CSE 102 – Assignment 8

Introduction

This assignment is the third and final in the series of assignments that will close out the semester on a connected project. What we will be doing is progressively building a health monitoring device (think FitBit or the like, although ours won't look as cool because we aren't going to work so hard on the packaging!).

You may work in pairs of up to two individuals on this assignment.

Pulse Monitoring

The first task is to provide for a capability to monitor heart rate in real time. Pulse oximetry techniques use the light absorptive characteristics of hemoglobin (blood) and the pulsating nature of blood flow in the arteries to determine the oxygen level (or oxygen saturation) in the blood. We won't be going all the way to oxygen saturation levels, but we will use pulse oxygenation probes to measure pulse, or heart rate. Our probe looks like the picture below.

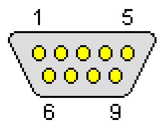
The probe houses a light source (both a red LED and an infrared LED, we will only use the red LED) and a light detector (photodiode). The probe is placed on the finger so there is a light-transmitting path from the source to the detector through a finger. As the blood flow pulsates in the arteries, the amount of light detected varies, and the signal from the light detector is used to measure pulse.

More information about the general technique of pulse oximetry is available here.

Interfacing to the Probe

We have two tasks that must be completed to interface to the probe. First, we must ensure that the red LED is on continuously. Second, we must bias the phototransistor detector (it is really a photodiode, but why complain about manufacturer's documentation? ☺) and read the analog value off the detector. The DB9 connector has the following pin-out.

connector end Pin	Pin Name	Description
9	phototransistor cathode	green wire
5	phototransistor anode	white wire; phototransistor detects level of IR and/or red light transmitted through the finger
7	shield	cable shield, connects to copper shield over the phototransistor
2	LED1	red wire; anode of the IR LED, cathode of the red LED
3	LED2	black wire; cathode of the IR LED, anode of the red LED



Connect the anode of the red LED (pin 3) to a 100Ω resistor, and connect the other side of the resistor to power (+5 V). Connect the cathode of the red LED ground. This will light the red LED and reverse-bias the infrared LED, so it will stay off.

Next connect the anode of the phototransistor (pin 5) to ground, and connect the cathode of the phototransistor (pin 9) to an analog input pin on the Arduino. In addition, connect a 1MΩ resistor from the phototransistor cathode to power (+5 V). This reverse-biases the detector (which is what is needed to read the light signal) and provides the information to the Arduino analog input.

Reading the Raw Signal

Wrap the sensor around your finger with the LED on the fingernail side. Tape the sides shut so that it is a snug (but not tight) fit. Using appropriate techniques developed earlier in the semester, read the analog input at a sample rate of 500 Hz. Note that you might want to use micros() instead of millis() as your method of accessing the Arduino's free-running timer.

Butterworth Bandpass Filter

If you look at a plot of the raw data as a function of time, you will see that it is quite noisy. We must filter the raw data to enable us to discern the heart rate signal that is buried in all that noise. For this assignment, we will use a bandpass filter (it "passes" frequencies within a specified "band") in the family of filter designs originally described by British engineer and physicist Steven Butterworth in 1930.

Computationally, the filter has the following properties. The inputs are a stream of values x_i and the filter outputs are a stream of values y_i where i is continually increasing. Given both current and previous input values as well as previous output values, the current output value is computed with the following difference equation.

$$y_i = b_0 x_i + b_1 x_{i-1} + b_2 x_{i-2} - a_1 y_{i-1} - a_2 y_{i-2}$$

where the values of a_i and b_i are constant coefficients. Author Java code to execute this filter, applied to raw data samples delivered from the Aruduino platform. You may perform this computation entirely using floating point numbers (double in Java).

Appropriate values for a_i and b_i for this filter are listed below:

a(1) = -1.9789467705, a(2) = 0.97927235

b(0) = 0.01036382, b(1) = 0, b(2) = -0.01036382

Peak Detector

To determine the heart rate, we must figure out what is the period of the filtered raw signals. This can then be converted to a rate. We will measure the period by assessing the time between successive peaks. There are many approaches to peak detection, but the simplest have the basic form

Sample i is a peak if (y(i) > y(i-1)) && (y(i) > y(i+1))

This is frequently augmented by adding a test to ensure that the values are above the mean (i.e., in our case, greater than 0).

Author Java code to detect the peaks in the filtered signal, and convert these peaks into heart rate. Display the result on the PC.

Pedometer (Step Counter)

Our health monitor will also function as a pedometer, counting steps. We will accomplish this through the use of an accelerometer sensor element. The interfacing of the accelerometer to the Arduino and detection of steps is the focus of this assignment.

We will be using the MMA8451Q Accelerometer from Freescale. The data sheet for the accelerometer can be found here. If you check out the block diagram (Figure 1, p. 3) on the data sheet, you will notice that the part is actually noticeably more complex than our Arduino processor! It has a built-in processor of its own (that perform the "embedded DSP functions" on the block diagram), in addition to three transducers (oriented along each axis), analog-to-digital conversion, and various support functions.

This document does a great job of describing how to hook up the accelerometer (we've soldered the connectors onto the board so you don't have to) and install the appropriate software drivers. It also provides a nice set of example software and use case.

For your pedometer, you can either (1) adapt to the current orientation of the sensor (i.e., sense which direction is down and look for steps in that direction), or (2) require the user to orient the sensor in a particular direction. In either case, you may assume that the orientation does not substantially change during the period of time over which you are counting steps.

There are sufficient options for built-in filtering that you should be able to count steps with a peak detection algorithm similar to that you are using for pulse measurement. It should be a tad easier, as you don't need to measure the time between steps.

If you wish, you may include a "clear" option, which tells the system to reset the step count to 0. This is not required.

Communication Protocols

The communication protocol to be used is the protocol we designed in class.

For all messages:

- First byte is the magic number 0x40 (an ASCII '@')
- Second byte is the command:
  - 0x01 is for a data value
  - 0x02 is for a message string
- Third and subsequent bytes depend upon the command:
  - Command 0x01
    - Third byte is a type (described below)
    - Fourth and Fifth bytes are the data value (high byte first)
  - Command 0x02
    - Third and Fourth bytes are the number of characters in the message string
    - Fifth and subsequent bytes are the actual characters (constrained to ASCII)

For messages going from the Arduino to the PC:

- Data value commands (0x01) support the following types (in the third byte of the message):
    - 0x01 is raw data value from the pulse-oximeter
    - 0x02 is raw accelerometer data in x dimension
    - 0x03 is raw accelerometer data in y dimension
    - 0x04 is raw accelerometer data in z dimension
- Message strings should be displayed on the console window of the PC (maybe with added identifying information indicating that it is a message string from the Arduino)

For messages going from the PC to the Arduino:

- Data value commands (0x01) support the following types (in the third byte of the message):
    - 0x05 is pulse (in beats per minute)
    - 0x06 is steps
- Message strings should be displayed on the LCD display

The receiving machine (on either side of the communication) should always look for the magic number to start a message, throwing away all incoming bytes until it sees the magic number.

Putting It All Together

Chose a screen format for the Arduino. (Can you call a 2x16 display a screen? Sure you can!) Display pulse (heartrate) in beats per minute and steps as a step count. If you wish, you can allow the user to add a distance per step calibration and display distance rather than step count (or in addition to step count). Make sure to reserve some space on the display for messages from the PC.

Collect raw data on the Arduino, send that data to the PC, process the raw data on the PC to turn it into displayable data, and display the results both on the PC and on the Arduino.

If you do not have a sufficiently long USB cable to walk around the room, it is fine to shake the Arduino to simulate steps.

Make sure you understand both the strengths and weaknesses of your design. What kind of errors (in the environment, in the raw data, etc.) could cause your system to be something less than ideal? What are alternative design approaches that might mitigate these errors? We'll ask about these items as you demo your working system. (No, we're not asking you to redesign it that way, just that you have contemplated improvements.)

Submitting Your Work

When you have finished your assignment and demonstrated it to the instructor or TA, make sure they record your completion.