This is to reflect the accuracy of actual oars on the water, that will never be exactly in time.

F. Wireless Charging Unit

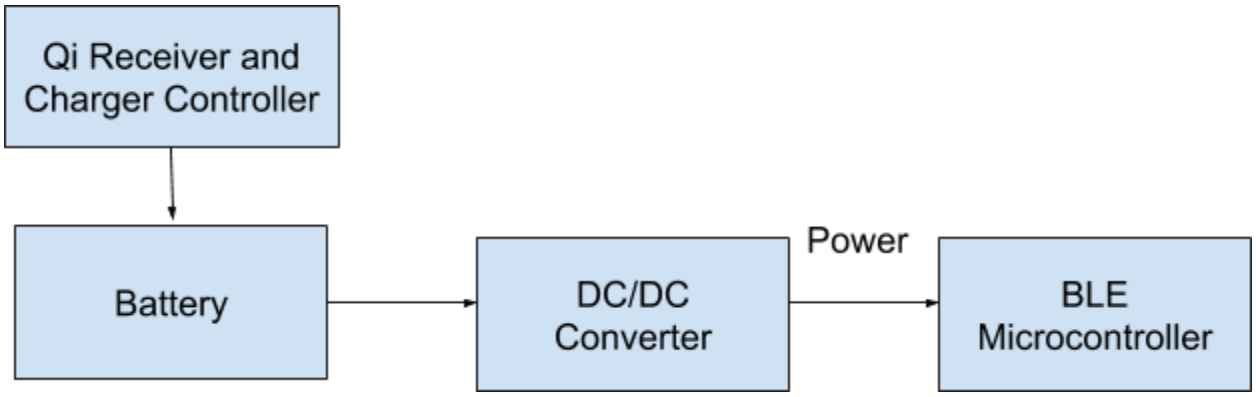


Figure 5: Flow Diagram for Wireless Charging

Show above is the block diagram for the wireless charging unit. The bq51050b chip by Texas Instruments was chosen to be the cornerstone of the charging unit because it is a Qi-compliant wireless power receiver integrated with a Li-Pol battery charge controller. It provides an output of 4.2V to the battery, which has a rated capacity of 190 mAh. While the device is charging, the charge controller turns on a red charging LED, shown below. The output of the battery is connected to the tps62172; this chip is a synchronous step-down DC-DC

converter. It takes the nominal 3.7V output of the battery and steps it down to a constant 3.3V. This 3.3V output from the converter is then used to power the BLE microcontroller.

Figure 6: Board before (left) and during (right) wireless charging

We chose to use the Hypercell HPL402323 battery, which has a 190mAh capacity, equivalent to 700mWh at its nominal voltage. With an established BLE connection and the status LED enabled, our device draws 24.4mA at a battery voltage of 4.08V, corresponding to 100mW. This results in an expected battery life between 7.0 and 7.8 hours, depending on if the calculations are performed using mAh or mWh. This battery life is more than sufficient to power the device for a typical practice, which lasts around 2 hours. With the green status LED disabled, battery life is increased to between 10.2 and 11.3 hours.

4 System Integration Testing

A. Subsystem Testing Description

Each individual subsystem and interface was rigorously tested before final system integration. Subsystem testing was separated into on board components and communication, charging board, custom board design, and cell phone application.

Initial development of code for components that were used in the final board design was done on the TI CC2640R2 LaunchPad. This development board included the CC2640R2 bluetooth enabled microcontroller that was used on the final board and the XDS110 programmer that was used to program both the development and final boards. Programming was done in TI's Code Composer Studio (CCS), which was able to include all necessary libraries and board definitions related to the operation of the microcontroller.

The BNO080 inertial measurement unit (IMU) and data analysis program were tested first to ensure that it was possible to accurately and repeatedly identify the catch in a stroke. This was done using an Arduino Uno because a library for interfacing with the IMU was already available. It was found that data analysis could successfully identify when a catch had occurred. Subsequently, the I2C interface between the IMU and CC2640R2F chip was tested to ensure that the modified version of the BNO080 library would function correctly. While testing the version of BNO080 library that had been modified for the CC2640R2F, a bug was found that would cause I2C to stall if the return value from the I2C transfer was checked. After this bug was fixed by removing the check of the return value the I2C connection worked and was tested by outputting accelerometer and gyroscope values over a serial connection from the microcontroller to a computer. The serial connection was then removed and the values received from the IMU

were instead used to update bluetooth characteristics. A third party BLE app, LightBlue for iPhone, was used to test that the IMU data could successfully be transmitted over bluetooth.

For the testing of the wireless charging subsystem, a small board containing only the components necessary for wireless charging was created and tested to ensure that the correct voltages (3.3V) were output and that the battery could be charged. After confirming the success of each individual component and confirming the wireless charging board worked as expected, the final board could be designed and tested.

The initial testing of the board followed a custom hardware startup procedure recommended by TI. First, the connection between the XDS110 programmer and the custom board was tested using CCS. Next, the simple peripheral bluetooth program provided by TI was loaded onto the board to ensure that bluetooth was operating as expected. Once this was confirmed, a custom board header file was used to define the pins needed for I2C and other operations, such as the 32 kHz clock output to the IMU. Finally, to complete the initial testing of the custom board, a simple program was loaded to blink the LED on the board.

The bluetooth capabilities of the final board were initially tested using a third party BLE app to confirm the transmission range and ability to transmit processed data from the IMU. The IMU initially did not provide any data to the microcontroller. It was found that the microcontroller pins connected to the reset and boot inputs on the IMU would momentarily go low during startup, causing the IMU to enter bootloader mode. This issue was fixed by setting these pins to high impedance rather than 1. Initially, the BLE range of the custom board was only about two feet, a very small fraction of what was expected. Ultimately this issue was determined to be mainly caused by a software error that was incorrectly setting the transmit

output power. Additionally, if the board were to be redesigned, a more detailed consideration of the antenna matching network and consistency in the trace leading to the antenna could improve the range. Ultimately, fixing the software issues provided range sufficient (30 feet) for the requirements of the project.

The cell phone application was independently tested with the development boards. A TI example program that takes button inputs and updates a characteristic value with 0 or 1 was used to simulate the expected 0s and 1s of the catch characteristic. The first tests occurred with one development board and were designed to test the ability of the app to connect and read data from a bluetooth characteristic. Once the ability to read from one board was confirmed, the application was tested with the two available development boards. Additionally, the threshold for determining whether a catch came early or late was tested at this time to ensure that it was set to a reasonable value (50 ms).

Once the app performance was tested, the full system was integrated. Final tests were performed using the OarTracker app and custom board. These tests were done both on the water and on land. On land testing was done with a real rowing oar and a moving seat that simulates the movement of a rower on the water. The devices were mounted to the oar and the mobile application was used to test success rate of identifying the catch. At this time, the four devices available were tested simultaneously with the application to ensure it could scale to larger crews. Two on water tests were performed with the University of Notre Dame's Men's Rowing team and catch timing from multiple devices was obtained.

B. Comparison with Design Requirements

The requirements for the wireless charging subsystem were to reliably communicate with the Qi wireless charger, regulate the voltage used to charge the battery, and regulate the voltage supplied from the battery. These requirements were met first with the test version of the charging board and later with the wireless charging subsystem present on the completed custom board. All voltages were tested and found to be at their expected levels.

The requirements for the BLE microprocessor and IMU subsystem were to track movement in three dimensions, send raw data over I2C from the IMU to the microprocessor at the maximum sample rate, and be contained in a waterproof enclosure. A waterproof enclosure was not built, but plastic wrap was used to build a prototype that functioned as desired. The other two requirements were tested simultaneously when it was confirmed that accelerometer and gyroscope data could be sent over I2C. The system was successfully able to send data in three dimensions at the maximum sample rate. Additionally, the specified bluetooth range was at least 30 feet, which was achieved after some modifications to the power output.

The data processing subsystem required a bluetooth interface able to receive data from four oars, a signal processing algorithm able to recognize various parts of a stroke and determine timing differences, and a cell phone application able to display the results to the user. The bluetooth interface was tested with four devices (two development boards and two custom boards) to demonstrate that it is possible for the system to receive data from four oars. The signal processing algorithm was tested first on land and then on the water, as mentioned previously. The ability to determine timing differences was successfully tested first with the buttons on the development board and then with two different devices. Finally, the application was successfully able to show the results of the catch to the user, as shown in Figure 4.

5 User Manual

A. Installing the Product

To install the product the user needs to complete two main steps: (1) download and run the mobile application, (2) mount the boards correctly on each oar.

The mobile application is not currently available on the Apple App Store, until such time the application must be downloaded via XCode. The user needs to use a Mac Computer running the latest operating system and must download XCode, Apple's IDE for application development. Then, the user should download the OarTracker XCode Project from the team website and open the project. They should connect their desired mobile device, which must be an iPhone, and click the run button to install and run the application on their phone.

Batteries should be connected to the peripheral devices using the labeled battery connection. The user must download Code Composer Studio and the relevant board files from the team website in order to program the board as a Port or Starboard Oar. To mount the peripheral device, it should be placed on the oar shaft, face up, while the blade lies flat. The antenna on the board should point towards the blade on a starboard oar and towards the rower on a port oar. The product does not currently include a waterproof case, so the user should use plastic bags, saran wrap, or other items to waterproof the device. Below is an example of two OarTracker devices mounted on starboard oars using saran wrap and tape to secure the devices.

Figure 7: OarTracker device mounted on oars.

B. Using the Product

To use product, the user navigates through the mobile application. They should open the app and tap the launch screen, then they can follow the prompts to discover and connect to the peripheral devices, enter their preferred reading mode, and view the timing of their oars. If the OarTracker devices are properly initialized, the LED on the device will turn on.

C. Troubleshooting the Product

If the product is not working as expected, there are a number of steps the user can take to identify and resolve the issue. If the mobile application crashes, that is because one or more of

the peripheral applications have disconnected. The user should check if the devices have moved out of range and if they are still connected to charged batteries. If the mobile application is running well but the user is not seeing catches, the problem is most likely with the peripheral device. In this case, the user should reset the devices using the physical reset button to ensure I2C communication is working properly, and check the device orientation on the oar. Improper orientation will lead to inability of the data analysis program to properly identify the catch.

6 To-Market Design Changes

A future improvement would be to develop a waterproof case that encompasses the boards, the batteries, and the charging coil. The case along with adding the mobile application to the Apple App Store would allow the product to ‘ready-for-use’ by the user. Other improvements fall into two main categories: (1) improving robustness, (2) increasing information available to the user.

Additional improvements to the main code of both the board and the mobile application could be made to improve the robustness of the product. The data analysis program currently detects whether the rower is rowing ‘on the feather’ or ‘on the square’ but could be improved to automatically detect if the rower is rowing port or starboard as well. This removes the need for the user to ever have to reprogram the board. The main board program could also detect long stretches of inactively or times when the battery is being charged and switch to a low-power mode that does not take in data from the IMU in order to conserve battery power. Improvements to the board layout could enhance the BLE range of the product, which would be essential for use in a eight person boat. The mobile application could be improved to provide more intelligent error messages to help the user troubleshoot problems that arise. Additions to the mobile application that could prevent errors are: a disconnect from specified peripherals feature, ability to switch which oar device is the stroke seat, ability to write to the devices for configuration changes or external resetting of the boards.

Currently, the expanse of data being collected from each oar is being processed with the specific goal of identifying catch timing. Several other key pieces of information could be extracted from this data to further add to the value of the product. Some of this information

includes: feather and square timing, strokes per minute, sharpness of each catch. The mobile application would also need to update both the user interface and the read requests to accommodate the additional information. The mobile application currently displays all information in real time but does not store any information about any of the oars. If the application collected either the raw data or data events about each oar and stored that information either within the application or on the cloud, this would allow the rowers and coach to view information from overall training and identify larger patterns. These additions would need to be balanced with processing speed of the board and mobile application so as not to decrease the real-time performance of the product.

7 Conclusions

Ultimately, our project can be widely considered a success. We started off with lofty goals about what our user interface would be able to do but other than that we were able to accomplish exactly what we set out to do. We were able to create a low cost light wireless device to be attached to an oar that is able to track and identify multiple parts of the stroke. Although we wish it could have done better out on the water, it was still encouraging that we were able to see catches at a somewhat normal rate.

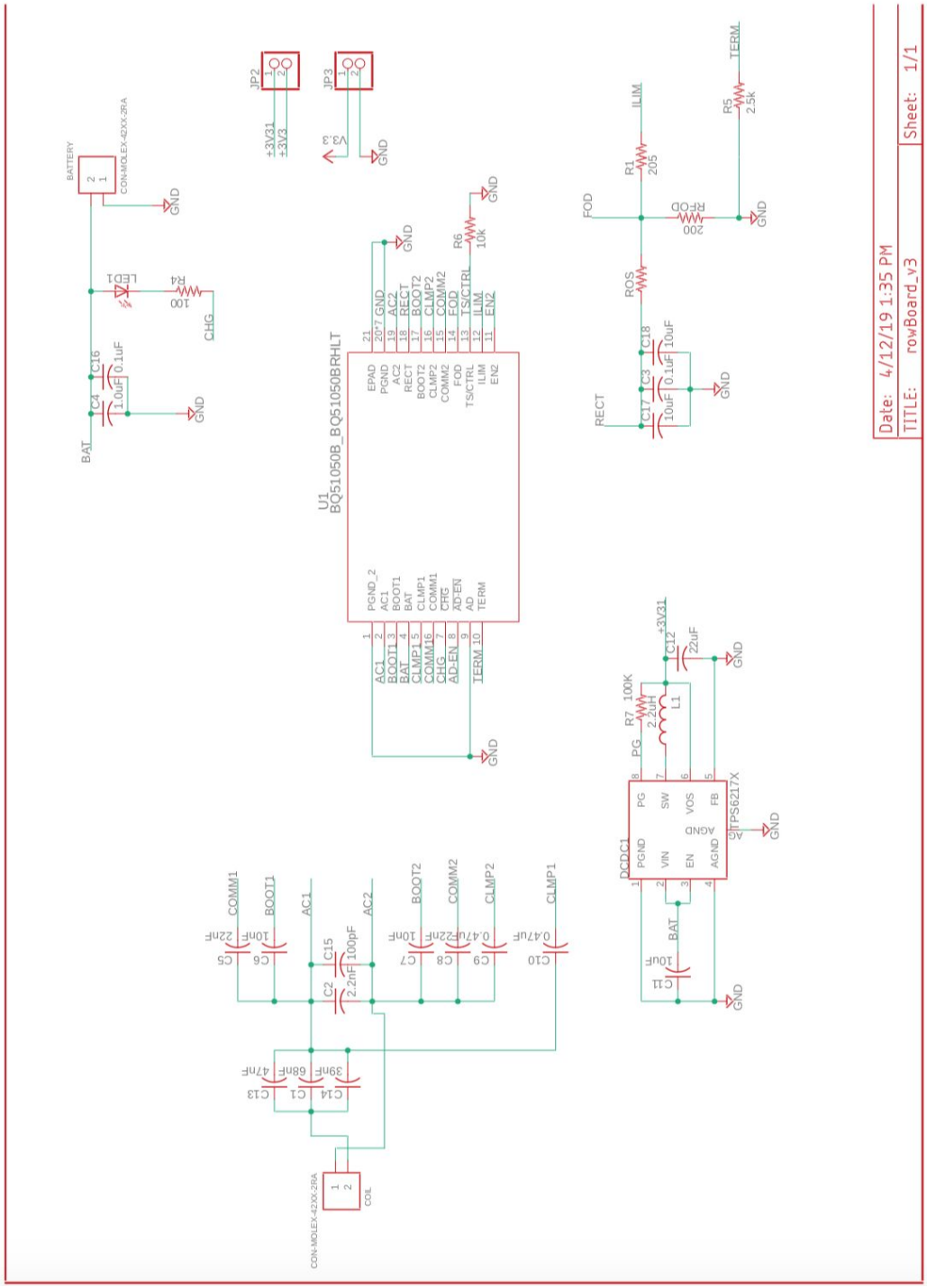
If we were to go forward with this project there would be many improvements that we would like to make. First would be to enhance the phone application to be able to display multiple parts of the stoke in a more data intensive user interface to get more immediate feedback on each rower's stroke. Additionally, a primary improvement for the product to be viable on the market would be to develop a plastic casing for it to be totally waterproof. Another thing we noticed when having people test our device during demonstration day is that the average person does not apply the same acceleration to the oar that a well practiced rower does so it would be beneficial to have different settings that have different thresholds in order to see catches more accurately.

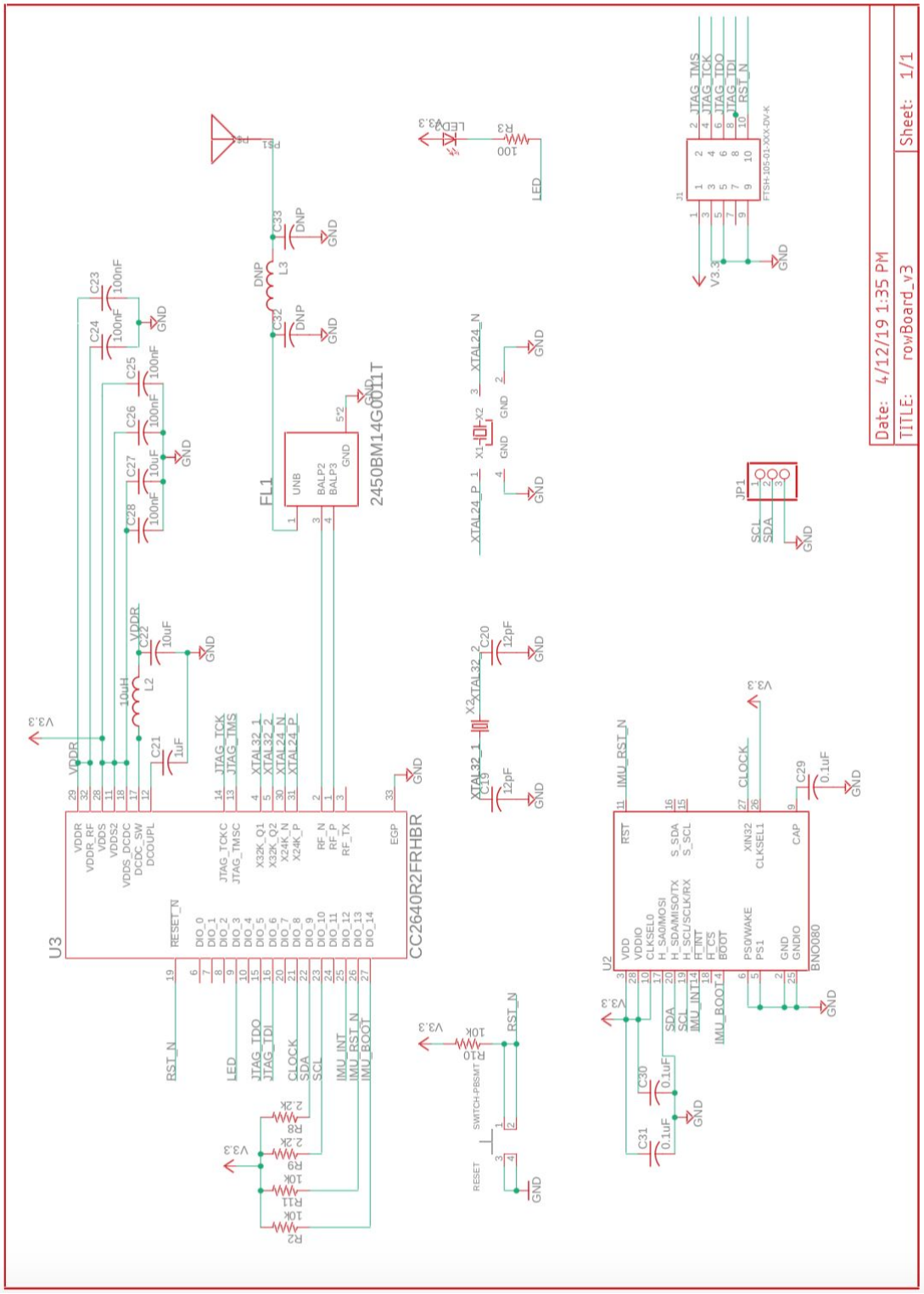
The subsystems of our design demonstrated a broad spectrum of things we learned over our four years and our time in Senior Design. We used knowledge from Embedded Systems and from the first semester of Senior Design in our board design and use of I2C to program the board. In addition, we were able to teach ourselves outside of class the basics of app development and bluetooth communication on the app and I2C side. We also were able to effectively design a wireless charging portion of our board. All of these things together

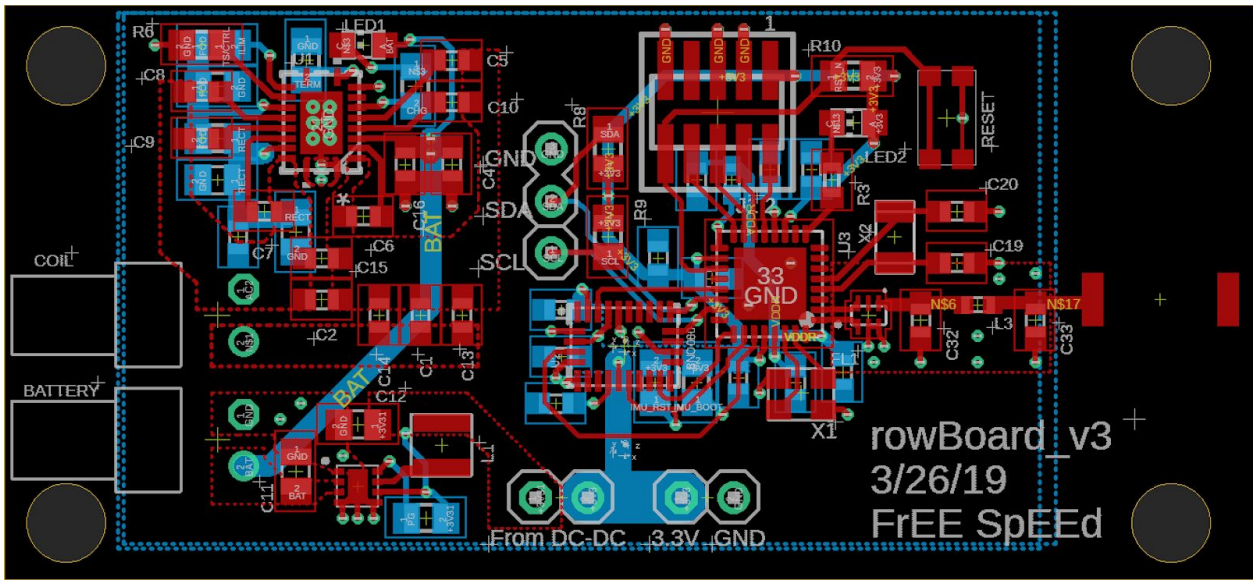
demonstrates how far we have come in our four years in electrical engineering and we are proud of the product that we were able to create.

8 Appendix

A. Board Schematic and Layout




U3
CC2640R2FRHBR
2450BM14G0011T
FL1
L2
L3
U2
BNO080
J1
JP1
X1
X2
SWITCH-PBSMT
LED
Date: 4/12/19 1:35 PM
TITLE: rowBoard_v3
Sheet: 1/1


COIL
BATTERY
GND
SDA
SCL
BAT
LED1
LED2
RESET
rowBoard_v3
3/26/19
FrEE SpEEd
From DC-DC
3.3V
GND

9 References

[1] http://www.usrowing.org/about-us/

[2] http://www.usrowing.org/rowing-quick-facts/

[3] http://www.peachinnovations.com

[4] https://github.com/sparkfun/SparkFun_BNO080_Arduino_Library