UNIVERSITY OF CALIFORNIA

Los Angeles

mFit: A Novel Sensor Fusion Platform for the Promotion of Exercise

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Science

in Electrical Engineering

By

Mahdi Ashktorab

2012

ABSTRACT OF THE THESIS

mFit: A Novel Sensor Fusion Platform for the Promotion of Exercise

by

Mahdi Ashktorab

Master of Science in Electrical Engineering

University of California, Los Angeles, 2012

Professor William J. Kaiser, Chair

Proper fulfillment of prescribed exercise has shown to greatly reduce the rate of patient re-admission and speed recovery of patients experiencing an array of health conditions. Currently, technology that allows doctors to monitor the fulfillment of exercise regiments requires patients to submit to expensive physical therapy and long term monitoring. An urgent need exists for low cost, rapidly deployable, and networked systems that enable monitoring of exercise activity for patients both in the clinic and at home.

The mFit exercise system utilizes modern sensor fusion techniques in conjunction with open source smartphone technologies to monitor: cadence, applied force, generated power, and dissipated energy while providing guidance and real time data to the user. The system also transmits patient data to a central server, which

allows service providers to oversee patients, ensuring adherence to prescribed exercise regiments, and provide feedback as needed.

To accommodate patients in different stages of recovery, two variants of the mFit exercise system have been developed. *The mFit restorator* outfits a standard restorator bike with polymer force and Hall Effect sensors, enabling direct measurement of a user's force and cadence. Data is transmitted via Bluetooth to a smartphone that displays the information in an intuitive graphical user interface. The smartphone further utilizes WiFi or a mobile telecommunication network to transmit data to a central server which can be accessed by the user or a third party.

The mFit stationary cycle has been developed for patients that are prescribed more strenuous exercise routines. An inexpensive, mass market stationary cycle was modified and a strain gauge, common in most weight scales, and a photo-interrupter were added to enable force and cadence measurements. Similar to the mFit restorator, the data is transmitted to a smartphone that displays the results and simultaneously pushes the data to a central server.

In this thesis, we present the novel mFit system concept, the hardware used to develop the mFit restorator and mFit stationary cycle, as well as the software architecture which presents the required metrics. We further describe applications which interface with the mFit systems to provide guidance and measure a user's fitness level.

The thesis of Mahdi Ashktorab is approved.

Robert Candler

Majid Sarrafzadeh

William J. Kaiser, Committee Chair

University of California, Los Angeles

2012

Table of Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to express my sincerest gratitude to my advisor, Professor William J. Kaiser; for his timeless patient, tireless guidance, and priceless mentorship.

I would like to thank the members of my thesis committee, Professor Majid Sarrafzadeh and Professor Robert Candler for their invaluable help and feedback with my thesis.

I would like to thank Yeung Lam for his support, encouragement, and friendship. I would like to further thank my fellow colleagues and dear friends at the Wireless Health Institute and ASCENT, we truly make a unique family.

Finally, I would like to dedicate this thesis to my parents, my brother, and my sister for giving me the confidence to reach for the stars.

Chapter 1

Introduction

1.1 Overview

Exercise has long been associated with improved cardiovascular health and longevity [1]. Recent clinical data has shown exercise to be of great benefit for patients recovering from stroke or respiratory disorder and geriatric subjects suffering from a diverse range of conditions [2-4]. Exercise has also been linked to decreased patient readmission and reduction in duration of hospital stay through proper rehabilitation exercises [4, 5].

Currently, for physicians to ensure the fulfillment of an exercise prescription, the patient is required to regularly visit specialized facilities and submit to monitoring by clinicians. This is both costly and inconvenient. In order to enable patients to monitor themselves from home, a system needs to be developed to measure the user's energy expenditure and send the data to a remote server accessible to the healthcare provider.

Conventional exercise systems, such as a stationary cycle, either do not include measurement of a user's energy, or are expensive, costing greater than $3000 per unit.

Even with the more expensive models, lack of network connectivity means there is no reliable way for the physician to receive the patients exercise data. An urgent need exists for a low cost, rapidly deployable, and networked exercise platform that can provide physicians the ability to monitor their patients without the need for regular and expensive clinical visits.

1.2 Objectives and Contribution

The objective of this thesis is to describe the development of a novel system that allows patients to exercise from the comfort of their home while the data is send remotely to their care provider. This system would allow care providers to monitor the patient's progress and provide guidance to speed the patient's recovery.

Herein, the mFit system is described as a platform for any resistive based exercise machine. The mFit restorator and cycle are two examples of the mFit systems implementation. Prior publication on the mFit restorator is available [6, 7]. This thesis provides a brief introduction to the mFit restorator, but focuses on the mFit cycle as the main research contribution.

Contribution to this thesis includes:

1) Modification of a standard exercise cycle with a sensor network capable of measuring the energy expenditure of the user
    - **a.** Cost effective system
    - **b.** Easily sourced and mass producible parts

c. Measure: cadence, input power, and energy expenditure.

2) Calibration procedure of the cycle to ensure reliable data

3) Implementation of a mobile interface to receive streaming exercise data

a. Intuitive interface for the user

b. Show user metrics in real time

c. Transmit data to a central server

Chapter 2

System Design

In this chapter the mFit system's implementation into a consumer restorator and stationary cycle are described. The hardware used in both devices and the method for calculating cadence are shown in detail.

2.1 System Overview

The mFit system is a low-cost sensor package consisting of over-the-shelf components that enable direct measurement of a user's cadence and input force. Fitness metrics, such as power, energy, and distance traveled, can be derived through the use of cadence and torque. The combined metrics from the mFit system are wirelessly transmitted to a local mobile device. The mobile device captures the data in real time and displays the information in a graphical user interface, providing a simple interface for presentation and manipulation of the user's data. The metrics can be streamed via the mobile device to a remote host, allowing for third parties to monitor the user and provide guidance and feedback.

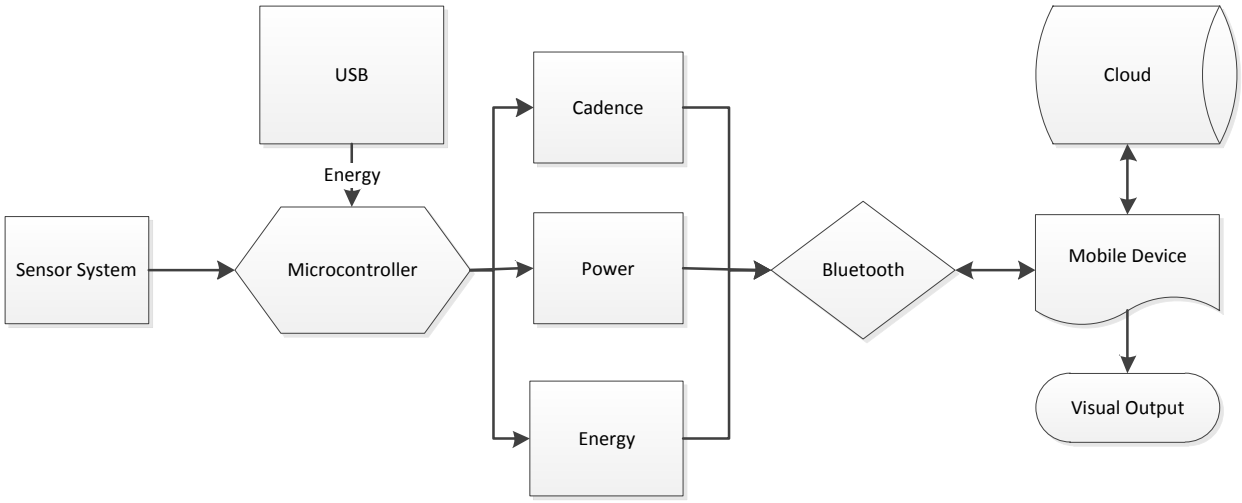


Figure 2.1: Block diagram of mFit system

2.2 mFit Hardware

The hardware for each of the mFit systems are described in detail.

2.2.1 mFit Restorator

Figure 2.2: mFit restorator

The mFit system implementation in the restorator (Figure 2.2) is designed to be low cost, rapidly deployable, intuitive to use, discrete, and structurally robust. To enable torque measurement, a tab was welded underneath the floating resistance control hub. A polymer force sensor [8] was placed atop the tab, bending based on the force placed on the resisting brakes. Twisting the resistance knob increases the clamping force on the two plastic friction pads that sandwich the center metal bar of the one-piece crank. (Figure 2.4) The increase in friction requires a larger force pressed onto the crank. This in turn causes a larger bend on the metal tab which is measured by the force sensor.

The data is collected by the custom built MicroLEAP compact sensing platform.

Specifications for the MicroLEAP are listed below:

- Processor: MSP430F1611
- Max Clock Frequency: 8MHz
- Voltage: 3.7V-5.5V
- GPIO: 48
- ADC: 8-channels, 16-bits
- Interfaces: I2C, USART
- Flash Memory: 8Mbit
- Radio Communication: Bluetooth 2.0
- Active Current Consumption (8Mhz): 26 mA

Figure 2.3: MicroLEAP

The MicroLEAP (Figure 2.3) [9] was chosen for four reasons. The MicroLEAP: 1) Provides the high resolution analog interface required to capture minute changes in the polymer force sensor. 2) Enables wireless communication to a mobile platform via Serial Port Profile (SPP) protocol using the built in Bluetooth chip. Bluetooth also allows for remote control via an ASCII-based protocol, enabling dynamic control of sampling frequency, charge state, and sensor channel selection. 3) Contains a processor powerful enough to perform the metric calculations yet maintains low power usage, enabling continuous wireless data transfer for over 24 hours on a Li-Polymer battery. 4) Is designed to allow for seamless integration into the restorator chassis.

Sensor data is acquired via any Bluetooth-enabled platform. The current mFit system utilizes an Android application for data acquisition and display of the data via a touchscreen GUI. The Android application in turn, transmits the data to a central server via WiFi or the mobile telecommunication network.

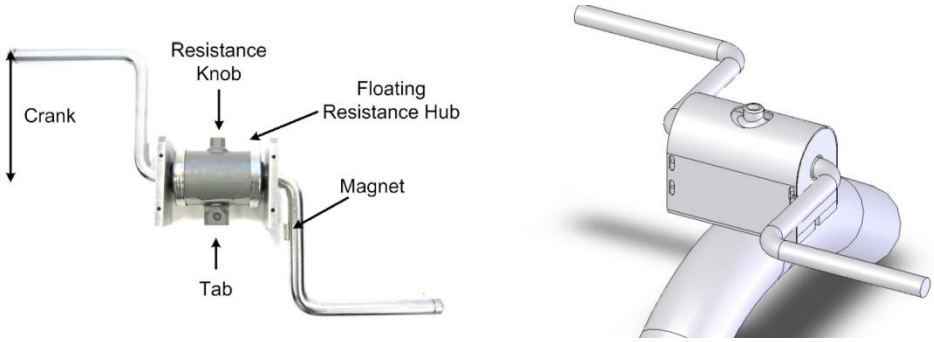


Figure 2.4: mFit restorator's chassis showing the cadence and torque sensors. [6]

2.2.2 mFit Stationary Cycle

Figure 2.5: mFit stationary cycle

The cycle chosen to be outfitted with the mFit system was the ProForm 290 SPX (Figure 2.5) [10] Indoor Cycle Trainer. The cycle was selected for three reasons: 1) Inexpensive, carrying only a $299 price tag. 2) It utilized a brake pad for increasing resistance to the flywheel. 3) The design of the cycle allowed for easy modifications to the frame and brakes.

The cycle's brake pads set the level of resistance by varying the frictional force required to turn the cycle's flywheel. The required pedaling force can be controlled by adjusting the resistance knob, which increases the force of friction on the two brake pads that sandwich the flywheel. The frictional force results in a strain on the mounting bracket attaching the brake calipers to the frame. By modifying the mounting bracket to include a strain sensor, a direct measurement of the frictional force on the wheels can be obtained. (Figure 2.6)

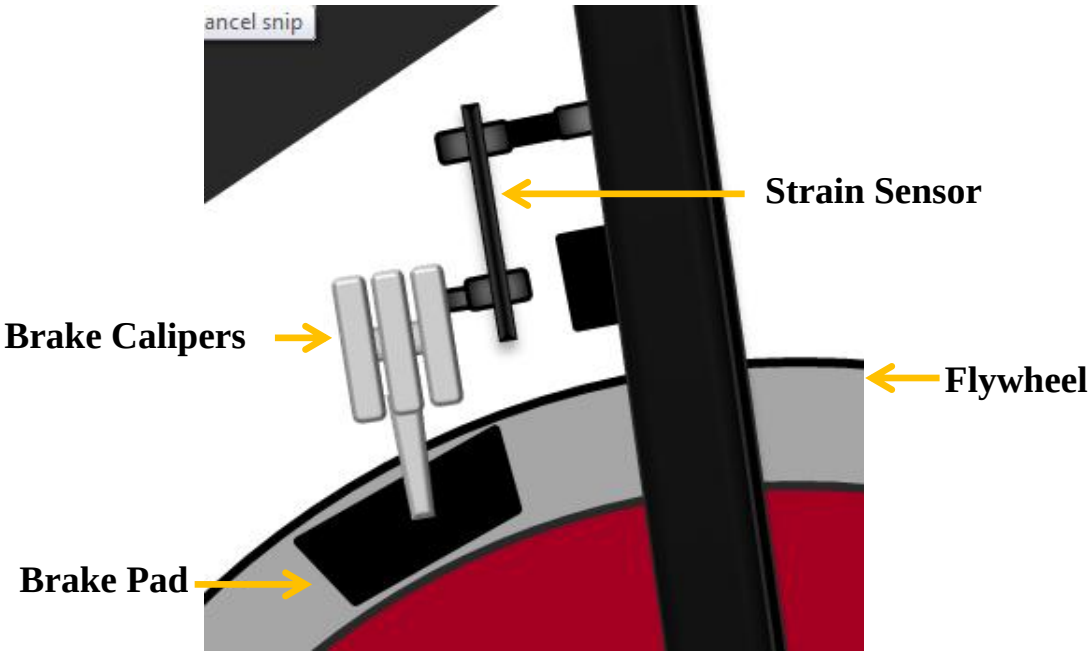


Figure 2.6: mFit cycle speed and torque sensing system components.

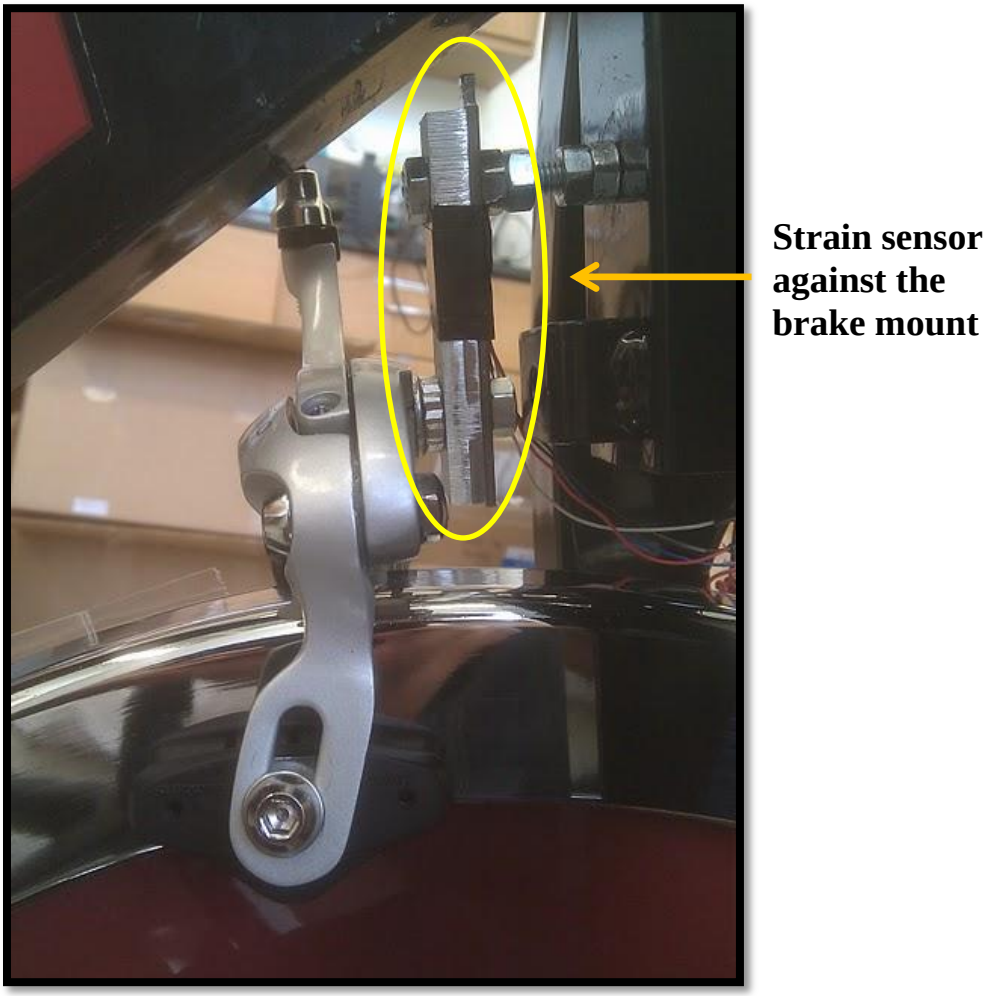


Figure 2.7: Strain sensor positioned against the brake mount.

The strain sensor serves to measure the force exerted on the brake calipers of the mFit cycle. Strain gauges are known for their accuracy and minimal drift over time, an ideal sensor for this application.

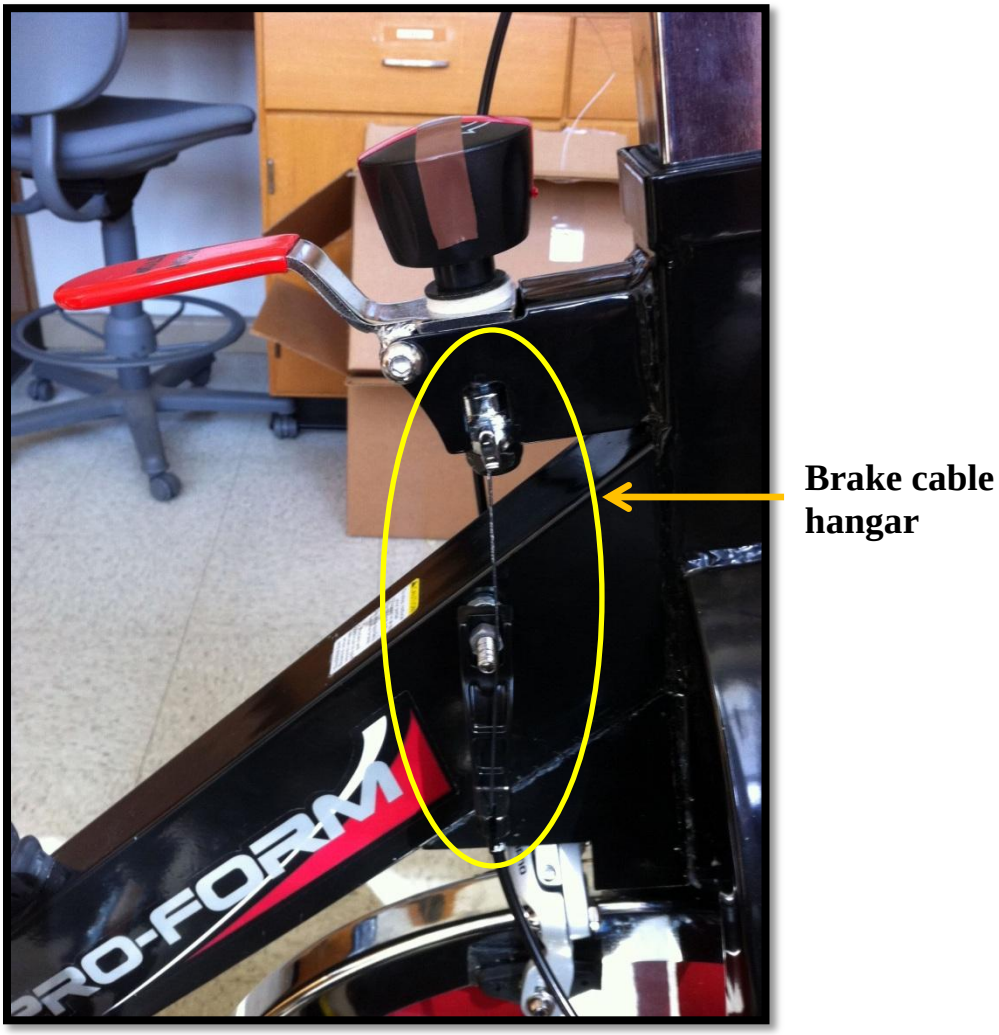


Figure 2.8: Side pull brake cable mounting system

In volume, strain sensors are inexpensive and are used in many common digital scales. A strain sensor was obtained by reverse engineering a $10 Taylor Digital Scale, Model 7329. The strain sensor can be seen mounted on top of a metal bracket. The bracket serves to restrict the strain sensors range of motion, protecting it from being over flexed. (Figure 2.7)

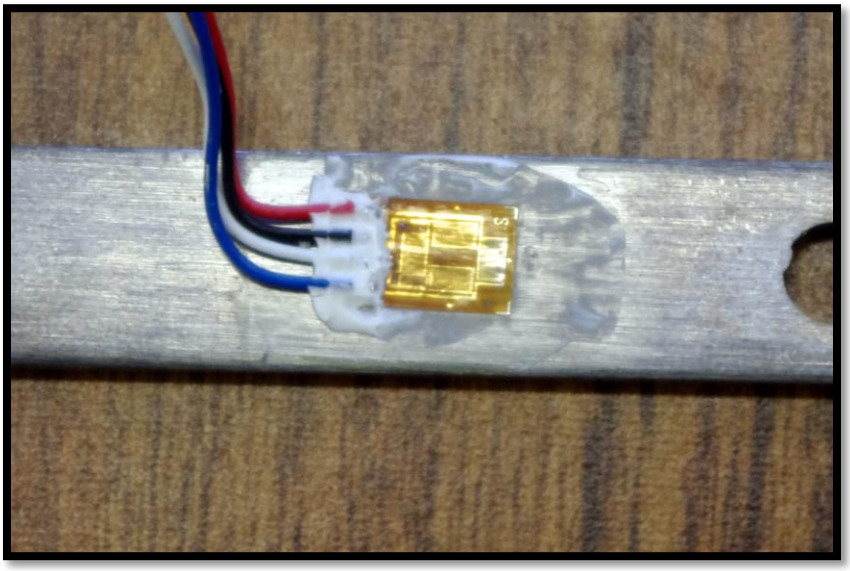


Figure 2.9: Strain sensor mounted on top of metal bracket.

The four distinct strain gauges seen in figure 2.9 categorizes this sensor as a full-bridge strain sensor circuit. The four gauges are placed in a wheat-stone bridge to provide maximum sensitivity and linearity. (Figure 2.10)

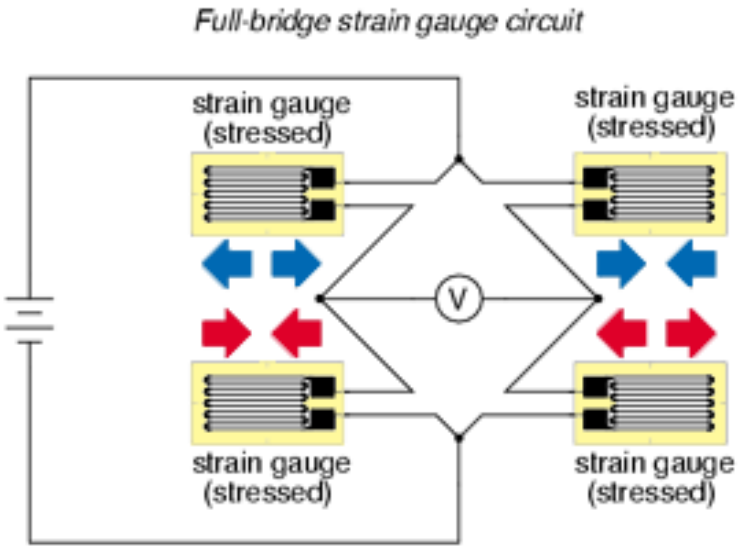


Figure 2.10: Full-bridge strain gauge circuit. [11]

Stressing the strain gauge causes a minute change in the total resistance of the circuit compared to the unstressed state. Appropriately placing a voltage source and a voltmeter across the Wheatstone bridge likewise shows a change of only a few mV, requiring large amplification to attain the required sensitivity. The output of the amplifier was connected to the ADC on the sensor platform.

By relating the frictional force of the brakes on the flywheel to the input force the user places on the pedals, an algorithm was developed that calculates the user's input power and energy expenditure using the strain sensor values.

Modification on the standard ProForm brakes were required as the initial center-pull braking system used strait wires to place an upward force on the caliper arms. This upward force also placed a force on the strain gauge equipped brake mount, changing the profile of the gauge's measurements. The solution was to replace the brake system with a single pivot side-pull caliper brake and use Bowden cables, isolating the pulling force from the brake lever to the calipers and removing the strain on the brake mounts. (Figure 2.8)

Similar to the mFit restorator, the mFit cycle is equipped with a microcontroller and a Bluetooth platform to wirelessly transmit the strain sensor data. The sensing platform chosen for the mFit cycle was the Arduino Pro Mini (Figure 2.11) [12] made by Sparkfun. Specifications for the Arduino Pro Mini are listed below:

- Processor: ATmega328

- Max Clock Frequency: 8MHz
- Voltage: 3.3V-12V
- GPIO: 22
- ADC: 8-channels, 10-bits
- Interfaces: I2C, UART
- Flash Memory: 32KBytes
- Active Current Consumption (8Mhz): 11 mA

The Pro Mini was chosen for several reasons: 1) Simplified coding and large support community for being part of the Arduino development environment. 2) Serial interface for wireless communication. 3) Low power processor with sufficient power for real time metric calculation. 4) Inexpensive and easily sourced.

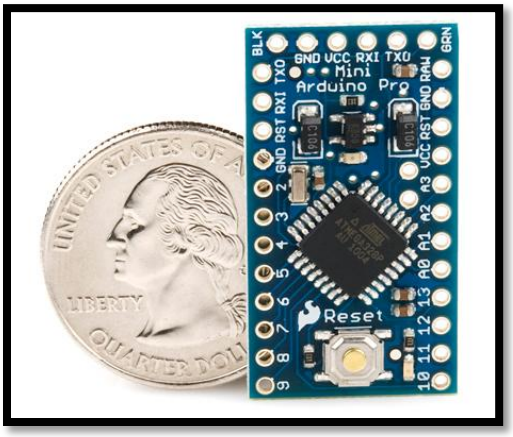


Figure 2.11: Arduino Pro Mini.

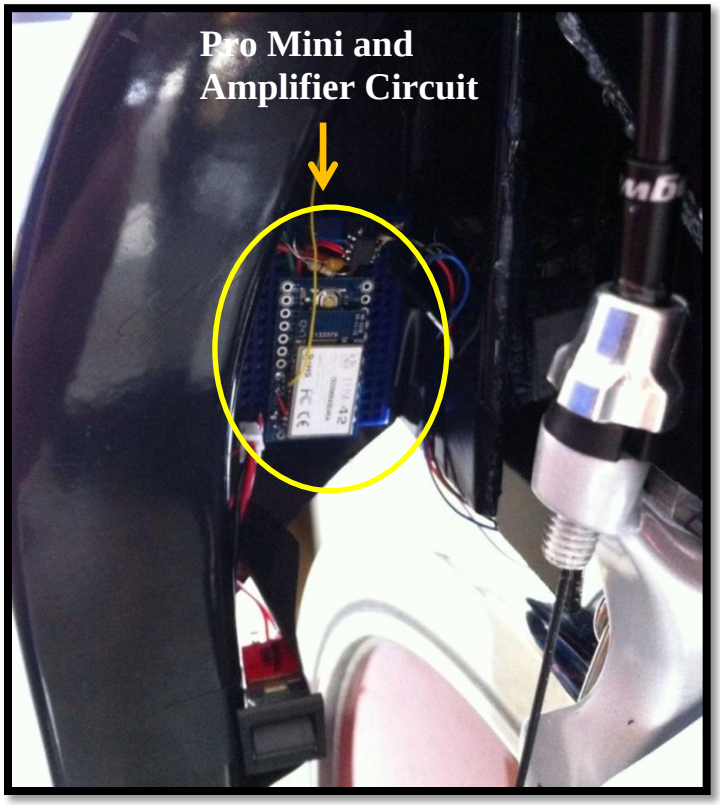


Figure 2.12: mFit system hardware mounted against the stationary cycle.

The mFit cycle uses the same SPP Bluetooth protocol as the mFit restorator and can thus easily integrate with the same Android application.

2.3 mFit Cadence Calculation

Cadence, or the number of rotation per minute (RPM), can be measured by introducing a fixed sampling location along the trajectory of the rotating crank or wheel. Every time the sampling location is passed, the mFit registers a trigger.

Cadence can be computed by measuring the time difference between two successive triggers (Δt). Cadence is a required metric to calculate power and energy.

2.3.1 mFit Restorator Cadence Calculation

For the mFit restorator, cadence is measured via a Hall Effect sensor [13] placed on the restorator's outer metal cover, with the accompanying magnet on the inner crank arm. As the crank arm with the attached magnet passes the hall-effect sensor, the corresponding magnetic field is detected and the information transmitted to the microLEAP, registering a trigger once a threshold is surpassed.

2.3.2 mFit Cycle Cadence Calculation

Two methods were developed for the mFit cycle to determine the cadence required to calculate power and energy.

2.3.2.1 Disturbance Detection

The first method uses the unique signature of a bump on the flywheel passing below the brake pad. The prototype cycle uses a piece of tape (Figure 2.13) placed on the outer radius of the flywheel to provide the necessary disturbance. The algorithm detecting the tape signature consists of two metrics. The first being standard deviation and the second, the percentage offset. Each metric, in turn, provides two thresholds: one threshold notes the beginning of the disturbance; the other marks the end of the disturbance.

When the correct combination is detected a timestamp is taken to mark the tapes signature. Each timestamp is subtracted from the previous value to determine the length of time for one rotation. Three successive rotations are continuously averaged and the RPM calculated. This minimizes the impact of anomalies and noise at the cost of accuracy.

Figure 2.13: Tape placed on the side of the mFit cycle's front wheel.

2.3.2.2 Photointerrupter

The second method utilizes a photointerrupter [14] placed behind the pedal as shown in figure 2.14. A flag was mounted atop the gear connected to the pedal (Figure 2.14)

and a photointerrupter was attached to the housing around the gear and positioned such the flag crossed between the LED transmitter and receiver pillars. Each crossing interrupts the LED beam between the pillars, in turn resulting in an interrupt call to the Arduino system. The interrupt immediately saves a timestamp and calculates the RPM using the data from previous timestamps.

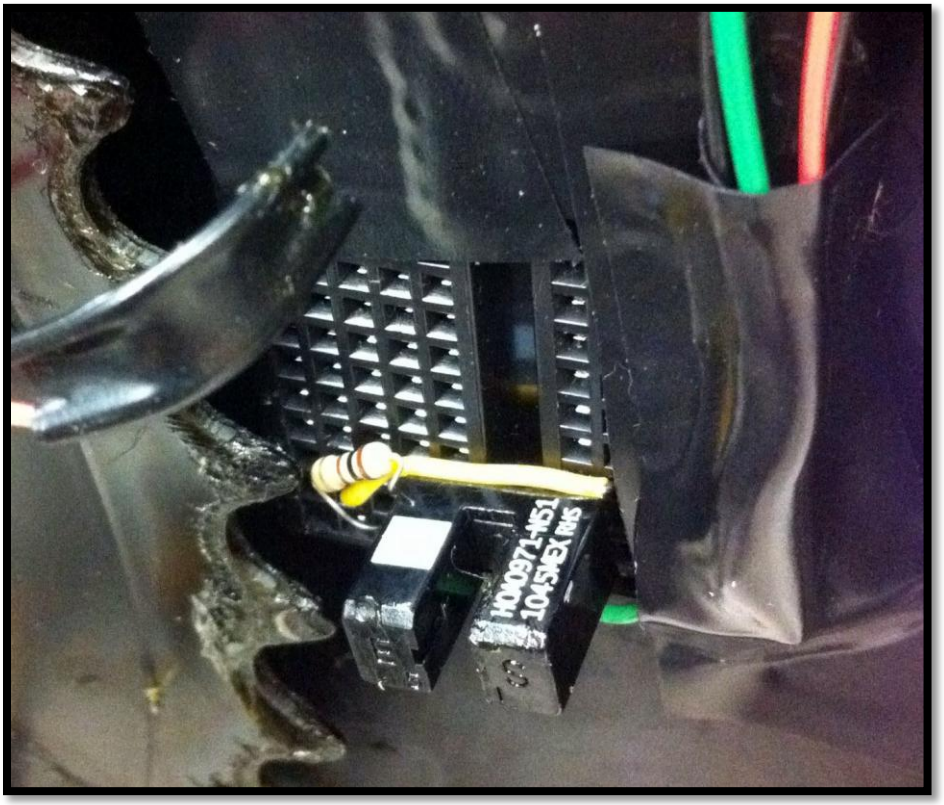


Figure 2.14: Photointerrupter and flag mounted near the pedal gear.

Chapter 3

Metric Calculation

In this chapter the mFit system's metric calculation algorithms are described.

3.1 Angular Velocity and Acceleration

The angular velocity of the wheel can be determined by multiplying the cadence calculated in the previous section, by the circumference of a circle, or 2π radians. The angular acceleration of wheel can be calculated by finding the difference in angular velocity of two successive rotations and dividing this value by the total time over which the velocity increased.

$$Angular\ Speed\ (Radian\ per\ Second) = \omega_{wheel} = \frac{2\pi}{60} RPM_{Current}$$

$$Angular\ Acceleration\ (Radians\ per\ Second^2) = \alpha_{wheel}$$

$$= \left(\frac{2\pi}{3600} * \ (RPM_{Current} - RPM_{Previous}) * RPM_{Current}\right)$$

Both velocity and acceleration of the pedal can be calculated by multiplying each of the previous equations by the gear ratio between the gear of the wheel and gear of the pedal.

$$\omega_{pedal} = \frac{2\pi}{60} * R * RPM_{Current}$$

$$\alpha_{pedal} = R\left(\frac{2\pi}{3600} * \ (RPM_{Current} - RPM_{Previous}) * RPM_{Current}\right)$$

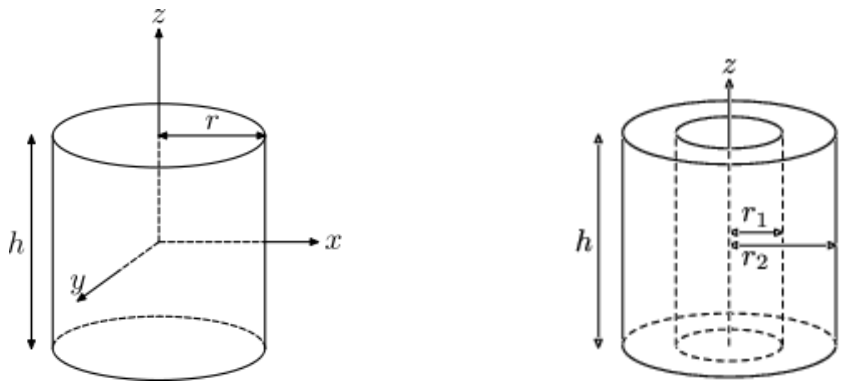
3.2 Torque and Force

The total torque applied by the user at the pedal is a combination of two forces resisting the pedal motion. The first is the moment of inertia of the cycle's 46.6lb flywheel. The second is the force of friction applied to the brakes at the outer rim of the flywheel.

The moment of inertia of the flywheel can be determined by estimating the distribution of weight of the wheel and using the appropriate equation for both a solid cylinder and a walled cylinder:

$$I_{solid\ cylinder} = \frac{mr^2}{2}$$

$$I_{walled\ cylinder} = \frac{1}{2}m(r_1^2 + r_2^2)$$



(a) (b)

Figure 3.1: Figure for calculating moment of inertia for solid cylinder (a) and walled cylinder (b) [15].

A weight of 46.6lb at normal earth gravity translates to 21.14kg of mass.

Estimating 9kg within the center with radius 19cm and 12.14kg circling the outer rim with outer radius 22.5cm, the moment of inertia is calculated:

$$I_{wheel} = I_{solid\ cylinder} + I_{walled\ cylinder} = \frac{mr^2}{2} + \frac{1}{2}m(r_1^2 + r_2^2)$$

$$= \frac{1}{2} * 9kg * (0.19m)^2 + \frac{1}{2} * 12.14kg * ((0.19m)^2 + (0.225m)^2$$

$$= 0.689kg * m^2$$

Torque is present when an inertial mass undergoes acceleration. Thus, the previously calculated angular acceleration will be combined with the moment of inertia of the wheel to determine the torque due to the mass of the flywheel.

$$\tau_{mass\ at\ wheel} = I_{wheel}\alpha_{wheel}$$

The resulting torque at the pedal is found through the product of the torque at the wheel and the gear ratio between the wheel and pedal gears.

$$\tau_{mass} = R * \tau_{mass\ at\ wheel} = RI_{wheel}\alpha_{wheel} = 1.607 * (0.689kg * m^2) * \alpha_{wheel}$$
$$= 1.107\alpha_{wheel}$$

The torque resulting from the inertial mass is only present during acceleration. Since such rapid acceleration of the flywheel is only present at the beginning of a workout and the inertial torque is much smaller than the torque resulting from the brake pads. Thus, the inertial torque was ignored in our calculations.

The second source of torque is the frictional force exerted onto the flywheel by the brake calipers. The torque resulting from the brake calipers is proportional to the torque at the pedals due to the direct coupling between the chain and sprockets.

$$\tau_{brakes} = \tau_{pedal}$$

The mFit cycle measures the torque at the pedal by measuring the torque exerted onto the brake calipers by the rotating flywheel via the strain gauge (τ_{brakes}). Flexing of strain gauge depends upon the cadence and tightness level of the brakes. This flex is proportional to the force at the pedal (F_{pedal}). By multiplying this force by the length of the pedal arm (r_{pedal}), the force at the pedal resulting from the brake pad resistance is determined.

$$\tau_{brakes} = r_{pedal}F_{pedal}\sin(\theta_{pedal})$$
$$F_{pedal} = \frac{\tau_{brakes}}{r_{pedal}(\sin(\theta_{pedal}))}$$

By placing the pedal such the force exerted on the pedal is perpendicular to the pedal arm θ_{pedal} approaches 90 degrees. Thus, the force equation can be simplified to:

$$F_{pedal} = \frac{\tau_{brakes}}{r_{pedal}}$$

Given the linear relationship between the torque resulting from the brakes and the force on the pedals, a simple method of calibration can be developed by placing a minimum of two known weights perpendicular to the pedal arm and measuring the strain gauge values. By drawing a best fit line between the points and calculating the slope (m) and y-intercept (b), an equation determining the force at the pedal via the strain sensor value (S) is found.

$$F_{pedal} = m * S + b$$

To attain torque applied to the pedal (τ_{pedal}) the force is multiplied by the length of the cycle's crank arm (L).

$$\tau_{pedal} = F_{pedal} * L = (m * S + b) * L$$

3.3 Power and Energy

The user input power can now be determined as the product of the total torque calculated in the previous section and angular velocity of the pedal calculated in section 3.1.

$$P_{pedal} = \tau_{pedal} \cdot \omega_{pedal}$$

Energy (*E*) is determined by multiplying the average power by the total time elapsed.

$$E_{total} = \sum_{0}^{x} P_{pedal,x} * (t_x - t_{x-1})$$

Chapter 4

Implementation

This chapter provides a brief introduction to the development environment before describing the algorithm and code used to determine calibration and cadence.

4.1 Arduino Development Environment [16]

All calculations were performed on the Arduino Pro Mini. Programming of the Arduino Pro Mini is done through an open source development environment that utilizes a C like language. Included in the development environment is: a text editor, serial monitor, and a message area for displaying errors in the compiling process.

Using the Arduino development environment allowed for rapid code development and easy prototyping; it further allowed the code to be hardware independent, as the code can be used on any of the Arduino boards.

4.2 Strain Sensor Calibration

To relate the strain sensor value to a force exerted on the pedal, calibration is required. It was shown that the relationship is linear; thus, a simple method of calibration would be to

place a minimum of two weights perpendicular to the pedal arm and measure the resulting strain sensor values. Figure 4.1 shows the result of 4 different weights (0lb, 25lb, 50lb, and 75lb) being placed twice on the pedal and the resulting ADC values. The slope and y-intercept of the best fit line provide the constants necessary for determining the equation that relates ADC values to the force on the pedal.

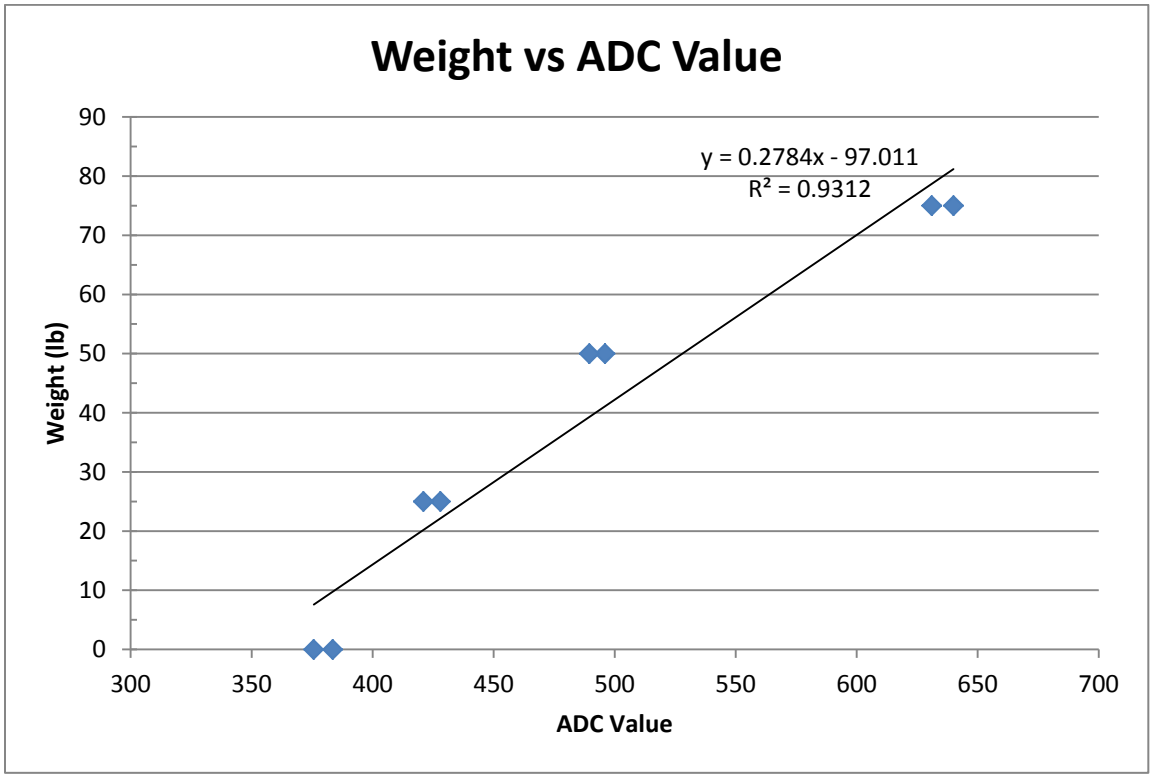


Figure 4.1: Plot showing relationship between the strain gauges ADC values and weight on the pedal.

The weight in pounds can be converted to force in Newton's by simple multiplication:

$$F_{pedal}(lb) = 0.2784 * (ADC\ Value) - 97.011$$

$$F_{pedal}(N) = (0.2784 * (ADC\ Value) - 97.011) * 4.448\frac{N}{lb}$$

$$= 1.238 * (ADC\ Value) - 431.5$$

4.3 Metric Calculation with Disturbance Detection

An analog input of the Arduino board was connected to the amplified strain gauge data. Sampling was performed using a 10 bit ADC at 100 Hz. Running averages of 20 ADC values were calculated to reduce noise. (Appendix A. 99-103)

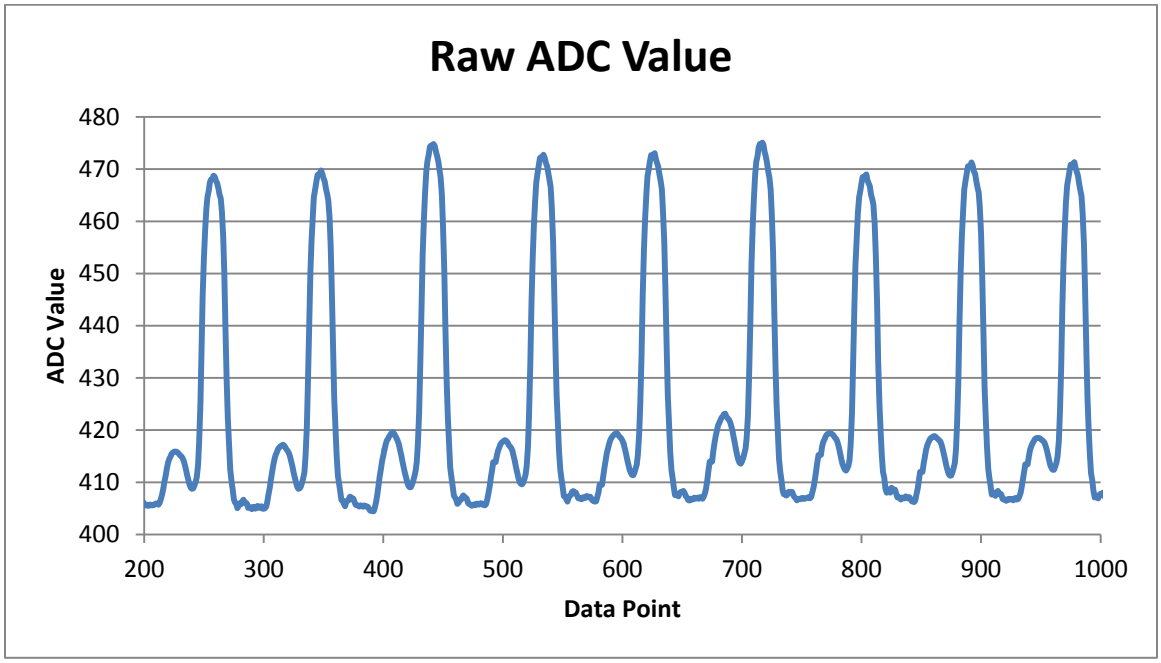


Figure 4.2: Raw ADC Value from the strain gauge.

To find cadence, two statistical metrics were used to accurately find the disturbance of the tape. The first is a 20 point standard deviation. (Appendix A. 120-124)

The second is the percentage offset between the current value and the previously found 20 point average. (Appendix A. 129)

Three thresholds need to be passed for a rotation to be registered. The first is surpassing an experimentally defined upper threshold of the percent error. (Green line, Figure 4.4) The second is surpassing a set upper threshold of the standard deviation. (Green line, Figure 4.3) The third is falling below a lower percent error threshold. (Red line, Figure 4.4) The upper standard deviation threshold is initially set to an experimentally determined value but later varies with the amplitude of the previously calculated standard deviations. To ensure proper cadence calculation the order of each passing threshold must be kept. (Appendix A. 155-210)

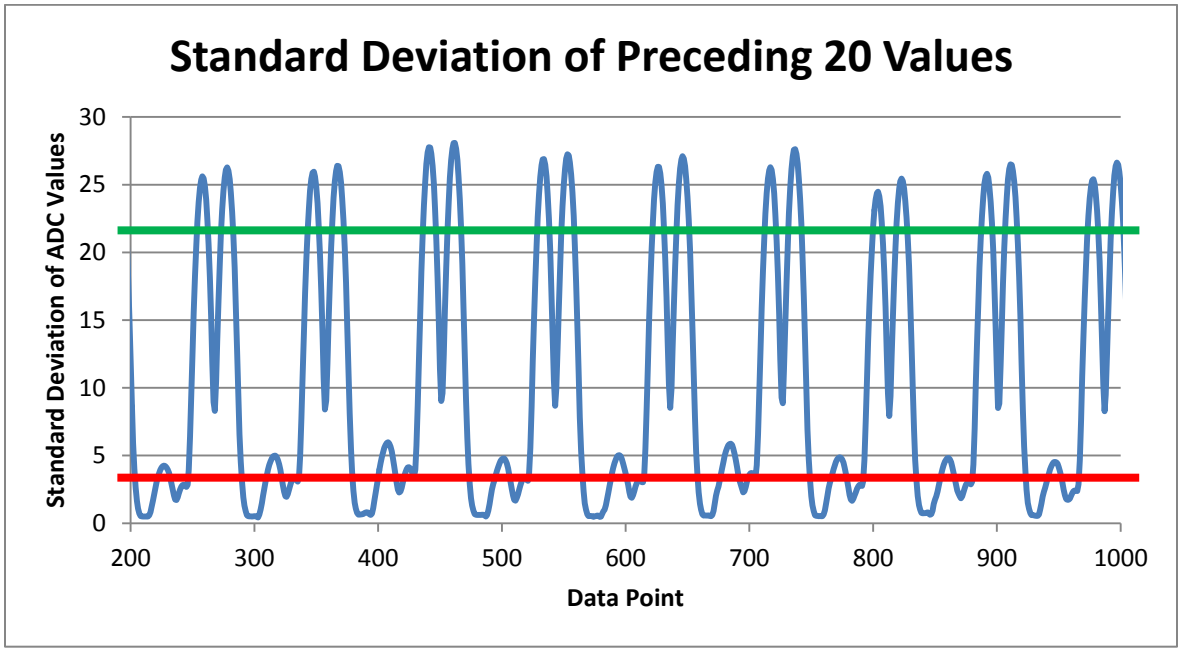


Figure 4.3: Standard deviation of the preceding 20 values.

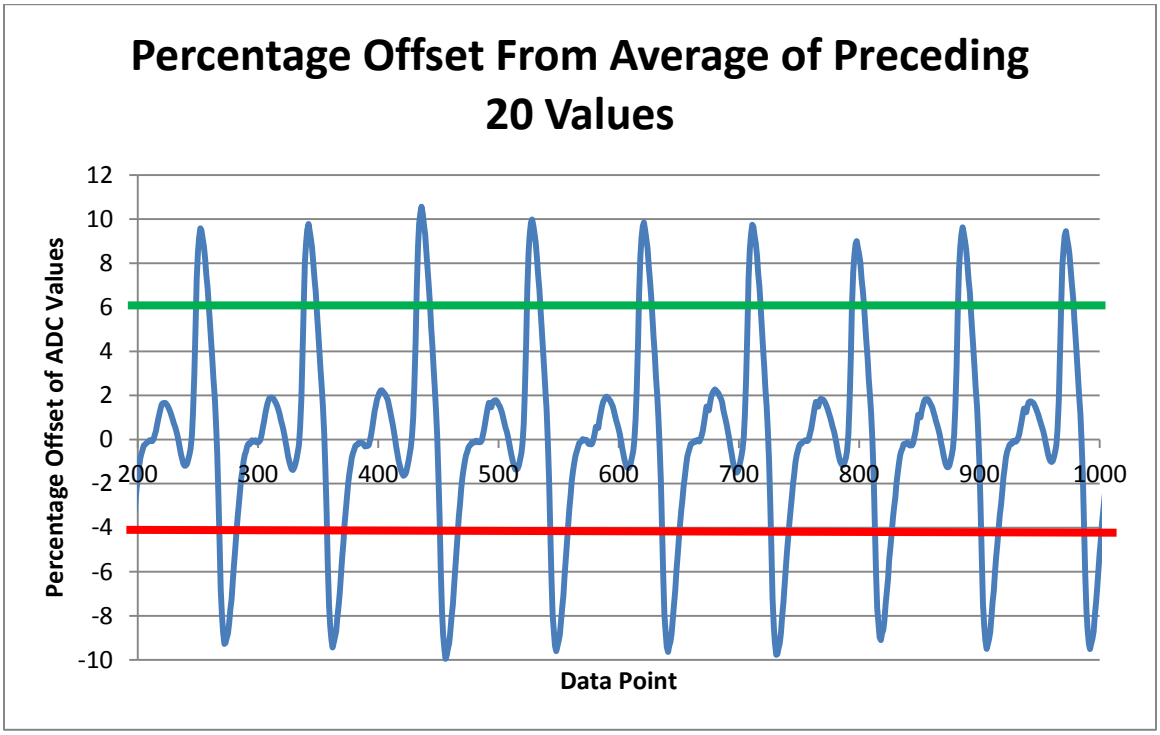


Figure 4.4: Percentage offset from average of preceding 20 values.

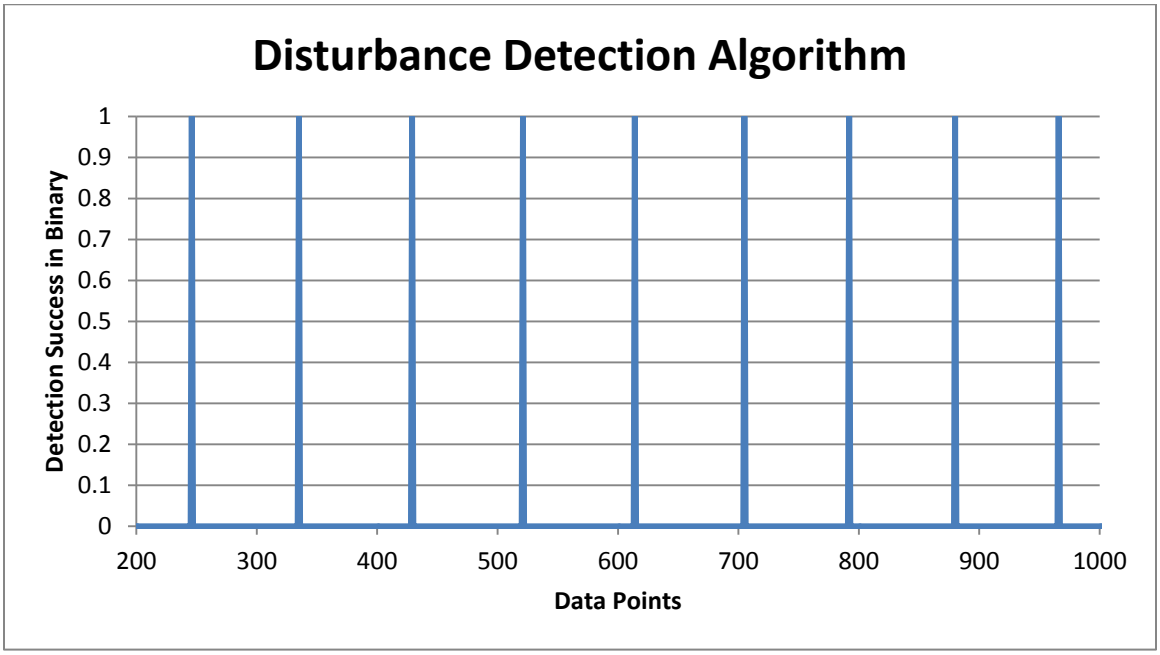


Figure 4.5: Results of the disturbance detection algorithm.

Three additional conditions help to reduce error in the data. The first waits for 20 rotations of the flywheel so an accurate standard deviation value can be calculated. (Appendix A. 89-103) The second detects rapid changes in cadence and ignores the following 3 values to ensure stabilization. (Appendix A. 143-145) The third resets the variables in the system after 3 seconds of inactivity. (Appendix A. 134-137)

4.3 Metric Calculation with Photointerrupter

The addition of a photointerrupter simplifies the code as a disturbance detection algorithm is no longer needed. The interrupt function of the microprocessor was used to calculate the cadence. As the flag intercepts the photointerrupter beam with each rotation of the pedal, the Pro Mini registers an interrupt, logs a timestamp, and calculates the cadence using the previous timestamps. (Appendix B. 164-185)

Chapter 5

Data Collection

The code in Appendix B was used to run two experiments to measure the calculated power values when the cadence and brake resistance were varied.

5.1 Power Measurement – Varying Cadence

The first test measures the power by varying cadence in increments of 20 rotations per second, every 10 seconds.

Figure 5.1 shows the resulting raw strain sensor measurements. The lack in fluctuation results from the decreasing coefficient of friction negating the strain that would have been caused by the increased cadence.

Figure 5.2 shows the cadence values. The calculated power values are seen in Figure 5.3. The strain sensor is sampled at 100Hz and thus shows better granularity as compared to the cadence or power values which are calculated once per rotation.

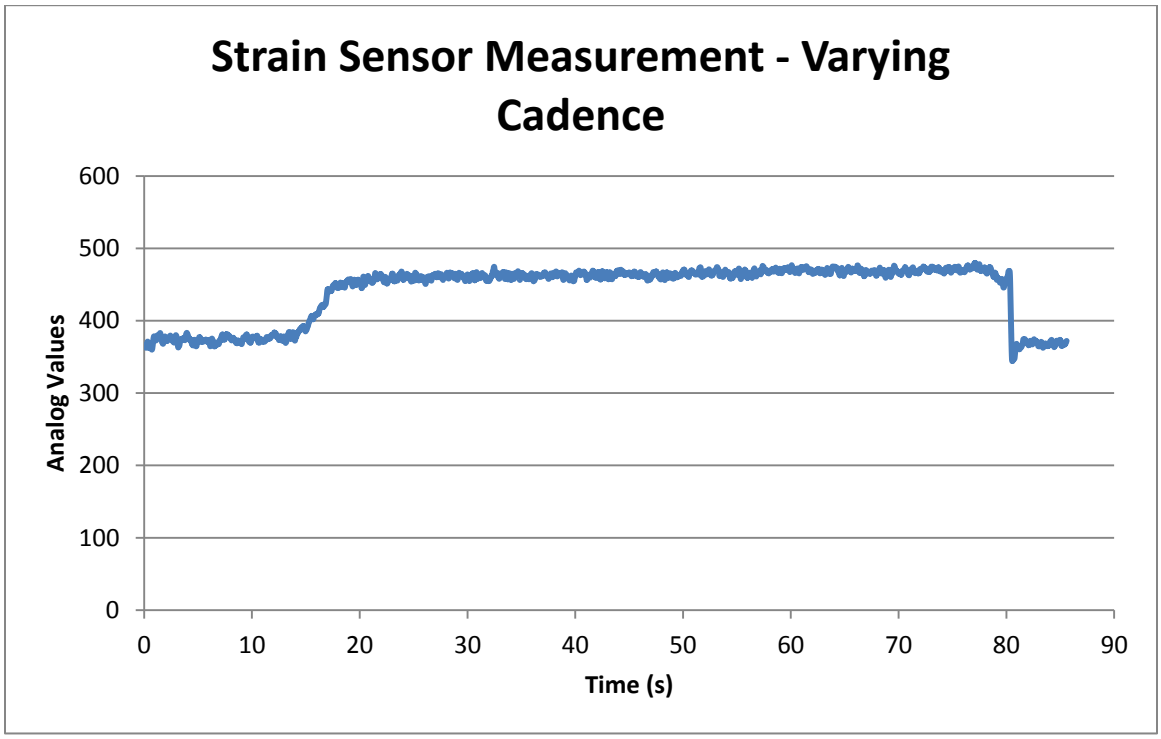


Figure 5.1: Resulting strain sensor measurement at varying cadence values.

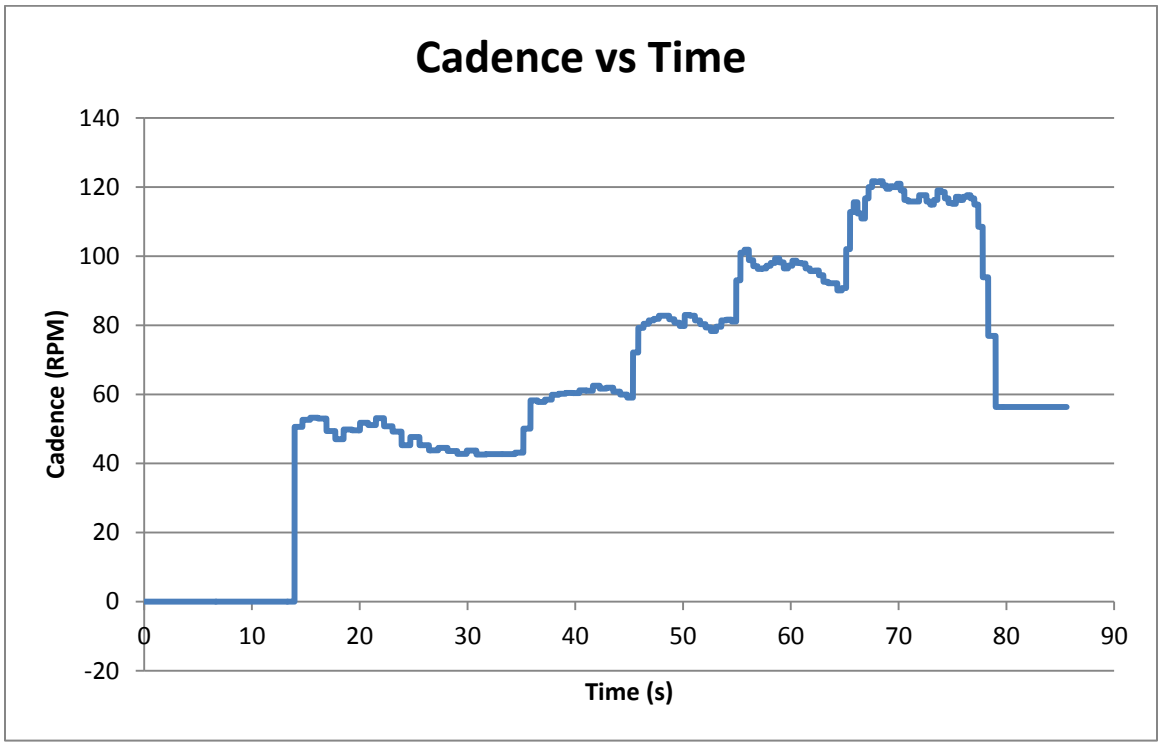


Figure 5.2: Variations in cadence in increments of 20 RPM, every 10 seconds.

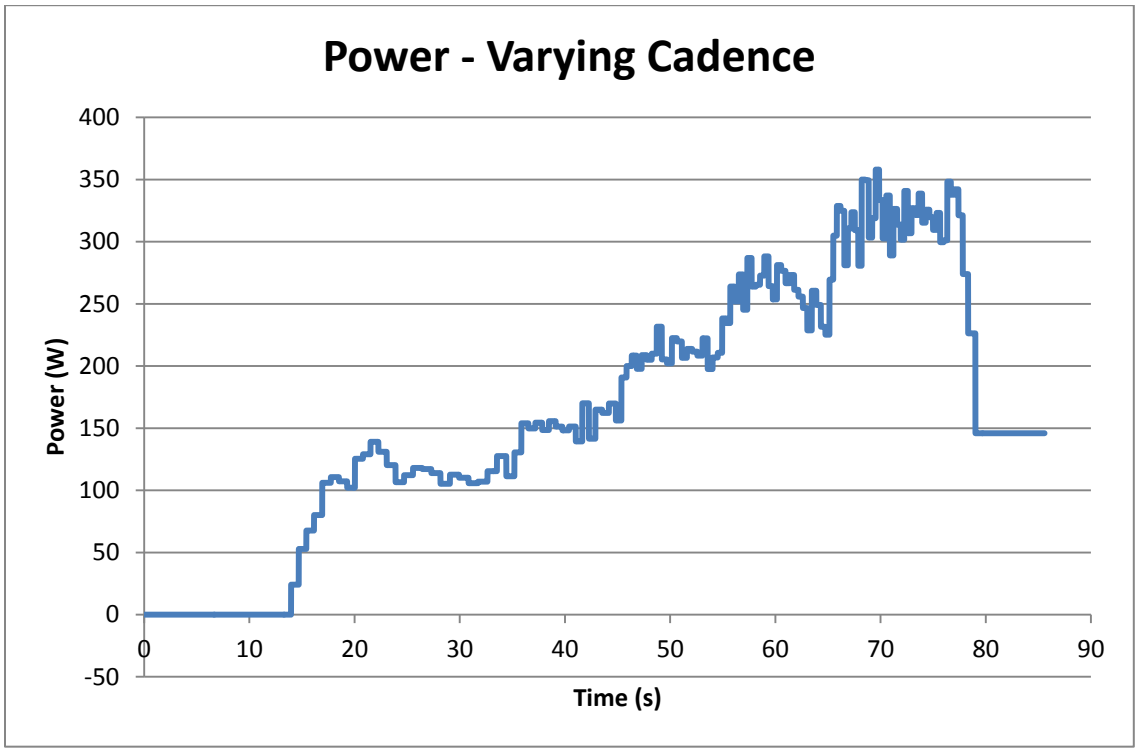


Figure 5.3: Calculated power at varying cadence values.

5.2 Power Measurement – Varying Brake Resistance

The second test measures the power while varying the brake resistances by turning the resistance knob a quarter turn every 20 seconds. Cadence was kept within a constant 60 rotations per second.

Figure 5.4 shows the resulting raw strain sensor measurements. Figure 5.5 shows the calculated power measurements. Due to the consistency at which the cadence was maintained, the power values closely mirror the change seen by the strain sensor.

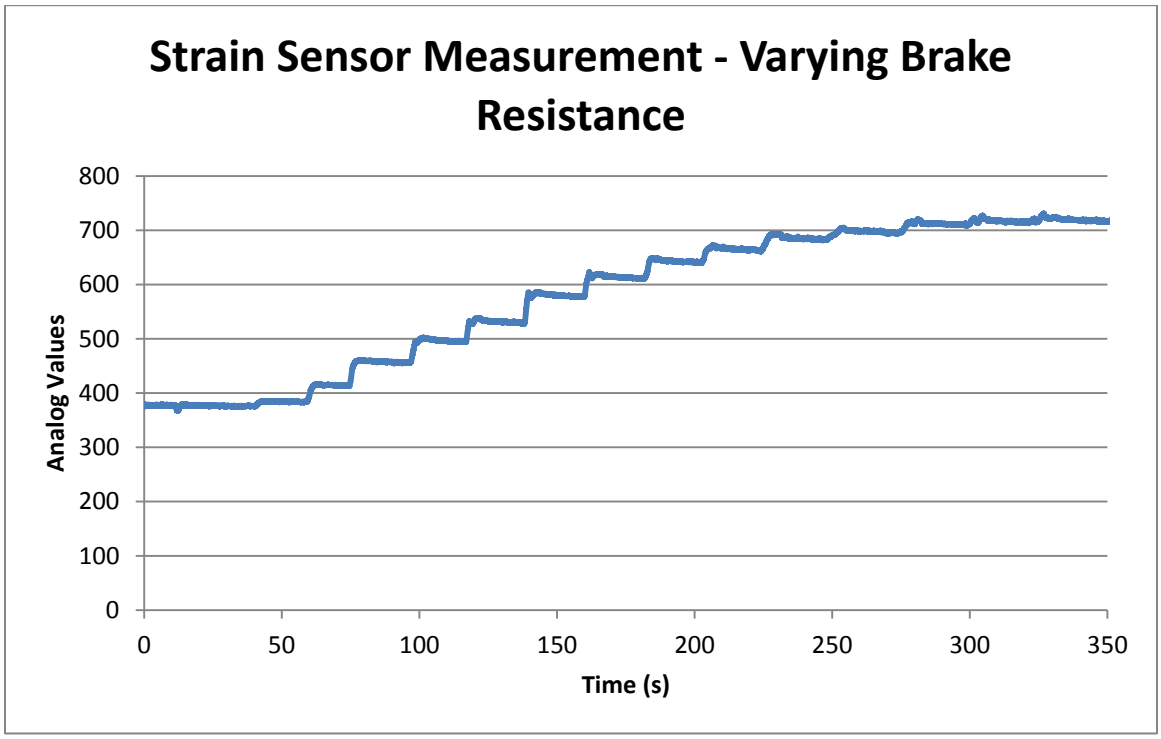


Figure 5.4: Resulting strain sensor measurement at varying brake resistance values.

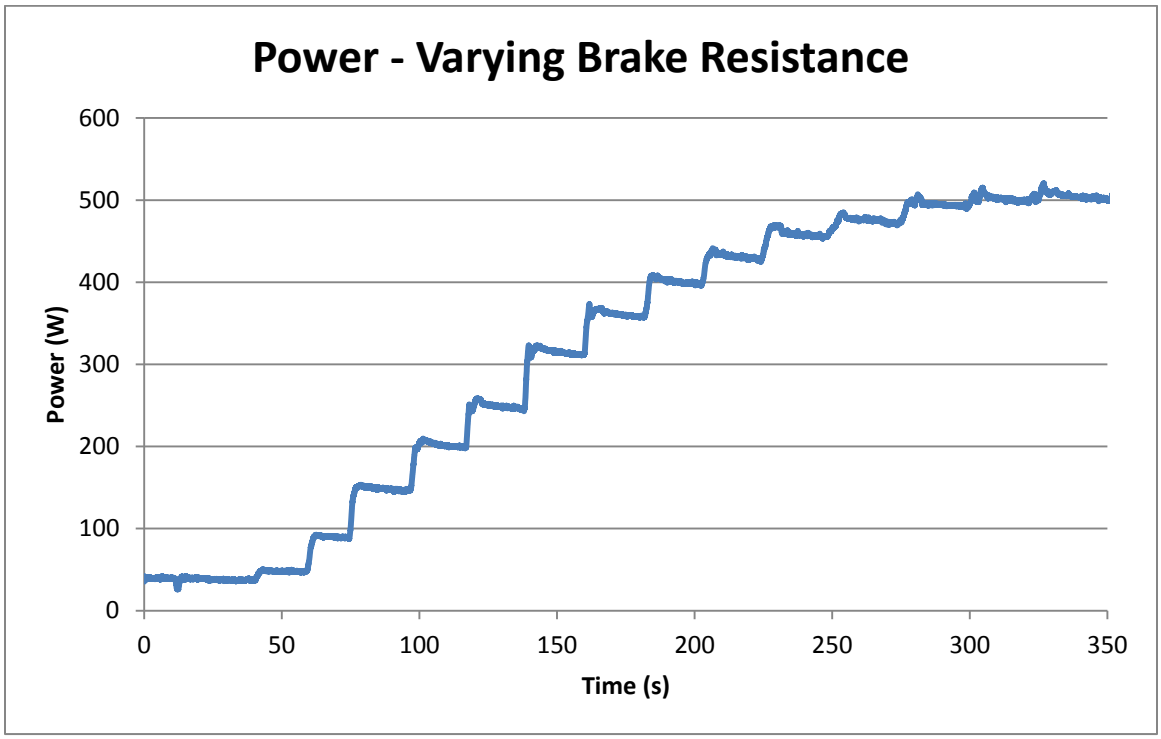


Figure 5.5: Calculated power at varying brake resistance values.

Chapter 6

Mobile Platform

The mFit system is able to connect to any device that accepts Serial Port Profile (SPP) Bluetooth protocol. This enables connectivity to a whole host of applications on a multitude of Bluetooth enabled platforms. Two examples of such applications built on the Android platform are described.

6.1 Metric Monitor

The mobile platform serves as the means to integrate the data and present it to the user in a convenient form. The current platform of choice is the Android operating system. Androids ubiquity and large user base make it an ideal choice. Figure 6.1 and 6.2 show how the mobile platform is placed atop the stationary cycle's handle bar.

The mFit User Metric Monitor application is built to show all the relevant metrics in real time. Heart rate information is gathered via the Zypher HxM, transmitting heart rate data via a separate Bluetooth stream to the Android platform. After the application connects to both the heart rate monitor and mFit system a swipe to the left reveals the metrics dashboard. (Figure 6.3) To differentiate between the measured or calculated

values transmitted by the Arduino via Bluetooth, a letter, either A, R, or P for analog data, RPM, or power data respectively, is added to the beginning of each transmission and a lookup table on the Android platform parses the data and displays it on the dashboard. An example data segment is shown in table 6.1.

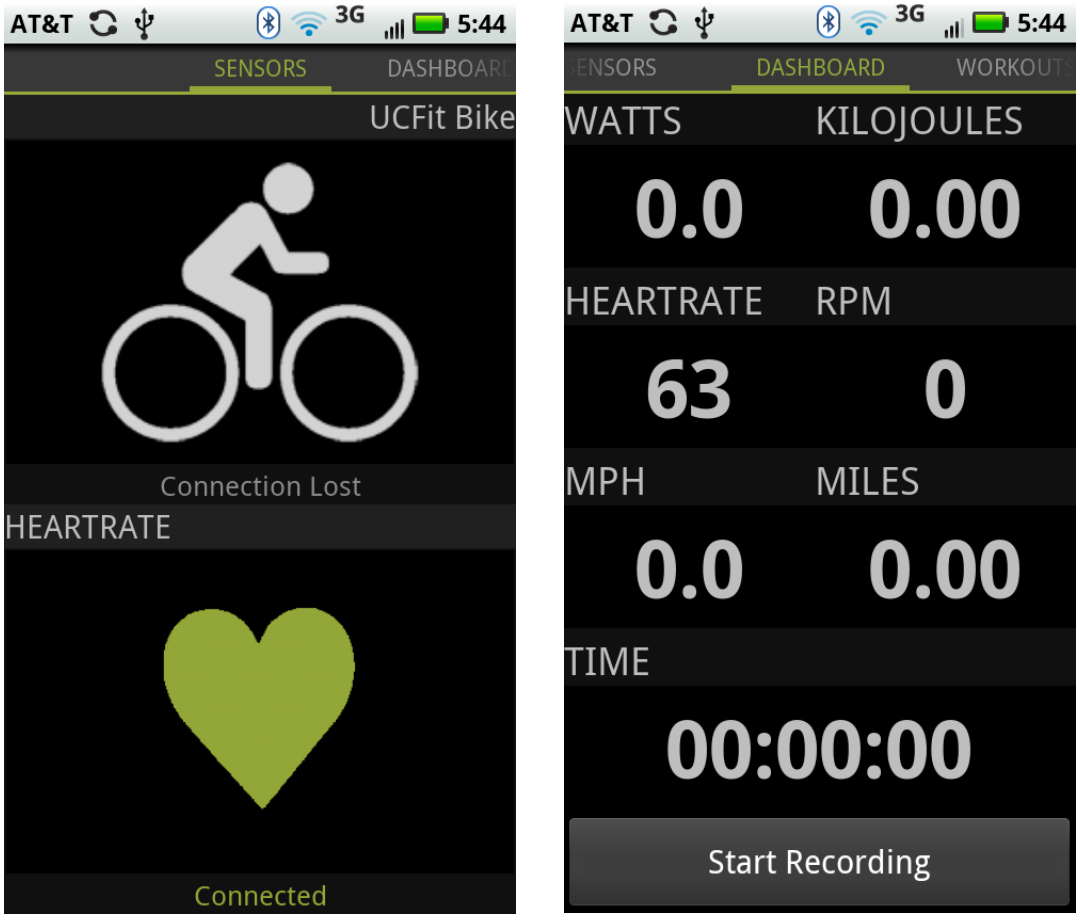
A366.05
A363.55
A360.70
A368.35
A375.60
A376.45
A378.60
A382.25
A374.85
A375.50
A376.60
A369.55
R50.59
P24.11
A370.35
A380.00
A383.85

Table 6.1: Data segment streamed to the Android platform. The letters at the beginning are used to parse the information.

Figure 6.1: Cycle handle bar showing placement of mobile platform.

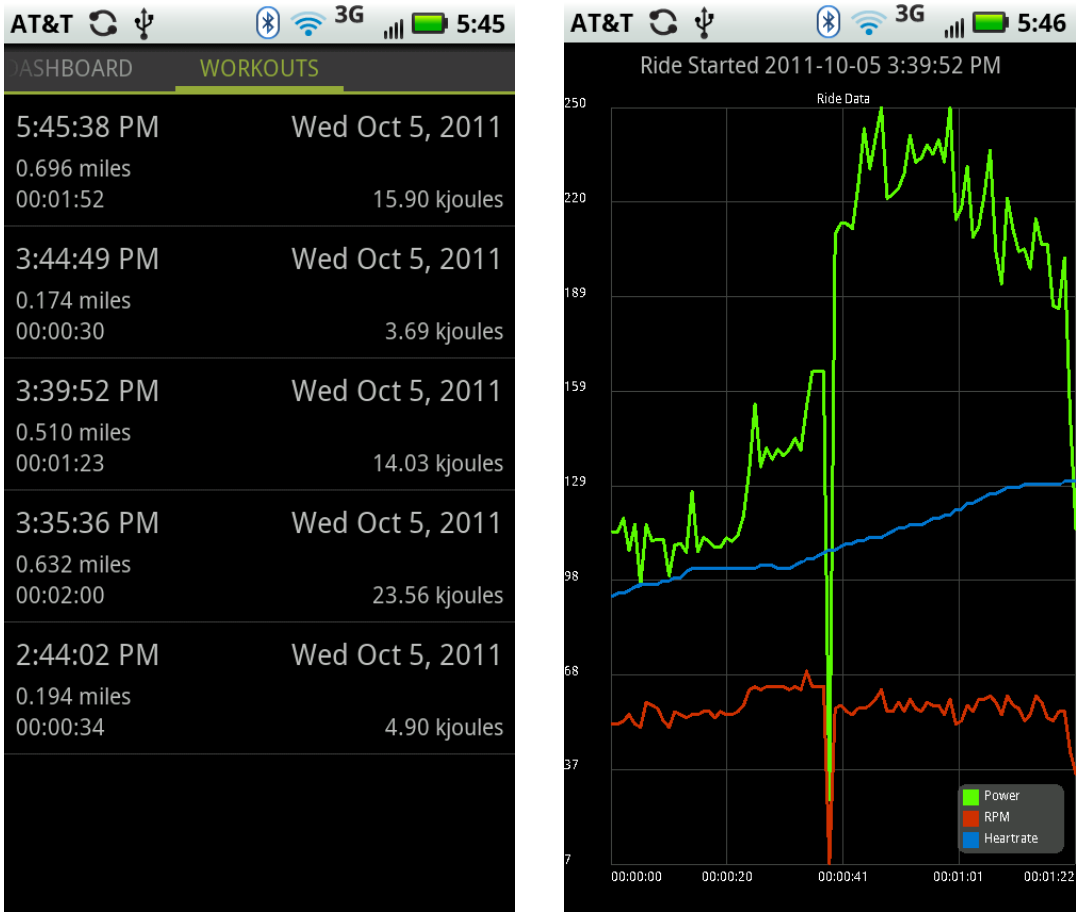
Figure 6.2: Cycle outfitted with mFit system showing placement of mobile platform.

On the dashboard is the user's: power input in watts, energy expenditure in kilojoules, heart rate, RPM, distance traveled in miles, and time elapsed during the workout. The button at the bottom allows the users to set the beginning and end of a workout. Each workout is immediately stored on the phone. Another swipe to the left reveals the workouts page which shows a brief time-stamped summary of the most recent workouts. (Figure 6.4a) Selecting one of the workouts presents a detailed graph showing power, RPM, and heart rate. (Figure 6.4b)



(a) (b)

Figure 6.3: mFit User Metric Monitor application screenshot showing: (a) connectivity to mFit system and heart rate monitor; (b) dashboard with numerical data presented.



(a) (b)

Figure 6.4: mFit User Metric Monitor application screenshot showing: (a) workout summaries of previous data; (b) graphical representation of workout data.

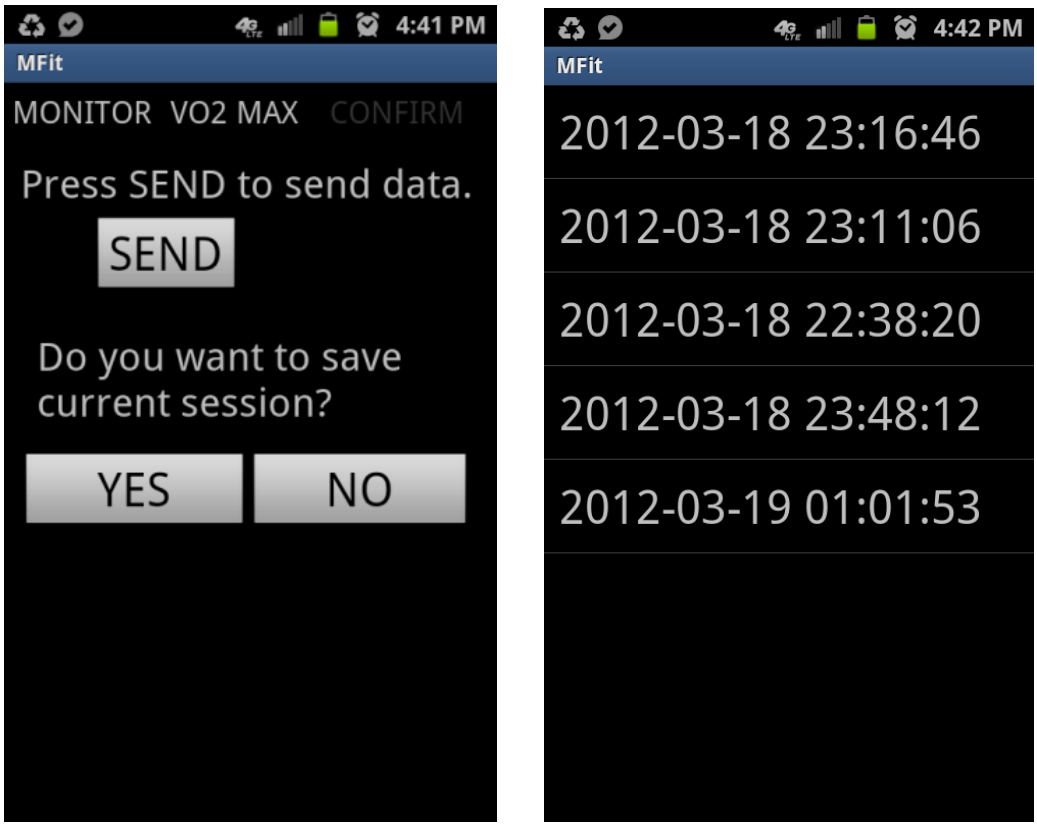
6.2 Submaximal VO_2

The second application also utilizes the Android platform to provide the user with a fitness and endurance metric. The application leverages the YMCA protocol to calculate

a submaximal VO_2. Step by step guidance is provided via interactive prompts. Extensive error checking and real-time data ensures the user remains within the protocols strict thresholds. After the protocol has run to completion the user is presented with their VO_2 max (Figure 6.5b) and given the option of saving or rejecting the session. The user can also select the data to be uploaded to a remote server. (Figure 6.6a)

(a) (b)

Figure 6.5: mFit Submaximal VO_2 application screenshot showing: (a) user metric dashboard; (b) VO_2 fitness dashboard



(a) (b)

Figure 6.6: mFit Submaximal VO_2 application screenshot showing: (a) session save and transmit screen; (b) workout summaries of previous data

Chapter 7

Conclusion and Future Work

To help combat the increase in health related costs, it is important for physicians to provide effective solutions to help speed the recovery of patients and reduce the rate of re-admission. With evidence showing the effectiveness of exercise as a treatment for a diverse set of conditions, it becomes important to ensure doctor prescribed exercise regiments are being followed. The mFit system described in this thesis provides a cost effective and convenient solution to hospitals and physicians looking to ensure patients are fulfilling these prescription. The device's mobility allows for monitoring of patients both at the clinic and at home, while keeping the physician informed of all activities.

The mFit exercise system's monitoring of cadence, applied force, generated power, and dissipated energy provides the necessary metrics to measure fitness, such as submaximal VO_2. The addition of Bluetooth connectivity allows for the transmission of real time data to a mobile device, making it accessible to the user via a graphical user interface. Furthermore, by transmitting the data to a central server the patient can be monitored by a physician.

Bluetooth's ubiquity allows for many devices to link with the mFit system. This enables new applications, developed in-house or by third parties, to expand the capabilities of the mFit ecosystem. Furthermore, the mFit's ease of implementation lays

the groundwork for expansion into other resistive based exercise platforms such as ellipticals and rowing machines.

Appendix A

Arduino Source Code with Disturbance Detection

```
#include <Wire.h>
#include <Math.h>

// pins
int sensorPin = A3;                  // Input pin for the strain gauge

// constants
const int sampleTime = 2.5;          // Millisecond delay between each sample
const int downThreshold = 2;         // Lower threshold for identifying a cycle
                                     // using the standard deviation of the ADC
                                     // value.
const int upPercent = 8;             // Upper threshold for identifying a cycle
                                     // using the variance from the average ADC
                                     // value.
const int downPercent = -3;          // Lower threshold for identifying a cycle
                                     // using the variance from the average ADC
                                     // value.
const int number_of_items = 20;      // Number of data points used to calculate
                                     // the average and standard deviation of
                                     // strain gauge values.
const double pi = 3.141592;          // Constant PI
const float gearRatio = 3.25;        // Gear ratio between wheel gear and pedal
                                     // gear
```

```
const float inertialFactor = 1.107;   // Moment of Inertia (0.689 kg*m)* Gear
                                      // ratio between wheel gear and pedal gear
                                      // (1.607)
const float pedalFactor = 0.175;      // Distance in meters between the center of
                                      // pedal gear, to the center of the pedal.

                                      // variables
int upThreshold = 12;                 // Upper threshold for identifying a cycle
                                      // using the standard deviation of the ADC
                                      // value.
int sensorValue[20];                  // ADC value from the strain gauge
unsigned long pMillis = 0;            // Stores the last time an ADC value was
                                      // sampled
int cnt = 0;                          // 0-19 loop for the sensorValue array
bool cnt1 = 0;                        // Binary value for looping through tStamp
                                      // and rpm arrays
int cnt2 = 2;                         // 0-2 loop for the timeDiff array
int cnt3 = 0;                         // Loop delays the printing of RPM and power
                                      // data till a set number of points is gathered
int cnt4 = 0;                         // Loop limiting the number of ADC values
                                      // output over bluetooth
float standard_deviation = 0;         // The standard deviation of the preceding
                                      // 9 strain gauge values.
float numerator = 0;                  // Standard deviations numerator value.
bool flag1 = 1;                       // Flag activated when the first threshold
                                      // for determining a cycle is passed
bool flag2 = 1;                       // Flag activated when the second threshold
                                       // for determining a cycle is passed
bool flag3 = 1;                       // Flag activated when the third threshold
                                      // for determining a cycle is passed
unsigned long tStamp[2] = {
 0,0};                                // Time stamp when first threshold for
                                      // determining a cycle is passed
float stDevMax = 0;                   // Determines the maximum standard deviation
                                      // value at which the timestamp is taken
float rpm[2] = {
 0,0};                                // Calculated RPM values
```

```
int i = 0;                          // Counter
float sensorAvg = 0;                // Average of sensorValue values.
unsigned long timeDiff[3] = {
 0,0,0};                            // Time difference between succeeding cycles
unsigned long totalTimeDiff = 0;    // Sum of three timeDiff's
float torqTotal = 0;                // Total torque input by the user
float power = 0;                    // Input power of user
float percentError = 0;             // Calculating the deviation of the current
                                    // sensorValue from the preceding average.

//*************************************************************************//

void setup()
{
 Serial.begin(115200);              // Bluetooth serial set to 115200 baud rate
}

void loop()
{
 /*************************************************************************
  ** Loop which reads the strain gauge ADC value every [sampleTime]
  *************************************************************************/
 if (millis() - pMillis > sampleTime) {
  sensorValue[cnt] = analogRead(sensorPin);

  pMillis = millis();

  cnt++;
  if (cnt >= number_of_items)  {
   cnt = 0;
  }

  sensorAvg = 0;
  numerator = 0;

  /*************************************************************************
   ** Running average of [number_of_items] strain gauge data
```

```
 ******************************************************************************/
for (i = 0; i < number_of_items; i++)  {
  sensorAvg = sensorAvg + sensorValue[i];
}

sensorAvg = sensorAvg / number_of_items;

/*******************************************************************************
 ** Print the average strain gauge data at 100Hz
 ******************************************************************************/
if (cnt4 == 3)  {
  Serial.print("A");
  Serial.println(sensorAvg);
  cnt4 = 0;
}
else  {
  cnt4++;
}

/*******************************************************************************
 ** Standard deviation of the previous [number_of_items] strain gauge values
 ******************************************************************************/
for (i = 0; i < number_of_items; i++) {
  numerator = numerator + pow((sensorValue[i] - sensorAvg), 2);
}

standard_deviation = sqrt (numerator/(number_of_items-1));

/*******************************************************************************
 ** Percentage offset of the previous [number_of_items] strain gauge values
 ******************************************************************************/
percentError = (sensorValue[cnt] - sensorAvg) * 100 / sensorAvg;

/*******************************************************************************
 ** Reset delay and threshold values if no rotation is detected for 3 seconds
 ******************************************************************************/
if (millis() - tStamp[cnt1] > 3000)  {
```

```
 cnt3 = 0;
 upThreshold = 12;
}

/*****************************************************************************
 ** Rapid RPM change results in skipping the next 3 values to check for
 ** consistency and minimize chances of error
 *****************************************************************************/
if (rpm[cnt1]-rpm[!cnt1]>20)  {
 cnt3 = 3;
}

/*****************************************************************************
 ** RPM algorithm: The strain gauge values are monitored for the unique
 ** signature of a disturbance. 3 flags need to be reset before an rotation is
 ** registered. Each flag represents a unique threshold. The first is surpassing
 ** a set upper threshold of the percent error. The second is surpassing a set
 ** upper threshold of the standard deviation. The third is falling below a
 ** lower percent error threshold.
 *****************************************************************************/
if (flag1 && percentError > upPercent)  {
 flag1 = 0;
 cnt1 = !cnt1;
 tStamp[cnt1] = millis();
 stDevMax = standard_deviation;
}

else if (!flag1 && flag2 && flag3 && standard_deviation > upThreshold)  {
 if (millis() - tStamp[cnt1] < 300)  {
  flag3 = 0;
  if (standard_deviation > stDevMax)  {
   stDevMax = standard_deviation;
  }
 }
 else  {
  flag1 = 1;
  flag2 = 1;
```

```
    }
   }
   else if  (!flag3 &&  percentError > downPercent && stDevMax <
standard_deviation)  {
    stDevMax = standard_deviation;
   }

   else if  (!flag3 &&  percentError < downPercent)  {
    if (standard_deviation > stDevMax)  {
     stDevMax = standard_deviation;
    }
    upThreshold = 0.7 * stDevMax;

    flag1 = 1;
    flag2 = 1;
    flag3 = 1;
    cnt2++;
    if (cnt2 >=3)  {
     cnt2 = 0;
    }
    timeDiff[cnt2] = tStamp[cnt1]-tStamp[!cnt1];
    for (i = 0; i < 3; i++)  {
     totalTimeDiff = totalTimeDiff + timeDiff[i];
    }
    rpm[cnt1] = timeDiff[cnt2];
    rpm[cnt1] = (1 / (rpm[cnt1] / 60000)) / gearRatio;

    torqTotal = (1.2358 * sensorAvg - 434.85) * pedalFactor;
    power = torqTotal * (2 * pi * rpm[cnt1] / 60);
    cnt3++;

    if (cnt3 > 5 && rpm[cnt1]<200)  {
     Serial.print("R");
     Serial.println(rpm[cnt1]);
     Serial.print("P");
     Serial.println(power);
    }
```

```
    }
  }
}
```

Appendix B

Arduino Source Code with Photointerrupter

```
#include <Wire.h>
#include <Math.h>

// pins
int sensorPin = A3;                 // Input pin for the strain gauge

// constants
const int sampleTime = 2.5;         // Millisecond delay between each sample
const int downThreshold = 2;        // Lower threshold for identifying a cycle
                                    // using the standard deviation of the ADC
                                    // value.
const int upPercent = 8;            // Upper threshold for identifying a cycle
                                    // using the variance from the average ADC
                                    // value.
const int downPercent = -3;         // Lower threshold for identifying a cycle
                                    // using the variance from the average ADC
                                    // value.
const int number_of_items = 20;     // Number of data points used to calculate
                                    // the average and standard deviation of
                                    // strain gauge values.
const double pi = 3.141592;         // Constant PI
const float gearRatio = 3.25;       // Gear ratio between wheel gear and pedal
                                    // gear
```

```
const float inertialFactor = 1.107;   // Moment of Inertia (0.689 kg*m)* Gear
                                      // ratio between wheel gear and pedal gear
                                      // (1.607)
const float pedalFactor = 0.175;      // Distance in meters between the center of
                                      // pedal gear, to the center of the pedal.

// variables
int upThreshold = 12;                 // Upper threshold for identifying a cycle
                                      // using the standard deviation of the ADC
                                      // value.
int sensorValue[20];                  // ADC value from the strain gauge
unsigned long pMillis = 0;            // Stores the last time an ADC value was
                                      // sampled
int cnt = 0;                          // 0-19 loop for the sensorValue array
bool cnt1 = 0;                        // Binary value for looping through tStamp
                                      // and rpm arrays
int cnt2 = 2;                         // 0-2 loop for the timeDiff array
int cnt3 = 0;                         // Loop delays the printing of RPM and power
                                      // data till a set number of points is gathered
int cnt4 = 0;                         // Loop limiting the number of ADC values
                                      // output over bluetooth
float standard_deviation = 0;         // The standard deviation of the preceding
                                      // 9 strain gauge values.
float numerator = 0;                  // Standard deviations numerator value.
bool flag1 = 1;                       // Flag activated when the first threshold
                                      // for determining a cycle is passed
bool flag2 = 1;                       // Flag activated when the second threshold
                                      // for determining a cycle is passed
bool flag3 = 1;                       // Flag activated when the third threshold
                                      // for determining a cycle is passed
unsigned long tStamp[2] = {
 0,0};                                // Time stamp when first threshold for
                                      // determining a cycle is passed
float stDevMax = 0;                   // Determines the maximum standard deviation
                                      // value at which the timestamp is taken
float rpm[2] = {
 0,0};                                // Calculated RPM values
```

```
int i = 0;                              // Counter
float sensorAvg = 0;                    // Average of sensorValue values.
unsigned long timeDiff[3] = {
 0,0,0};                                // Time difference between succeeding cycles
unsigned long totalTimeDiff = 0;        // Sum of three timeDiff's
float torqTotal = 0;                    // Total torque input by the user
float power = 0;                        // Input power of user
float percentError = 0;                 // Calculating the deviation of the current
                                        // sensorValue from the preceding average.
bool vertCheckf1 = 0;

//*************************************************************************************/
/

void setup()
{
 Serial.begin(115200);                  // Bluetooth serial set to 115200 baud rate
 attachInterrupt(0, rpmCalc, RISING);  // Interrupt added to pin 2 to detect rising
edge of photointerrupter.
}

void loop()
{
 /*************************************************************************************
  ** Loop which reads the strain gauge ADC value every [sampleTime]
  *************************************************************************************/
 if (millis() - pMillis > sampleTime) {
  sensorValue[cnt] = analogRead(sensorPin);

  pMillis = millis();

  cnt++;
  if (cnt >= number_of_items)  {
   cnt = 0;
  }

  sensorAvg = 0;
```

```
numerator = 0;

/*****************************************************************************
 ** Running average of [number_of_items] strain gauge data
 *****************************************************************************/
for (i = 0; i < number_of_items; i++)  {
  sensorAvg = sensorAvg + sensorValue[i];
}

sensorAvg = sensorAvg / number_of_items;

/*****************************************************************************
 ** Print the average strain gauge data at 100Hz
 *****************************************************************************/
if (cnt4 == 3)  {
  Serial.print("A");
  Serial.println(sensorAvg);
  cnt4 = 0;
}
else  {
  cnt4++;
}

/*****************************************************************************
 ** Standard deviation of the previous [number_of_items] strain gauge values
 *****************************************************************************/
for (i = 0; i < number_of_items; i++) {
  numerator = numerator + pow((sensorValue[i] - sensorAvg), 2);
}

standard_deviation = sqrt (numerator/(number_of_items-1));

/*****************************************************************************
 ** Percentage offset of the previous [number_of_items] strain gauge values
 *****************************************************************************/
percentError = (sensorValue[cnt] - sensorAvg) * 100 / sensorAvg;
```

```
  /*****************************************************************************
   ** Reset delay and threshold values if no rotation is detected for 3 seconds
   *****************************************************************************/
  if (millis() - tStamp[cnt1] > 3000)  {
    cnt3 = 0;
    upThreshold = 12;
  }

  /*****************************************************************************
   ** Rapid RPM change results in skipping the next 3 values to check for
   ** consistency and minimize chances of error
   *****************************************************************************/
  if (rpm[cnt1]-rpm[!cnt1]>20)  {
    cnt3 = 3;
  }

  if (cnt3 > 5 && rpm[cnt1]<200 && flag1)  {
    Serial.print("R");
    Serial.println(rpm[cnt1]);
    Serial.print("P");
    Serial.println(power);
    flag1 = 0;
  }
 }
}


/*****************************************************************************
 ** RPM detection: Each interruption of a photointerrupt results on an interrupt
 ** call on rpmCalc which grabs a timestamp and calculates the current RPM.
 *****************************************/
void rpmCalc () {
 if (vertCheckf1) {
  cnt1 = !cnt1;
  tStamp[cnt1] = millis();
  timeDiff[cnt2] = tStamp[cnt1]-tStamp[!cnt1];
  for (i = 0; i < 3; i++)  {
```

```
    totalTimeDiff = totalTimeDiff + timeDiff[i];
   }
   rpm[cnt1] = timeDiff[cnt2];
   rpm[cnt1] = (1 / (rpm[cnt1] / 60000));

   torqTotal = (1.238 * sensorAvg - 431.5) * pedalFactor;
   power = torqTotal * (2 * pi * rpm[cnt1] / 60);
   cnt3++;
   flag1 = 1;
   vertCheckf1 = 0;
  }
  else  {
   vertCheckf1 = 1;
  //  Serial.println("F");
  }
}
```

References

[1] J. Myers, "Exercise and Cardiovascular Health," *Circulation,* vol. 107, pp. e2-e5, 2003.

[2] B. Dobkin, "Training and exercise to drive poststroke recovery," *Nature Clinical Practice Neurology,* vol. 4, pp. 76-85, 2008.

[3] E. Rydwik, K. Frändin and A. G., "Physical training in institutionalized elderly people with multiple diagnoses--a controlled pilot study," *Archives of Gerontology and Geriatrics,* vol. 40, no. 1, pp. 29-44, 2005.

[4] D. Needham, R. Korupolu, J. Zanni, P. Pradhan, E. Colantuoni, J. Palmer, R. Brower and E. Fan, "Early physical medicine and rehabilitation for patients with acute respiratory failure: a quality improvement project.," *Archives of Physical Medicine and Rehabilitation,* vol. 91, no. 4, pp. 536-542, 2010.

[5] N. F. Gordon MD PhD Cochair, M. Gulanick PhD APRN Cochair, F. Costa MD, G. Fletcher MD, B. A. Franklin PhD, E. J. Roth MD and T. Shephard RN MSN, "Physical Activity and Exercise Recommendations for Stroke Survivors," *Circulation,* vol. 109, pp. 2031-2041, 2004.

[6] L. K. Au, B. Jordan, C. Xu, M. Batalin, A. A. Bui and B. D. M.D., "UCFit: Wireless-Enabled Cycle Restorator," University of California, Los Angeles, 2010.

[7] L. K. Au, M. A. Batalin, B. L. Jordan, C. Xu, A. A. T. Bui, B. Dobkin and W. J. Kaiser, "Demonstration of WHI-FIT: a wireless-enabled cycle restorator," in *Wireless Health*, San Diego, 2010.

[8] Tekscan, "Flexiforce Sensors," [Online]. Available: http://www.tekscan.com.

[9] L. Au, W. Wu, M. Batalin, D. Mclntire and W. Kaiser, "MicroLEAP: Energy-aware Wireless Sensor Platform for Biomedical Sensing Applications," in *Biomedical Circuits and Systems Conference*, Montreal, 2007.

[10] ProForm, "290 PSX Bike," [Online]. Available: http://www.proform.com/webapp/wcs/stores/servlet/Product_-1_14201_16002_29511_139603. [Accessed 5 June 2012].

[11] Sensorland, "The Strain Gauge," [Online]. Available: http://www.sensorland.com/HowPage002.html. [Accessed 5 June 2012].

[12] Arduino, "Arduino Pro Mini," [Online]. Available: http://arduino.cc/it/Main/ArduinoBoardProMini. [Accessed 5 June 2012].

[13] Honeywell, "SS49E Series Economical Linear Position Sensors," 2004. [Online]. Available:

http://sccatalog.honeywell.com/imc/printfriendly.asp?FAM=solidstate&PN=SS49E. [Accessed 5 June 2012].

[14] Honeywell, "HOA0971-N51," [Online]. Available: http://sensing.honeywell.com/product%20page?pr_id=39446. [Accessed 5 June 2012].

[15] Wikipedia, "List of moments of inertia," [Online]. Available: http://en.wikipedia.org/wiki/List_of_moments_of_inertia. [Accessed 5 June 2012].

[16] Arduino, [Online]. Available: http://arduino.cc/. [Accessed 2012].

[17] Roving Networks, "RN-42," [Online]. Available: http://www.rovingnetworks.com/products/RN_42. [Accessed 5 June 2012].