ECE155: Engineering Design with Embedded Systems Winter 2013

Lab 2 (Detecting Footsteps) — Week 5

Kirill Morozov *version 1*

Deadline: You must submit the lab to the SVN repository and be prepared to demonstrate Lab 2 to a TA by the start of your assigned Lab 3 session. The best way to demo is during a lab session, but any earlier time where you can convince a TA to watch is OK too.

Note: We've deliberately made Lab 2 much less directive than Lab 1. We believe that we've included all of the background information you need. But implementation decisions are up to you.

1 Objective

The goal of this lab is to interpret sensor data. You will create a pedometer application for your Android phone. Your application will read the phone's accelerometer to count the user's steps. You may choose how the user will hold the phone during the demonstration. (A phone held in the user's hand will produce very different acceleration data from a phone stored in the user's pocket, or strapped to the user's shoe.) Your goal is to make your pedometer as accurate as possible.

During this lab, you will have an opportunity to:

1. Study accelerometer readings to identify patterns.
2. Implement more sophisticated event handlers for the accelerometer sensor.
3. Filter raw sensor data to account for noise and bias.
4. Design and implement a pedometer algorithm.

2 Background

This lab builds on what you learned in Lab 1. We assume that you now know how to read the sensors on an Android phone. Most students will probably start with their Lab 1 code as a template.

Here are a few hints.

1. You may want to use the `TYPE_LINEAR_ACCELERATION` virtual sensor from Android. That sensor filters out gravity, while `TYPE_ACCELEROMETER` gives raw data. You might also consider registering your listener with `SENSOR_DELAY_FASTEST` to get more sensor readings.
2. Accelerometers are noisy. You'll need to differentiate noise from data. Start by looking at the graph of the acceleration readings you get from taking ten steps. Try to see which parts of the data correspond to steps and which parts don't belong. Then try to eliminate the noise.

3. You need to detect acceleration patterns corresponding to walking. One option is to use a simple state machine, which can work well enough for full marks. State machines also occur in ECE124; we'll review them here. More sophisticated solutions include multiple cooperating state machines. If you want more challenge after implementing state-machine-based solutions, consider a neural network for bonus marks. We're not going to say anything about those.

4. This manual will introduce you to high-pass filters, low-pass filters and finite state machines. You may find these constructs helpful, but you are free to use them or ignore them as you think best. There are many ways to correctly implement this lab. (Not all staff implementations of Lab 2 use explicit finite state machines. We strongly recommend thinking about the problem in terms of an FSM, but you don't actually need to implement a full FSM.)

5. Tired of losing your step counts on phone rotation? Implement `onSaveInstanceState()`. (Not required.)

3 High and low pass filters

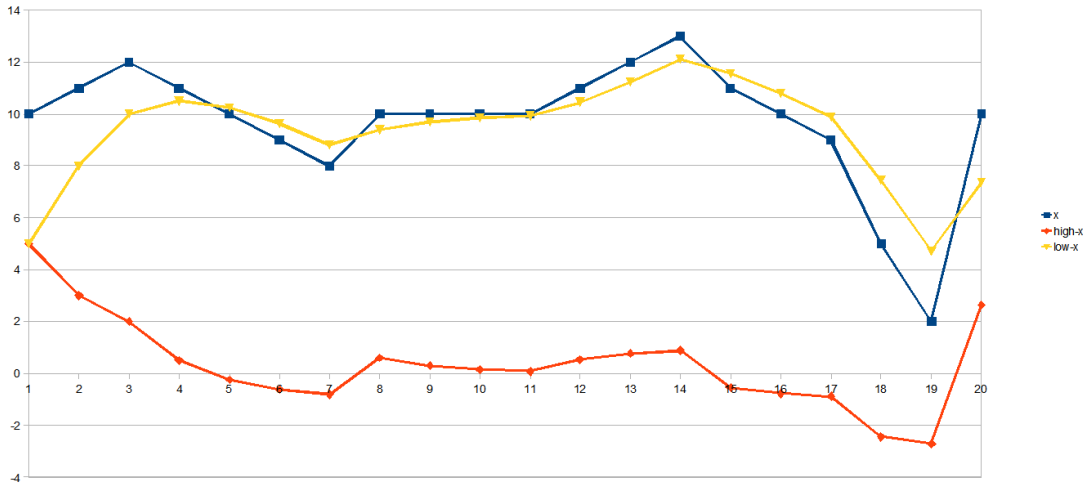

Figure 1: High-pass and low-pass filters applied to input data.

We'll introduce two mathematical constructs that you may find useful. These filters come from electronics and signal processing. A high pass filter works by attenuating low frequency signals, while a low pass filter attenuates high frequency signals.

Consider an AM radio signal[1]. Applying a high-pass filter to the signal will leave us with the carrier wave while applying a low-pass filter will eliminate the carrier wave and leave us with the data. Figure 1 shows what happens when you apply a high-pass and a low-pass filter to some input data. In this case, the data series 'x' is a sum of the function $f(x) = 10$ and some random noise. After a few samples, the data series 'high-x' closely follows the noise, while the data series 'low-x' more closely approximates the constant value 10.

[1] `http://en.wikipedia.org/wiki/Amplitude_modulation`

3.1 High-pass filter algorithm

```
// Credit: http://en.wikipedia.org/wiki/Highpass_filter
float[] highpass(float[] in, float dt, float RC) {
   float[] out = new float[in.length];
   float α = RC / (dt + RC);
   out[0] = 0;
   for(int i = 1; i < in.length; i++) {
      out[i] = α * out[i-1] + α * (in[i] - in[i-1]);
   }
   return out;
}
```

In this implementation, α is to be derived from the duration between samples and a time constant. The time constant determines which frequencies are attenuated and which ones are left unchanged. Its name comes from the product of the resistance (R) and capacitance (C) used in the equivalent electric circuit implementing this high-pass filter.

A large α means that the filtered data series will decay slowly and be more sensitive to changes in the input. From a physical point of view, the larger the α, the lower the frequency of signals that are attenuated. Conversely, the smaller the α, the higher the frequency of attenuated signals.

Since the `dt` and `RC` values are constant, you don't need to calculate them; you can simply choose an α value you find appropriate.

3.2 Low-pass filter algorithm

```
// Credit: http://en.wikipedia.org/wiki/Low-pass_filter
float[] lowpass(float[] in, float dt, float RC) {
   float[] out = new float[in.length];
   float α = dt / (dt + RC);
   out[0] = 0;
   for(int i = 1; i < in.length; i++) {
      out[i] = α * in[i] + (1-α) * out[i-1];
   }
   return out;
}
```

The high-pass and low-pass filter algorithms are quite similar. For the low-pass filter, large α means that the output will be less sensitive to changes, so you would need a more sustained spike in the input to create an equivalent change in the output. The larger the α, the lower the frequency of attenuated signals; conversely, the lower the α, the higher the frequency of attenuated signals.

In fact, you can simply use the following simplified code as your low-pass filter:

```
   smoothedAccel += (newValue - smoothedAccel) / C;
```

This slowly moves `smoothedAccel` towards `newValue`, with `C` determining the attenuation.

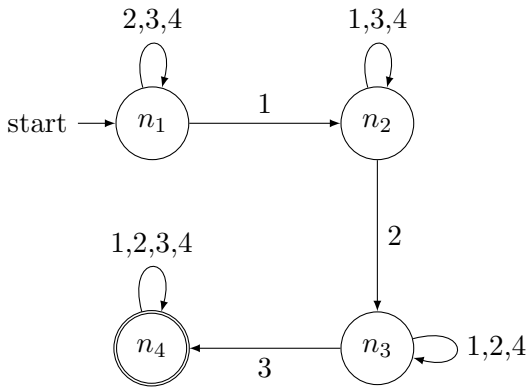

Figure 2: A Finite State Machine.

4 Finite State Machines

During this lab, you will need to identify patterns in the input. Part of the ECE124 syllabus includes (finite) state machines; it turns out that they are helpful in this course as well. Recall that a FSM has a current state, out of a finite number of states, and changes state in response to inputs. In ECE155, you'll find Finite State Machines to be useful for finding a sequence that satisfies some rule, in a stream of data. (Technically, you actually want a transducer—you want to execute actions, like increasing the step count, when you take certain transitions.)

For example, say you have a data source that generates integers in the range of 1 to 4. You want to detect if you ever get a 3 that is preceded by a 1 and a 2 in order, but you don't care if there are other numbers in between. So you want the sequence "11144323" to match, but not "2244313". To detect this, you might use an FSM that looks like Figure 2. Finite State Machines are usually represented as flow-charts.

This is how it works: The state machine starts in the circle pointed to by the "start" arrow. It processes the input, number by number. If the input matches the label of an arrow coming out of the current state, the machine takes the transition indicated by the arrow. After all the input is consumed, the state machine accepts the input if it finishes in a state with a double border (an accepting state); otherwise, it rejects the input.

You'll need to figure out what causes state transitions in your input. You might consider something like "filtered acceleration exceeds X".

5 Phases of the lab

This lab contains the following steps:

1. Gather acceleration data for different types of walking. You can look at how phone placement affects the accelerometer data. You could also look at how acceleration data for a running gait differs from a walking gait. (We'll test fast-walk versus slow-walk.)
2. Write an algorithm to eliminate noise from your input.
3. Write an algorithm to detect the gaits you have identified.

6 Deliverables

For this lab, your application must display the number of steps the user has taken from a starting position. (See Figure 3 for an example; you only need to include the number of steps, though.) You must also have a way to reset the count of steps. For full marks, your counter should be accurate to within 10% of the actual number of steps taken. You may, within reason, ask that the TA evaluating your application carry the phone in a particular way.

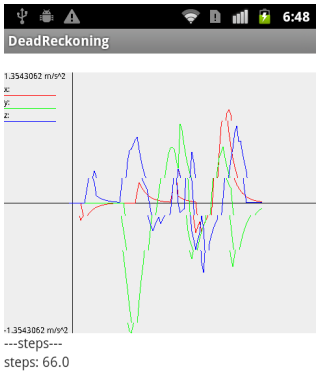

Figure 3: A complete lab 2 implementation.

7 Important Reminders

1. Remember to name your project **Lab2_SSS_XX** where SSS is your lab section (e.g. 201) and XX is your team number (e.g. 02).
2. Different hardware has different levels of noise and bias. Your application should be capable of calibrating itself to new sensors, but this is something to keep in mind.

8 Demonstration

Once you are satisfied that your design and implementation are working correctly, arrange a demonstration with a teaching assistant. You will need to have the TA complete an assessment form. All of the members of your laboratory group must sign the completed assessment form. Your grade will be based on the results of the project demonstration at the end of the laboratory session and an evaluation of how well your code follows engineering design principles. (We will run plagiarism detection software on the source code, which may subsequently lower your grade and trigger a Policy 71 case. Don't plagiarize!) You may bring your own device for the demonstration, or you may use load your application onto the device that the teaching team provides.

Requirements We will measure your pedometer on 5 walks, each between 20 and 30 steps. We'll throw out the worst result. The first four walks will each be at different speeds. The last walk will be at the same speed as the trial in which you had the greatest error. This procedure is intended to protect you from random bugs and user error, but to expose systemic flaws in your implementation.

Bonus There is a bonus mark available for this lab. During testing, you will notice that your pedometer responds to step-like events that are not actually steps. Examples include standing up, rotating the phone, or shaking the phone. If you can eliminate or significantly reduce one of these kinds of false positives, you will earn a bonus mark on this lab:

- Shaking the phone should not cause steps.
- Rotating the phone or turning around in place shouldn't cause steps (for instance, turning around in a swivelling chair).
- Moving your hand up and down should not trigger a step.
- Standing up should not trigger a step.

The evaluation of the bonus aspect of the lab will be more subjective than the core lab. You will need to explain how you are accounting for these false positives, and your solution should be noticeably less prone to these false positives than a solution that does not implement your improvements.

If you want to go for the bonus marks, start by writing an implementation which satisfies the requirements of the lab but is prone to false positives. Then, improve your solution. To help convince your teaching assistant that you have made significant improvements in accuracy, you should keep your original implementation and display the number of steps it is reporting for comparison. You might want to display both numbers at the same time.

9 Submission of project files

Upload the version of your code used in the demo to your subversion repository and commit it. The address is: `https://ecesvn.uwaterloo.ca/courses/ece155/w13/groups/group-NNN-MM`

Email Patrick Lam if you can't commit to that address.