BME354 Arduino Project
The "FitBrick" Pedometer

Duke University
Biomedical Engineering

November 11, 2015

1 Concepts

- Wearable Technology
- Arduino
- Accelerometers

2 Introduction

Wearable technology receives more and more attention. These devices include wristwear, glassses, skin patches and much more. Recently a trend in fitness and health tracking has created an explosion in this market that is already valued at roughly $20 billion. Devices such as the FitBit, Pebble and Nike's Fuel Band allow users to track their exercise over time as well as sleep quality and a variety of other obscure features, and interface with smartphones to display and monitor progress longitudinally. For this project you will be constructing a pedometer that will interactively track activity.

2.1 Accelerometers

At the core of every technology that tracks motion, from pedometers to smart phones, is an accelerometer. Conceptually, an accelerometer can be thought of as a damped mass on a spring. When the system is subject to an acceleration, a force equal to the product of mass and acceleration ($F = ma$) will act on the mass, causing it to deflect and stretch the spring. Of course, gravitational acceleration is always acting on the object at rest. You will notice this when reading the acceleration from the system you build. A three axis accelerometer has 3 orthogonal accelerometer systems, one for each direction (i.e. in xyz Cartesian space).

2.2 Digital Accelerometers

Accelerometers on chips typically detect displacements with either capacitive or piezoelectric displacement sensors. The specific accelerometer in this project is a microelectromechanical system (MEMS) that relies on a differential capacitor with a capactive bridge circuit

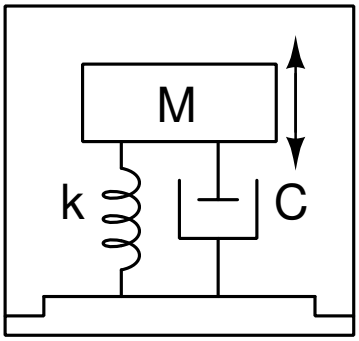

Figure 1: Accelerometer modeled as a spring mass damper system.

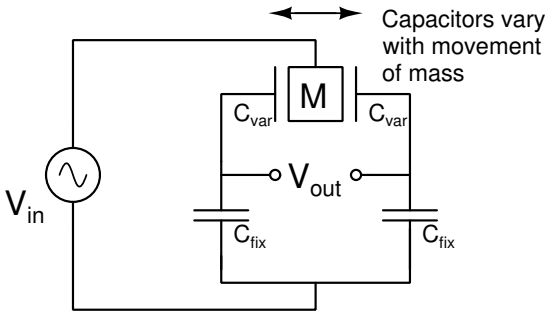

Figure 2: Capacitance based accelerometer

to detect displacement (Fig. 2). A nice description of how these devices work is available here:

https://www.youtube.com/watch?v=KZVgKu6v808

From this system the acceleration can be calculated as follows. In general, the capacitance, C, of a capacitor is given by

$$C = \epsilon \frac{A}{x_o} \tag{1}$$

Where

- C is the capacitance in Farads
- x_o is the separation of the plates
- A is the area of the plates
- ϵ is the dielectric constant of the material between the plates

For our differential MEMS system shown in Figure 2, we have a differential capacitor in a capacitive Wheatstone Bridge detection circuit:

$$C_0 = \epsilon\frac{A}{x_o}; C_1 = \epsilon\frac{A}{x_o + x} = \epsilon\frac{A}{x_o(1 + \frac{x}{x_o})} = \frac{C_0}{1 + \delta} \quad (2)$$

$$C_2 = \epsilon\frac{A}{x_o - x} = \epsilon\frac{A}{x_o(1 - \frac{x}{x_o})} = \frac{C_0}{1 - \delta} \quad (3)$$

$$\delta = \frac{x}{x_o} \quad (4)$$

$$if \quad \delta \ll 1 \quad v_{out} = v_{in} * \delta \quad (5)$$

Assuming we know the spring constant, k, and the mass, m, we can compute the acceleration from our expressions for force on a spring:

$$F = kx = ma; a = \frac{k}{m}x = \frac{k}{m}x_o\delta = \frac{k}{m}x_o * \frac{v_{out}}{v_{in}} \quad (6)$$

- a = acceleration
- A = the area of the plates
- x_o = initial distance of plates
- x = distance center plate moved
- k = spring constant of the system
- m = mass of the proof mass

2.3 Volatile and Non-Volatile Memory

You are probably most familiar with the volatile random access memory (RAM) on the Arduino. All of the variables in your code are stored in RAM. This volatile RAM memory requires power to maintain the stored information. An interruption in the power supply will cause the data stored in your variables to be lost.

Non-volatile memory will retain information written to it even if the power supply is interrupted. Examples of non-volatile memory include EEPROM (of which the Arduino has a small amount), flash memory (such as your SD card) or spinning disk drives. Non-volatile memory is almost always slower to read and write than volatile memory, which in part explains why we have volatile memory at all.

You will need to use non-volatile memory in this project. The battery might become dislodged or simply run out of power, but we wouldn't want the user to lose all their fitness information.

The Arduino's processor is the ATmega328. Built in the ATmega328 is 2 KB (2048 bytes) of volatile RAM memory, 32 KB of nonvolatile flash memory, and 1 KB of nonvolatile EEPROM memory.

The ATmega's RAM is used at run time to store all the variables you declare. It also stores some executable code while it is running.

The ATmega nonvolatile flash memory is used for storing executable code. Flash memory is where your code goes when you upload a sketch from the Arduino Sketchbook. It is possible to programmatically access flash memory for arbitrary purposes at runtime, but you are strongly discouraged from doing so. It would be easy to trample your code, or even the Arduino bootloader, both of which would have disastrous results.

The ATmega's nonvolatile EEPROM can be used to store a limited amount of information that you do not want to lose should the batteries die. Examples of what you might store in EEPROM include lifetime number of footfalls and calibration data.

In order to use EEPROM, include `<EEPROM.h>` with the rest of your includes. You need to know which byte location you wish to write to, which means you'll have to do some bookkeeping to know which memory locatins are used in EEPROM and which are not. For more examples on using EEPROM, see:

https://www.arduino.cc/en/Tutorial/EEPROMWrite
http://forum.arduino.cc/index.php?topic=111366.0

Before you start using EEPROM, you should clear it. This is a one time event. We suggest you write a small sketch to clear the EEPROM, run that sketch, and then load and run your FitBrick sketch. It should be obvious that you do not want to clear EEPROM in the `setup()` of your FitBrick code.

EEPROM is optional in this project. It needn't be used if you'd prefer to do everything on the SD card.

2.4 I²C and SPI Serial Communication

In this project, you will need to use both I²C and SPI communication protocols. Each of these protocols is a way for embedded controllers, such as the Arduino, to communicate with other Integrated Circuits (ICs) in the system.

I²C stands for "Inter-Integrated Circuit" and is pronounced "I-squared-C." I²C a standard originally defined by Intel in 1995. You can read all you want to read on the subject at Wikipedia[1]. You'll need to talk I²C to the LCD screen and the accelerometer. Fortunately for us, you don't need to know much other than which Arduino libraries to use.

The LCD Shield and the accelerometer both use I²C. Both require you to include `<Wire.h>`

The LCD Shield also requires

```
<Adafruit_MCP23017.h>
<Adafruit_RGBLCDSHield.h>
```

The accelerometer also requires

```
<Adafruit_MMA8451.h>
<Adafruit_Sensor.h>
```

You probably already have the Adafruit LCD Shield libraries, but if you do not, they can be found at

[1] https://en.wikipedia.org/wiki/I2C

https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library/archive/master.zip

You will need to download the Adafruit MMA8451 accelerometer library, and the Adafruit Sensor Library from

https://github.com/adafruit/Adafruit_MMA8451_Library/archive/master.zip

https://github.com/adafruit/Adafruit_Sensor/archive/master.zip

The MMA8451 datasheet is at

www.adafruit.com/datasheets/Freescale_MMA8451QR1.pdf

A tutorial on using the accelerometer is at

https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/wiring-and-test

SPI stands for "Serial Peripheral Interface[2]." It is another way for the embedded controller to talk to other ICs. You will use SPI to access your SD Flash card. Again, all you need to know is which libraries to use.

The SD card reader requires `<SD.h>`.

The SD library itself includes the SPI library so you needn't include it explicitly. The SD library and SPI library are included in the default Arduino sketchbook distribution so they do not require a separate download. Find SD documentation at

https://www.arduino.cc/en/Reference/SD

2.5 Battery Power

Of course, wearable technology would not be pleasant to wear without batteries. Though it is a power hog relative to other wearable technology, your device will run on four (4) AA batteries. We estimate your device will take around 30 mA to keep it running. The Energizer Company says their battery has a capacity of around 2500 mA h at that discharge rate[3].

Technology built specifically for battery power has a lot of engineering devoted to conserving battery life. Your Arduino Nano and LCD Shield, in particular, have no such optimisation built in. With a little clever programming, you could work to extend the battery life of your device a little bit beyond what it would otherwise last.

[2] https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
[3] http://data.energizer.com/PDFs/E91.pdf

Figure 3: Provided FitBrick enclosure.

3 Project Requirements

3.1 Basic Functionality

You will work with your lab partner on this project. Divide your labor intelligently.

Your FitBrick project will comprise your Arduino Nano, SD card reader, LCD shield, and four (4) AA batteries along with a Freescale MMA8451Q 3-Axis accelerometer, which we will provide. It will reside in an enclosure we provide, shown in Figure 3. There will be a small amount of soldering to create interconnections between the Arduino and peripherals. All items in the enclosure will be secured; items floating around inside the box is not acceptable.

We will not provide AA batteries. You must provide your own AA batteries.

Baseline software functionality (i.e. a grade of 90%) should include:

- Monitor the accelerometer and determine when footfalls occur, regardless of the orientation of your device.
- LCD display shows total lifetime footfalls (like a car's odometer). This value must survive a shutdown.
- LCD display shows a resettable number of footfalls (like a car's trip odometer). This need not survive a shutdown.
- LCD display shows current level of activity, "low," "moderate," or "intense."
- Every time the level of activity changes, record the `millis()` time to a file on the SD card along with the new level of activity. The file format should be readable in Excel or MATLAB for later analysis.
- LCD can be put into a mode that shows moment-to-moment acceleration on all three axes, updated at least once per second.
- Code is formatted and commented appropriately.

Make your LCD display visually appealing, easy to read, and free of flicker. Be creative.

Be prepared to explain the operation of your code in detail.

3.2 Extras that will improve your grade:

Baseline functionality is required. Once you achieve basic functionality and you wish to go beyond that, you could attempt any of:

- Estimate distance travelled in a given, resettable period
- Measure vertical motion (floors climbed)
- Apnea detection
- Stopwatch that automatically starts and stops with vigorous activity
- Calibrate the step acceleration threshold for each user instead of setting a default acceleration per step (in case certain users have more bounce in their step)
- Demonstrably extend the battery life of your device
- Allow the user to calibrate the step acceleration threshold for each user instead of setting a default acceleration
- Any interesting functionality of your own creation. Whatever you implement, be prepared to explain the operation of your code in detail.

Or interesting functionality of your own creation. Whatever you implement, be prepared to explain the operation of your code in detail.

3.3 Programming Notes

This project will use the accelerometer to sense footfalls. Think carefully about how often you need to be sampling acceleration data to adequately count all footfalls.

Remember that writing to the Adafruit LCD Shield is expensive. Writing to the SD card is also time consuming. Excessive time spent writing to either device could cause your device to miss counting a step, for example. Be careful and write to the LCD shield and SD card only when necessary.

Use modular coding techniques as much as possible to help with debugging. This means that instead of writing one long block of "everything" code, you should break your code out into functions, which are called in your main loop. For example, if you were making a heart rate monitor, your loop might contain something like this:

```
void loop()
{
  read_signal();
  detect_beat();
  update_display();
}
```

4 Final Project Deliverables

In lab, Thursday 12/3/2015:

- Paper copy of the code running in your Arduino at the time of the presentation.
- A short, 5 slide maximum, PowerPoint presentation on your computer (there will be no projector) during your device demonstration that highlights the hardware and software design. Include a diagram of the Arduino and its attached peripherals, showing pin numbers and signal labels (e.g. "MOSI," "SCLK," and "MISO" on the SPI bus). Also include a pseudocode or flowchart of your Arduino code.
- 15 minute oral presentation about your device.
- Your SD card will have at least 3 hours of data, available for inspection.
- Your functional device available for inspection and demonstration. Have the enclosure open.
- A brief (no more than one page) user guide for your device.