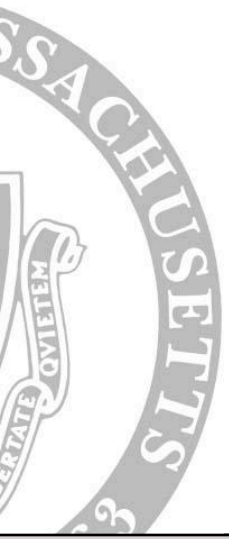
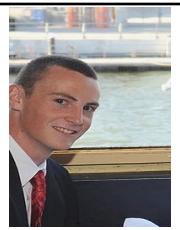
Comprehensive SWAG



Comprehensive Small Wearable Analyzer of Gait

Meet The Team



Scott Clark - CSE

Eric Grimaldi - EE

Alex Fanolis - EE

Jack Shaffery - EE

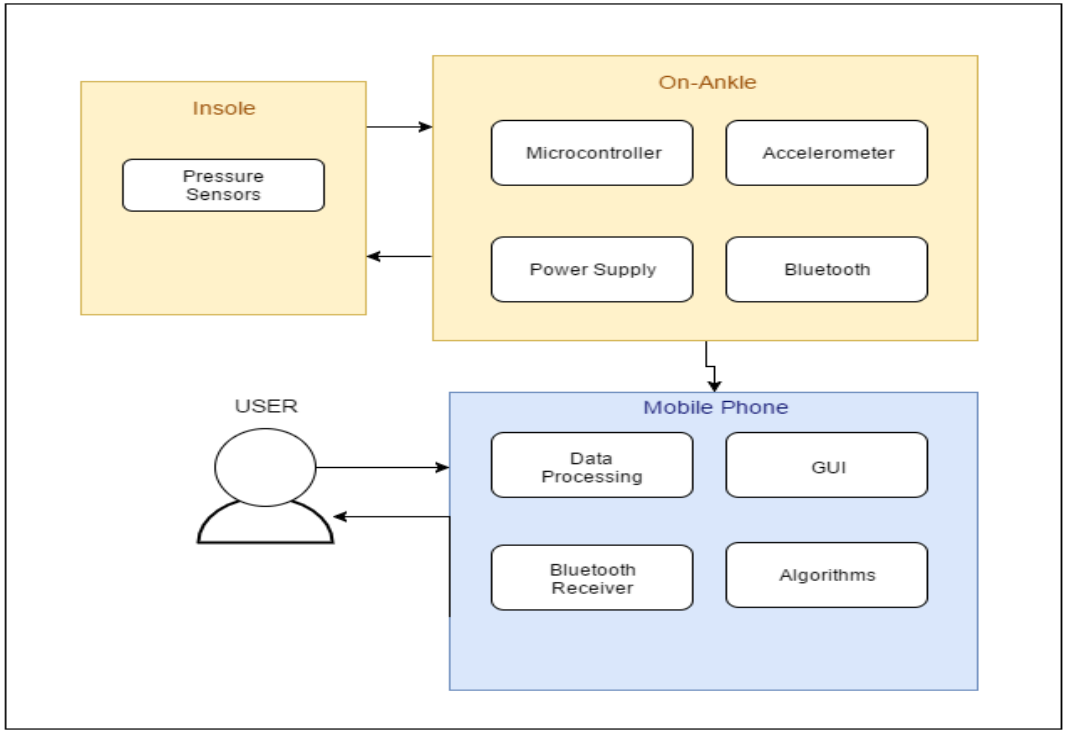
Advisor - Professor Aura Ganz

Gait Analysis

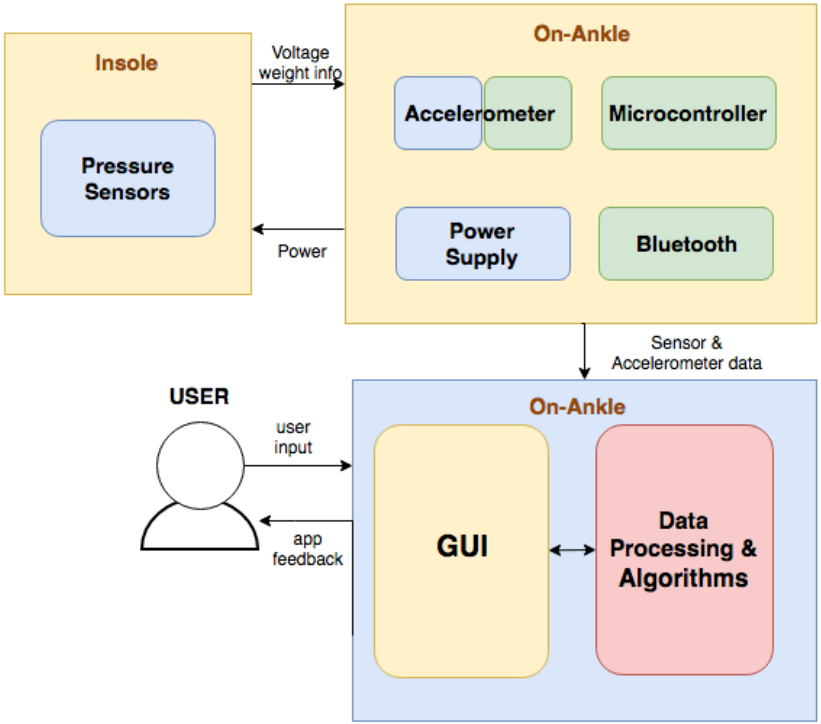
- Motivation
  - Predict diseases, prevent disorders, rehab from injuries
- Current Gait Analysis
  - Analysis usually requires a doctor's referral
  - Limited variable terrain testing
  - Appointments are expensive
  - Analysis can take anywhere from 30 minutes to 3 hours
- Our Solution
  - Completely portable
  - Real time analysis
  - Much cheaper than current devices

Our Solution

Previous Block Diagram



Redesigned Block Diagram



Promised MDR Deliverables

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers and gyroscopes
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface

MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface

MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface

MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface



MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface




MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface





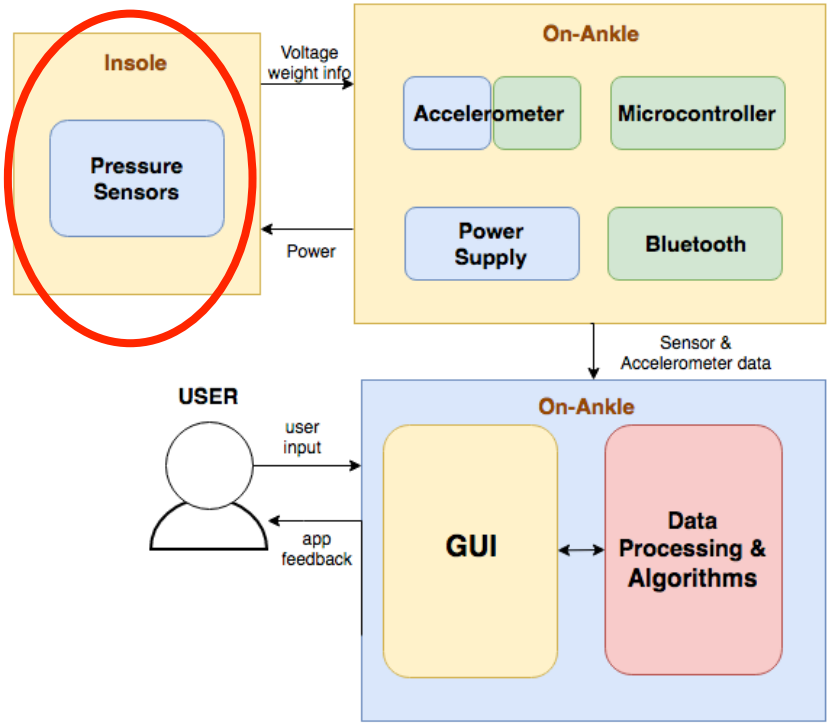
MDR Checkoff

- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface
- Functional algorithm working

MDR Checkoff

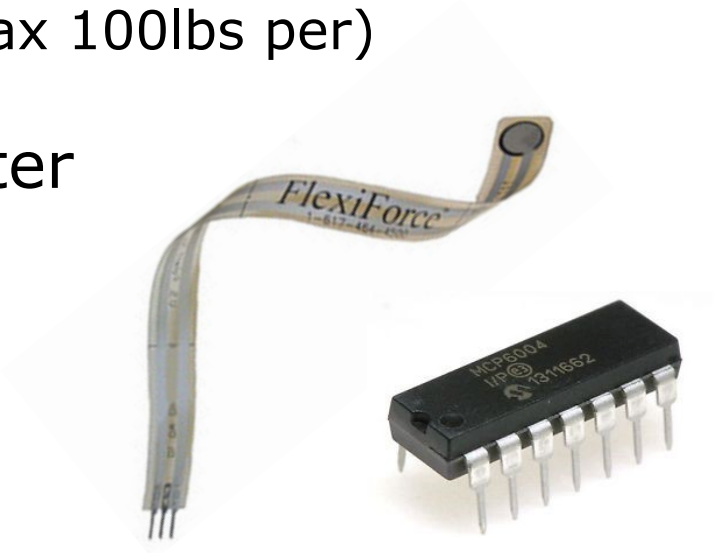
- Identify and purchase required hardware:
  - Microcontroller
  - Pressure sensors
  - Accelerometers ~~and gyroscopes~~
- Establish communication between sensors and microcontroller
- Identify specific power source
- Research algorithms for specific gait parameters
- Rough draft of graphical user interface
- Functional algorithm working
- Early stages of app developed

Insole


Insole
Pressure Sensors
Voltage weight info
Power
On-Ankle
Accelerometer
Microcontroller
Power Supply
Bluetooth
Sensor & Accelerometer data
USER
user input
app feedback
On-Ankle
GUI
Data Processing & Algorithms

Sensors

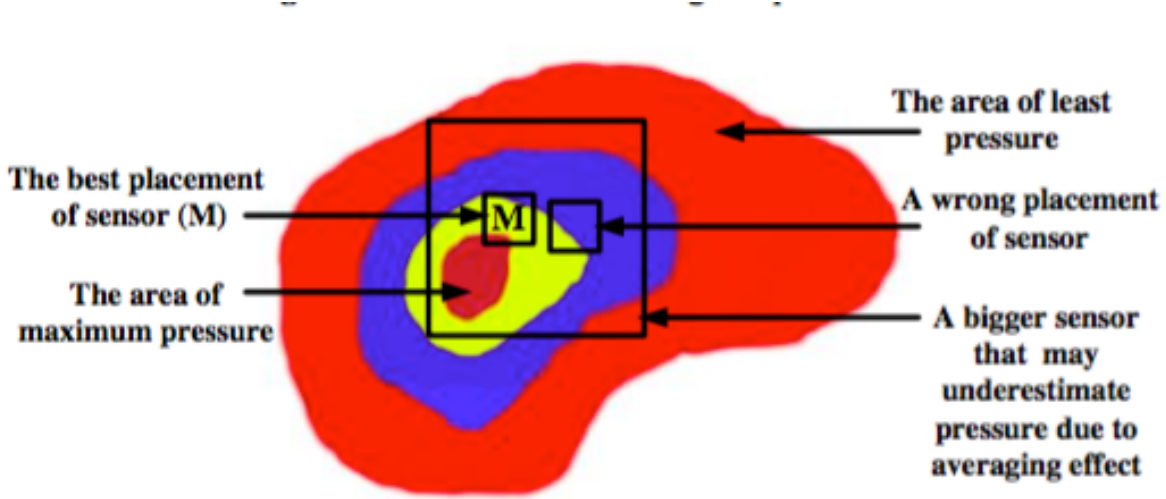
- Tekscan Flexiforce A201 Resistive Sensor
  - Low physical profile
  - High weight tolerance (Max 100lbs per)
  - Low power drive circuitry
- LTC660 Voltage Converter
  - Provides negative voltage for sensor
  - 100mA safely, 10mA minimal voltage drop



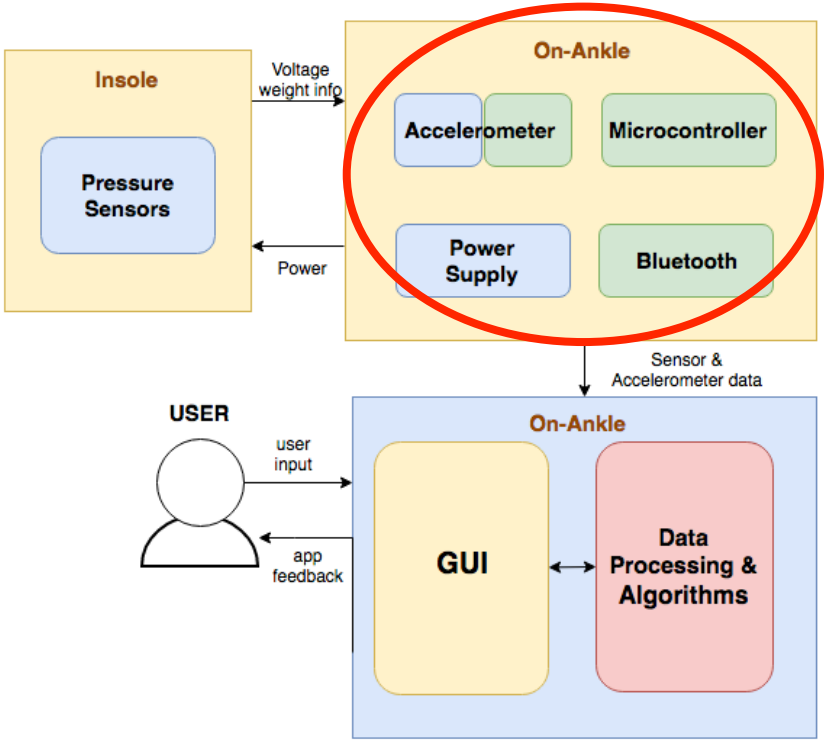
Sensor Demo

https://www.youtube.com/watch?v=LMUANdEJ1qY

Optimizing Sensor Placement



On-Ankle

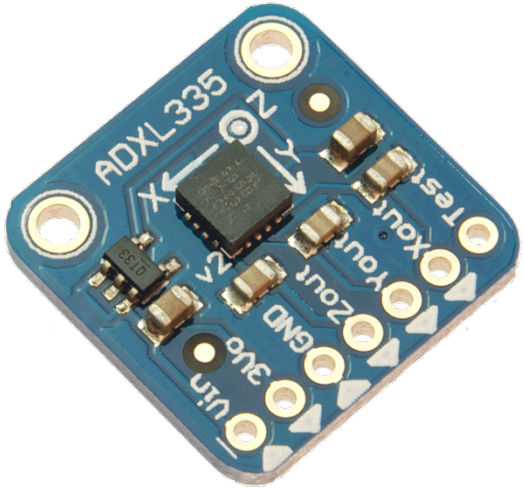

Insole
Pressure Sensors
Voltage weight info
Power
On-Ankle
Accelerometer
Microcontroller
Power Supply
Bluetooth
Sensor & Accelerometer data
USER
user input
app feedback
On-Ankle
GUI
Data Processing & Algorithms

Power

- USB Power Bank (2000mAh)
  - Portability
  - Capacity
  - Ease of implementation, reuse
- 5V, 3.3V, <20mA
  - External circuit power <163 mW
  - Arduino idle power 232.5 mW
  - Max power 315.5mW
  - Battery life ~5.05 hrs on one charge

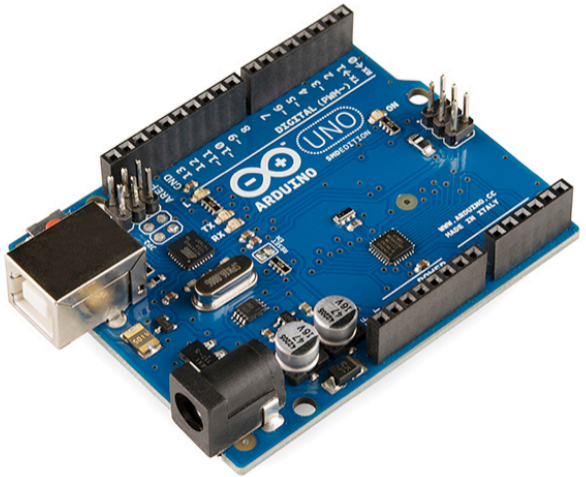
Accelerometers

- ADXL335
  - Low power (350 μA typical for 1.8-3.6V)
  - Ease of use



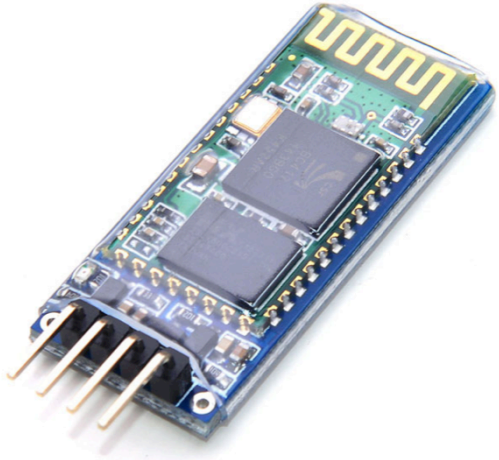
Microcontroller

- Control communication between sensors and mobile app via Bluetooth
  - Built in ADC for simple reading of analogue sensors
  - Simple to program for prototyping
  - Requires additional hardware for Bluetooth functionality



Bluetooth

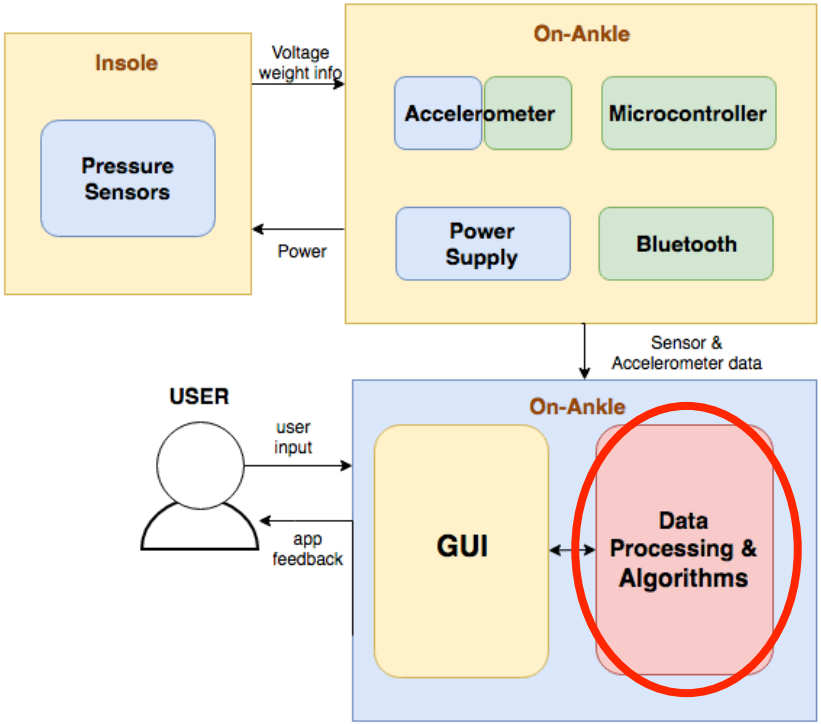
- JY-MCU Arduino Bluetooth Wireless Serial Port Module
  - Utilizes Arduino's built in serial communication library
  - 2.1 Mbps data rate
  - 30 ft range



Data Rate Calculations

- 4 x 10 bit force sensor readings
- 3 x 10 bit accelerometer reading
- 50 Hz sampling frequency
- 3.5kbps sampling total
- Bluetooth capable of sending 2.1Mbps

Data Processing and Algorithms



Algorithms and Data Processing

- Convert array of raw sensor data to useable format
- Filter beginning and end of data, so only steps are studied
- Calculate specific parameters given sensor data
- Compare parameters to healthy averages, and identify abnormalities

Algorithms and Data Processing

- Avg step Time ~ .5s
- Sample rate of 50Hz will give us about 25 samples per step
  - Well within Arduino frequency range
  - Depending on testing could choose higher frequency

Algorithms Demo

- Filtered beginning and end (to avoid using “non step samples)
- Heel Strike Times
- Toe off Times
- Step/Swing Times
- Cadence

Algorithms and Data Processing

Time	Force Sensor 0 (toe)	Force Sensor 1 (mid foot)	Force Sensor 2 (mid foot)	Force Sensor 3 (heel)	Accelerometer x	Accelerometer y	Accelerometer z
t0	fs00	fs10	.	.	.	.	Az0
t1	.						.
t2		.					.
.			.				.
.				.	.		.
.						.	.
tn	.	.	.	.	.	.	Azn

First Heel/ Last Toe

```
public static int getFirstHeel(int[][] sensorData){
      int prevHeel = 1;
      int firstHeel=0;
      for (int i = 0; i < sensorData.length ; i++){
            if ( prevHeel == 0 && sensorData[i][4] == 1){
                  firstHeel=i;
                  break;
            }
            prevHeel=sensorData[i][4];
                  }
      return firstHeel;
}

public static int getLastToe(int[][] sensorData){
      int lastToe = 0;
      int nextToe=1;
      for (int i = sensorData.length-1; i >= 0; i--){
            if (nextToe == 0 && sensorData[i][1]==1){
                  lastToe = i+1;
                  break;
            }
            nextToe=sensorData[i][1];
      }
      return lastToe;
}
```

- firstHeel - time where heel sensor first switches from no load to load – first heel strike
- lastToe - last time where toe sensor switches from load to no load – last toe off
- We know values between these are steps – all data between is good data

Heel Time/ Toe Times

```
public static double[] getHeelTimes(double[][] sensorData, int stepCount, int firstHeel, int lastToe) {
        double prevHeel = 0;
        int j = 0;
        double[] heelTimes = new double[stepCount];
        for (int i = firstHeel; i < lastToe; i++) {
                if (sensorData[i][4] >= 1 && prevHeel < 1) {
                        heelTimes[j] = sensorData[i][0];
                        j++;
                }
                prevHeel = sensorData[i][4];
        }
        return heelTimes;
}
public static double[] getToeTimes(double[][] sensorData, int stepCount, int firstHeel, int lastToe) {
        double prevToe = 0;
        int j = 0;
        double[] toeTimes = new double[stepCount];
        for (int i = firstHeel; i <= lastToe; i++) {
                if (sensorData[i][1] <= 1 && prevToe > 1) {
                        if (j < stepCount) {
                                toeTimes[j] = sensorData[i][0];
                                j++;
                        }
                }
                prevToe = sensorData[i][1];
        }
        return toeTimes;
}
```

- Finds all heel strike times, between first heel and last toe
  - Creates Array of heel strike times
- Finds all toe off times, between first heel and last toe
  - Creates Array of toe off times

Step Time / Swing Time

```
public static int[] getStepTimes(int[] heelTimes, int[] toeTimes, int stepCount) {
     int[] stepTimes = new int[stepCount];
     for (int i = 0 ; i < stepCount; i++){
          if (heelTimes[0] < toeTimes[0])
               stepTimes[i] = toeTimes[i] - heelTimes[i];
          else
               stepTimes[i] = toeTimes[i + 1] - heelTimes[i];
     }
     return stepTimes;
}

public static int[] getSwingTimes(int[] heelTimes, int[] toeTimes, int stepCount) {
     int[] swingTimes = new int[stepCount-1];
     for (int i = 0; i < stepCount-1; i++) {
          if (heelTimes[0] > toeTimes[0])
               swingTimes[i] = heelTimes[i] - toeTimes[i];
          else
               swingTimes[i] = heelTimes[i + 1] - toeTimes[i];
     }
     return swingTimes;

}
```

- Step time = Toe off time - Heel Strike Time
  - Creates array of all step times
- Swing time = Heel strike time – Toe off time
  - Creates array of off swing times
- Averages are calculated in separate method

Compare Step and Swing Time

```
public static void compareSwingandStep(double[] swingTimes, double[] stepTimes){
        for (int i = 0 ; i < swingTimes.length && i < stepTimes.length ; i ++){
                if (swingTimes[i] < .8*stepTimes[i])
                        System.out.println("YOUR SWING TIME IS SIGNIFICANTLY SHORTER THAN YOUR STEP TIME");
                if (stepTimes[i] < .8*swingTimes[i])
                        System.out.println("YOUR STEP TIME IS SIGNIFICANTLY SHORTER THAN YOUR SWING TIME");
        }
}
```

- Compares step times to swing times, and prints message if there is an abnormality

Cadence

```
public static int getCadence(double avgStepTime){
        return (int) (60/avgStepTime);
}
public static void compareCadence(double cadence){
        if (cadence < 100)
                System.out.println("User walks with slow cadence.");
        if (cadence>120)
                System.out.println("User walks with fast cadence.");
}
```

- Cadence = steps per minute
- Average walking cadence is between 100 and 120 steps per minute
  - Prints message if user is outside normal cadence

Test 1

```
{{0.00, 10, 10, 10, 10},{0.02, 20, 10, 10, 0},{0.04, 30, 5, 5, 0},{0.06, 40, 0, 0, 0},
{0.08, 0, 0, 0, 0},{0.10, 0, 0, 0, 0},{0.12, 0, 0, 0, 0},{0.14, 0, 0, 0, 0},{0.16, 0, 0, 0, 0},{0.18, 0, 0, 0, 40},
{0.20, 0, 0, 0, 40},{0.22, 0, 0, 0, 40},{0.24, 0, 5, 5, 30},{0.26, 0, 10, 10, 20},{0.28, 5, 10, 10, 15},{0.30, 10, 10, 10, 10},
{0.32, 20, 10, 10, 0},{0.34, 30, 5, 5, 0},{0.36, 40, 0, 0, 0},{0.38, 10, 0, 0, 0},{0.40, 10, 0, 0, 0},{0.42, 10, 0, 0, 0},
{0.44, 10, 0, 0, 0},{0.46, 10, 0, 0, 0},{0.48, 10, 0, 0, 0},{0.50, 10, 0, 0, 0},{0.52, 10, 0, 0, 0},{0.54, 10, 5, 5, 0},{0.56, 10, 10, 10, 0},
{0.58, 5, 10, 10, 0},{0.60, 10, 10, 10, 0},{0.62, 20, 10, 10, 0},{0.64, 30, 5, 5, 0},{0.66, 40, 0, 0, 0},{0.68, 40, 0, 0, 0},
{0.70, 0, 0, 0, 0},{0.72, 0, 0, 0, 0},{0.74, 0, 0, 0, 0},{0.76, 0, 0, 0, 0},{0.78, 0, 0, 0, 0},{0.80, 0, 0, 0, 0 },{0.82, 0, 0, 0, 0},
{0.84, 0, 0, 0, 0},{0.86, 0, 0, 0, 0},{0.88, 0, 0, 0, 0},{0.90, 0, 0, 0, 0},{0.92, 0, 0, 0, 0},{0.94, 0, 0, 0, 0},{0.96, 0, 0, 0, 0},
{0.98, 0, 0, 0, 0},{1.0, 0, 0, 0, 0},{1.02, 0, 0, 0, 0},{1.04, 0, 0, 0, 0},{1.06, 0, 0, 0, 0},{1.08, 0, 0, 0, 0},{1.10, 0, 0, 0, 0},
{1.12, 10, 10, 10, 10},{1.14, 20, 10, 10, 10},{1.16, 30, 5, 5, 10},{1.18, 40, 0, 0, 10},
{1.20, 10, 0, 0, 10},{1.22, 10, 0, 0, 10},{1.24, 10, 0, 0, 10},{1.26, 10, 0, 0, 10},{1.28, 10, 0, 0, 10},{1.30, 10, 0, 0, 40},
{1.32, 10, 0, 0, 0},{1.34, 10, 0, 0, 0},{1.36, 10, 5, 5, 0},{1.38, 10, 10, 10, 0},{1.40, 5, 10, 10,0 },{1.42, 10, 10, 10, 0},
{1.44, 20, 10, 10, 0},{1.46, 30, 5, 5, 0},{1.48, 40, 0, 0, 0},{1.50, 10, 0, 0, 0},{1.52, 10, 0, 0, 0},{1.54, 10, 0, 0, 0},
{1.56, 10, 0, 0, 0},{1.58, 10, 0, 0, 0},{1.60, 10, 0, 0, 0},{1.62, 10, 0, 0, 0},{1.64, 10, 0, 0, 0},{1.66, 0, 0, 0, 0},
{1.68 ,0 ,0 ,0 ,0},{1.70 ,0 ,0 ,0 ,0},{1.72, 0, 0, 0, 0}};
```

Cadence- 113 steps per minute

First Heel @ .18 s

Last Toe @ 1.66 s

heel times[0.18, 1.12]

toe times[0.7, 1.66]

step times[0.52, 0.5399999999999998]

swing times[0.42000000000000015]

avg swing time 0.42

avg step time 0.53

Test 2

```
{{0,1,1,1,1},{.02,0,1,1,1},{.04,0,0,0,1},{.06,0,0,0,0},{.08,1,0,0,0},{.1,1,1,1,0},{.12,1,1,1,1},{.14,0,1,1,1},{.16,0,0,0,1},
{.18,0,0,0,0},{.2,1,0,0,0},{.22,1,1,1,0},{.24,1,1,1,1},{.26,0,1,1,1},{.28,0,0,0,1},{.30,0,0,0,0},{.32,1,0,0,0},{.34,1,1,1,0},
{.36,1,1,1,1},{.38,0,1,1,1},{.40,0,0,0,1},{.42,0,0,0,0},{.44,1,0,0,0},{.46,1,1,1,0},{.48,1,1,1,1},{.50,0,1,1,1}};
```

Cadence – 2999 steps per minute

First heel @ .12 s

Last toe @ .5s

heel times[0.12, 0.24, 0.36, 0.48]

toe times[0.14, 0.26, 0.38, 0.5]

step times[0.020000000000000018, 0.020000000000000018, 0.020000000000000018, 0.020000000000000018]

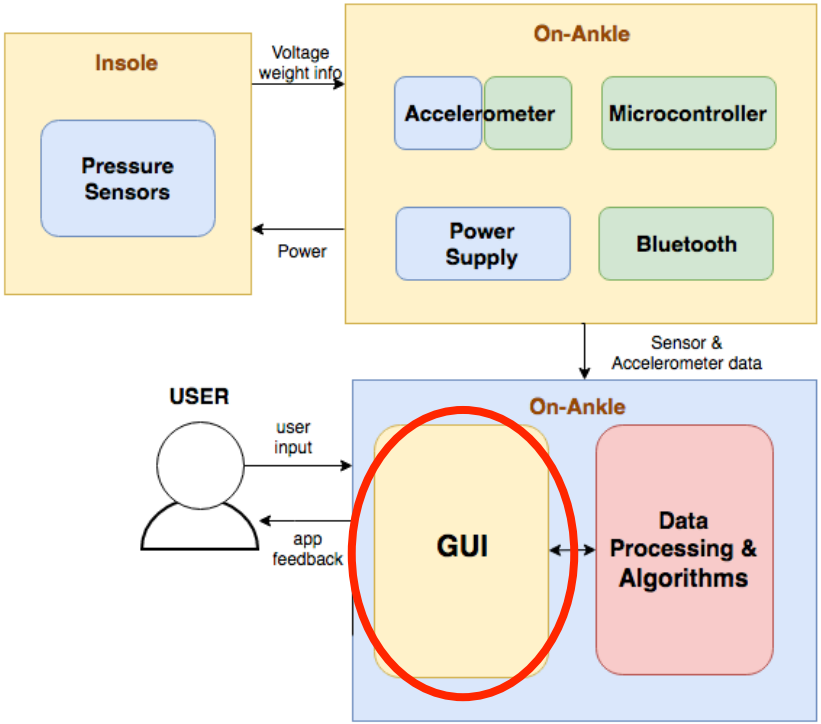
swing times[0.09999999999999998, 0.09999999999999998, 0.09999999999999998]

avg swing time 0.10

avg step time 0.02

Should throw errors for fast cadence, as well as mismatched step and swing time

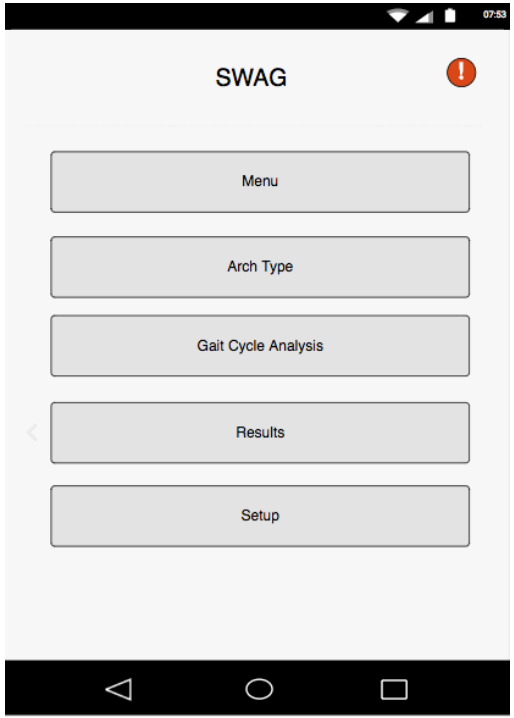
GUI

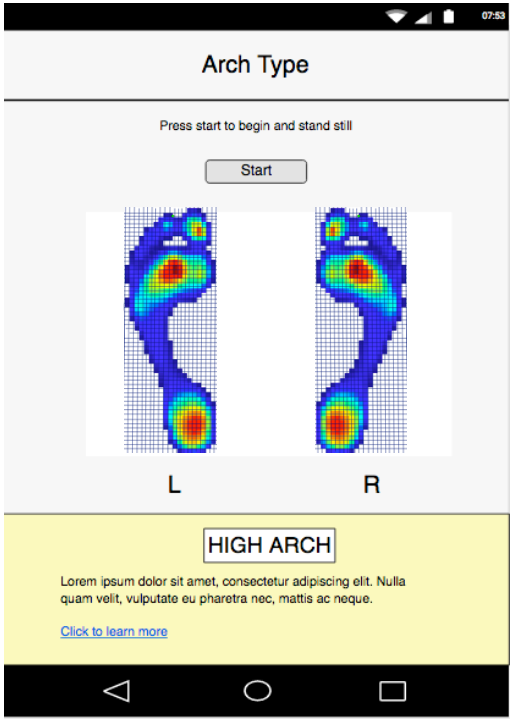

Insole
Pressure Sensors
Voltage weight info
Power
On-Ankle
Accelerometer
Microcontroller
Power Supply
Bluetooth
Sensor & Accelerometer data
USER
user input
app feedback
On-Ankle
GUI
Data Processing & Algorithms

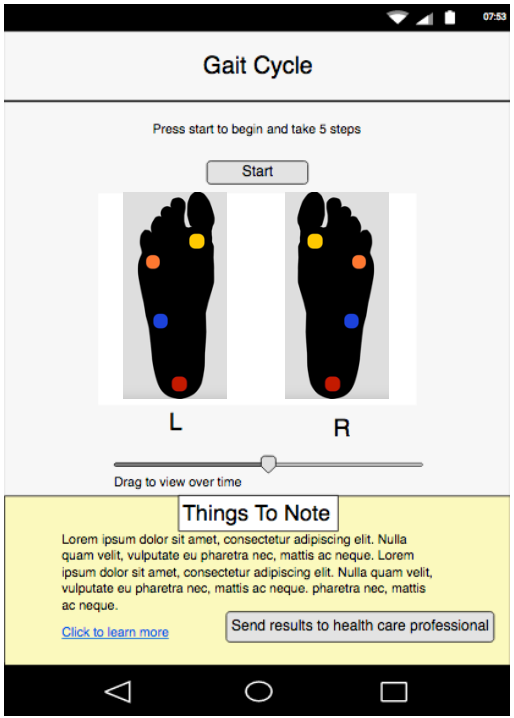
Inputs for Mobile Application

- Sensor 1,2...n -> force (N) at given time (sec)
- Average force (N) applied at given time (sec)
- Time between step (sec)
- Distance foot swings outside starting stance (cm)
- Height above ground during step (cm)
- Height above ground (cm)
- Stride length per step (meter)

UI Mockup

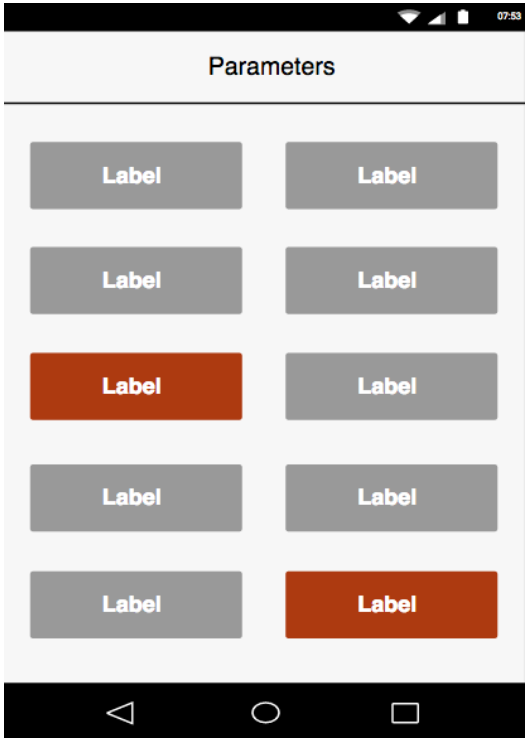

07:53
SWAG
Menu
Arch Type
Gait Cycle Analysis
Results
Setup

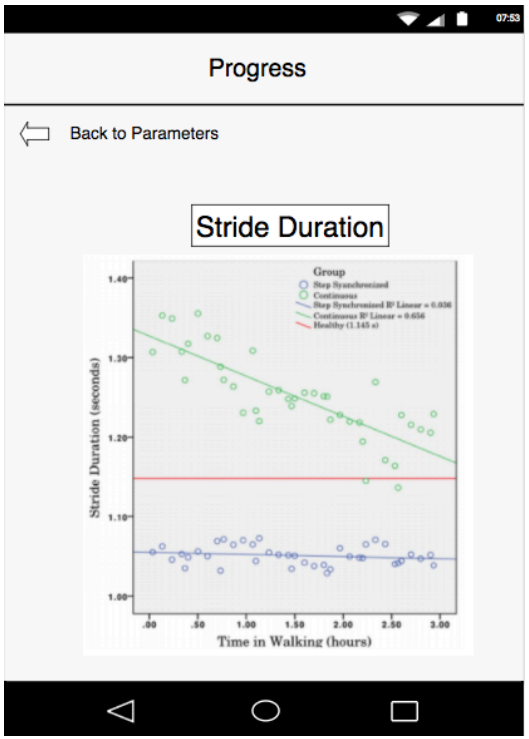

07:53
Arch Type
Press start to begin and stand still
Start
L
R
HIGH ARCH
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque.
Click to learn more


07:53
Gait Cycle
Press start to begin and take 5 steps
Start
L
R
Drag to view over time
Things To Note
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. pharetra nec, mattis ac neque.
Click to learn more
Send results to health care professional

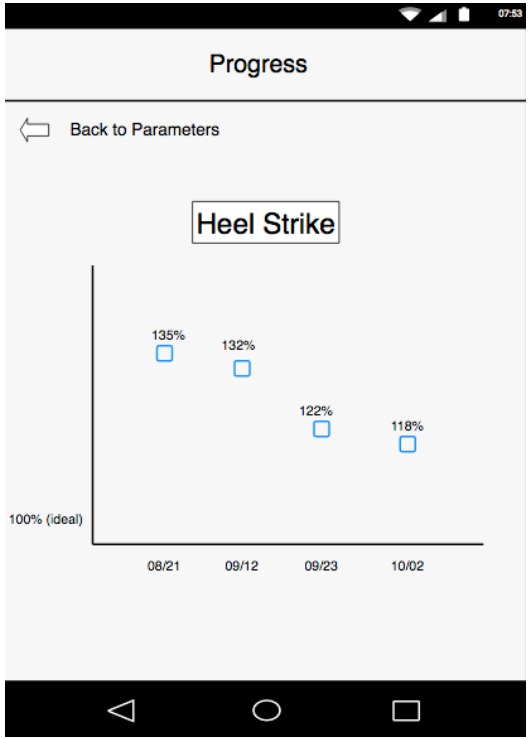
UI Mockup


Setup
Bluetooth Connected?
Left Foot
Right Foot


Parameters
Label
Label
Label
Label
Label
Label
Label
Label
Label
Label

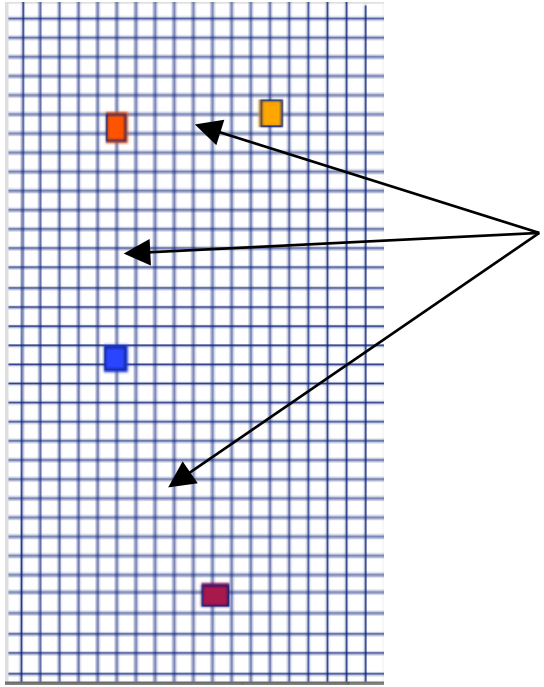

Progress
Back to Parameters
Stride Duration
Group
Step Synchronized
Continuous
Healthy (1.145 s)
Stride Duration (seconds)
1.40
1.30
1.20
1.10
1.00
.00
.50
1.00
1.50
2.00
2.50
3.00
Time in Walking (hours)

UI Mockup


07:53
Progress
Back to Parameters
Heel Strike
135%
132%
122%
118%
100% (ideal)
08/21
09/12
09/23
10/02

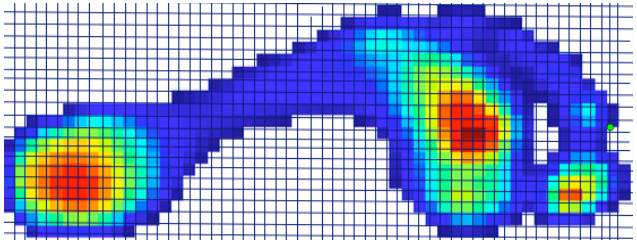

07:53
Patients
John Doe
contact@example.com
John Doe
contact@example.com
John Doe
contact@example.com
John Doe
contact@example.com
John Doe
contact@example.com
John Doe
contact@example.com

Heat Map Generation



- 20x60 Two Dimensional Array
- Points in between sensors approximated by values taken at sensors and center of gravity
- Populate 2d array with values correlating to the data read at sensor locations

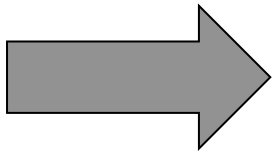
		1.09x	1.02x
	1.18x	1.14x	1.15x
1.30x	1.20x	x	1.19x
1.34x	1.29x	1.28x	1.24x
1.41x	1.39x	1.37x	1.30x

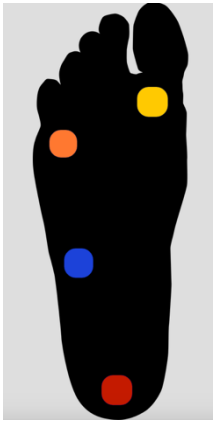


Gait Cycle

Input: Force for each sensor at time t
Output: Color representation of force distribution over time

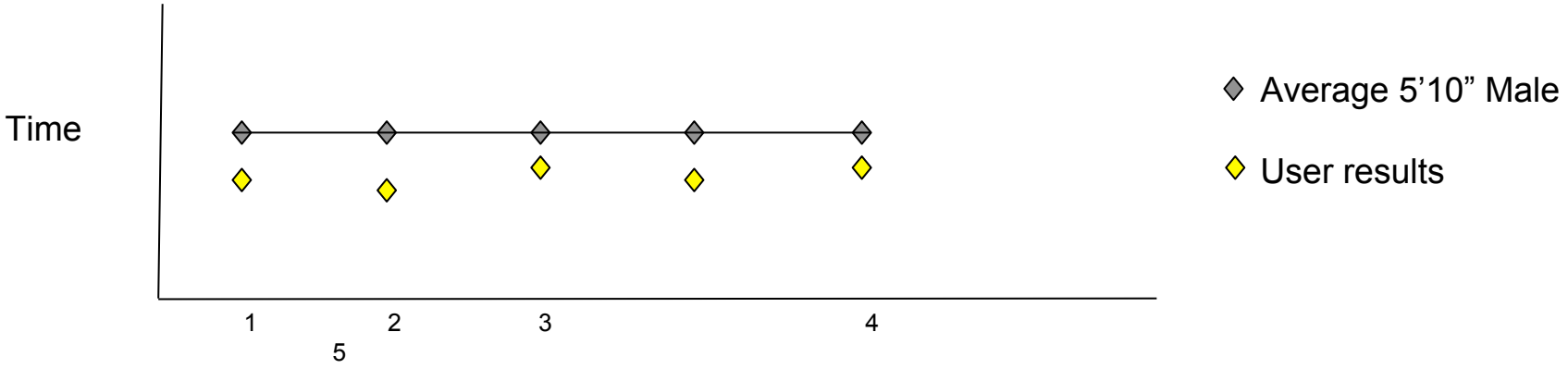
T	FS1	FS2	FS3	FS4
t1	fs1	.	.	.
t2	fs2	.		
t3	fs3		.	
.	.			.
.	.			
.	.			
tn	fsn	.	.	.





Tracking Progress on App

- Parameters will be visually represented by graphs
- Show average vs calculated
- Example (time between steps):



CDR Deliverables

- Integrate accelerometers
- Determine optimal location on foot to place sensors
- Establish bluetooth communication between mobile device and microcontroller
- Complete data processing algorithms
- Completed Android GUI
- End to end integration of final system

Timeline

	December/January	February	March	April
Alex	Center of Gravity and Weight Distribution Algorithms	Accelerometer Algorithms	Completed algorithms	
Jack	Accelerometer Integration	Assemble insole	Finalize Insole Design	
Scott	Finish GUI/ Bluetooth Communication	Visual Analytics Code	Finalize App Design	
Eric	Accelerometer Integration / Bluetooth Communication	PCB Design	Finalize Anklet Design	

Thank You!

Questions?