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]