Final Project – Networked Pedometer

Part 2: Data Logging and Transmission

1. Introduction:

In Part 1 of this project, you developed an accelerometer-based step counter. In this lab, you will add more functionality to your pedometer. You will learn how to use the Arduino’s onboard nonvolatile memory (EEPROM) to permanently store data. You will write the total number of steps taken into the memory every minute while the step counter is activated. You will also incorporate a second switch to prepare the pedometer data (steps, distance, time) to be sent serially via XBee (later).

2. Storing in nonvolatile memory (Adapted excerpt from BE 470, Lab 4; written by Prof. Dan Bogen):

So far you have interfaced the Arduino to a variety of sensors (e.g., photoresistor, accelerometer) to acquire data. You have also used the Arduino to process, analyze, and display data. Note that you have actually worked with three different methods to display data (or analyze data) — printing to the Serial Monitor/LCD, using LEDs as indicators, and using sounds as indicators. But the data you have acquired has been volatile; it has been stored in the Arduino RAM where, if it is not overwritten by subsequent data acquisition, it disappears when power is removed from the Arduino.

To save the data, you must put it in some kind of storage that is nonvolatile — that does not disappear when the device is powered down. One common type of nonvolatile memory is EEPROM – Electrically Erasable, Programmable, Read-Only Memory. This basically means that you can write to the memory (store data) and then read it back some time later, even if the power has been removed from the device; BUT although you can read the data as many times as you want, you can only write data infrequently. This is because this kind of memory is “worn out” by frequent use. EEPROM and FLASH memory (like on a USB memory stick) are similar. In fact, FLASH can be considered to be a subtype of EEPROM.

The microcontroller chip on your Arduino board, the ATmega328, has 1024 bytes of built-in EEPROM that you can use for nonvolatile storage. To write data to this EEPROM you use the write() function from the EEPROM library. Not surprisingly, to read data from the EEPROM you use the read() function. The documentation is at


The functions are used in the following manner:

```cpp
value = EEPROM.read(address);
EEPROM.write(address, value);
```
The `address` variable is an integer value starting at 0 indicating the address to write to, and the `value` variable is the value to write, from 0 to 255, represented as a byte (1 byte = 8 bits). Note that ints are represented with 16 bits on the Arduino. If you would like to write an integer value to the EEPROM, you must typecast it into a byte before writing.

\[
\text{value} = \text{(byte)} \text{value}; \\
\text{OR} \\
\text{value} = \text{byte}(\text{value});
\]

3. **Data Transmission:**

For this project, we would like to be able to wirelessly transmit three pieces of data—total number of steps, approximate distance traveled in feet, and total time active in minutes—from the pedometer to a master receiver, with which each pedometer can communicate. The master receiver must know how to parse the data coming from each pedometer, so the data needs to be transmitted in a specific format, as outlined in Figure 1.

![Transmission packet structure and description](image)

Your TAs have written the receiver code that will be able to parse data of this form; you just have to ensure the data gets sent correctly in the following order:
1. Send the character ‘R’ to let the receiver know to expect data.
2. Send the character corresponding to the number of your source address (the unique address of your XBee). This will be unique for each group, and you will be given this number in lab.
3. Send the character ‘S’ to let the receiver know that the following bytes that it receives will need to be reconstructed into an integer representing the number of steps taken.
4. Send the integer number of steps taken. Recall that this will be transmitted one digit at a time, so the number of bytes that this will require will be variable. The number will be reconstructed until the next signal byte (‘D’) is seen.
5. Send the character ‘D’ to let the receiver know that the following bytes that it receives will need to be reconstructed into an integer representing the approximate distance (in feet) walked.
6. Send the integer number of feet walked. The number will be reconstructed until the next signal byte (‘T’) is seen.
7. Send the character ‘T’ to let the receiver know that the following bytes that it receives will need to be reconstructed into an integer representing the approximate time (in minutes) that were spent walking.
8. Send the integer number of active minutes. The number will be reconstructed until the next end byte (‘X’) is seen.
9. Send the character ‘X’ to let the receiver know that the transmission is over.

Once the receiver parses all of the data, it will then send a byte back to the pedometer to let it know that it received the data successfully. The byte that it sends back will correspond to the source address (MY) of the pedometer’s XBee. We will not be incorporating the XBees until the next lab; for now, you will just print all of the data to the serial monitor.

4. **Procedure:**

1. Open your step counter Arduino sketch from Part 1.
2. Import the EEPROM library into your sketch.
   - Sketch ➔ Import Library… ➔ EEPROM
3. Modify your code such that the step count per minute is written to the EEPROM every minute. You should increment the EEPROM address after each time you write a value. That is, store the number of steps taken within the first minute \((t=0 \rightarrow t=1)\) in address 0, store the number of steps taken between the first and second minutes \((t=1 \rightarrow t=2)\) in address 1, etc. The address number should correspond to the total number of full minutes that have elapsed since the step counter button was turned on.
   - Hint: The \texttt{millis()} function returns the number of milliseconds since the Arduino board began running the current program. \[\text{http://www.arduino.cc/en/Reference/Millis}\]
   - You may want to make use of the fact that the result of an integer division rounds down to the nearest whole number. For example, \(1/2 = 0\), \(3/2 = 1\), and \(29/6 = 4\).
   - Don’t forget to reset the total number of steps after each minute so that you are only keeping track of the number of steps in each one minute interval.
4. Next, wire up one of your DIP switches to a digital pin (do not use 0 or 1 – these are used for serial communication). Initialize this pin as an input in setup(). Call it something like send_switch.

5. Modify your code so that when the step counter switch is turned off and this new switch is turned on, the number of steps stored in each address that was written to is printed to the serial monitor. Then, add all of these values together to get the total number of steps taken, and print this to the serial monitor. After the data is printed, the program should do nothing until this switch is turned off.

6. Test your device by walking around for a few minutes (time yourself) and keeping track of your total number of steps per minute, as well as the total number of steps you take before turning the step counter mode switch off. Verify that the data was stored correctly in the EEPROM by turning on the “send” switch and checking the accuracy of the data.
   - Note that if you run the step counter mode for, say, 2 minutes and 30 seconds, only the data up to 2 minutes will be stored; any steps you take after the 2-minute mark will not be captured. If you have time later, bonus points if you can come back and find a solution to this issue.
   - It is okay if your step counter is not perfectly accurate, but you should try to be within 5 or so from the actual step count per minute.

7. From the total number of steps, calculate the approximate total distance walked in feet. To do this, measure the average distance of one step in inches and use that to calculate the approximate total distance in feet (1 foot = 12 inches).

8. Calculate the estimated total time in minutes spent walking. (This isn’t much of a calculation; you already have this value!)

9. Write a function sendData(int totalSteps, int totalDist, int totalTime) which prints data to the serial monitor as specified in section 3 (see Figure 1 and associated explanation). After the last byte ‘X’ is received by the master module, it will send a byte back to confirm that the data was received successfully. Your function should look for this byte after it sends ‘X’ and make sure that it matches the source address byte (provided to each group individually by the TAs). If it does, flash the LED on pin 13 on and off, and then do nothing until the “send” switch is turned off. If the byte does not match the source address byte, turn the LED on pin 13 on and keep it on until the “send” switch is turned off.
   - Note that you can test this by inputting characters directly into the serial monitor.

MAKE SURE YOU SAVE YOUR CODE TO YOUR S-DRIVE (NOT C-DRIVE) AND/OR EMAIL IT TO YOURSELF! You will need it for the rest of the project.