Arduino Project

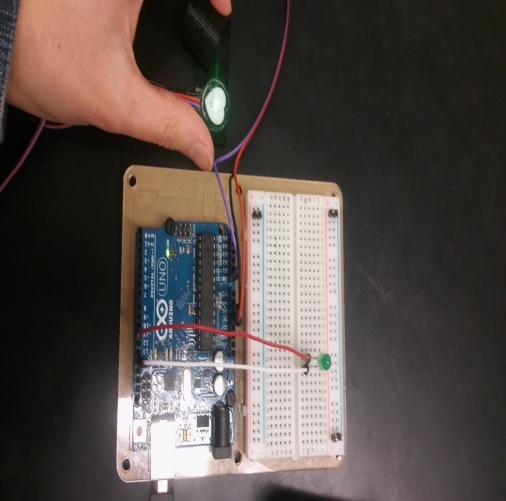
--The project given was to create a code to link with the Arduino and a pulse sensor to display a working heart rate monitor that shows the beats per minute in an acceptable manner. It is clear that MATlab was the essential tool needed to produce a working code due to its capabilities of working with the Arduino.  MATlab is the language one would likely go to when working with data, in general. The code worked as expected, and despite the pulse sensor not being the most effective possible tool for this project, valid results were produced.

a. Introduction (10 pts)

---There are plenty of reasons that heart rate monitoring needs to be automated. For example, it is important to a person’s health to know their heart rate, especially if they are not already a healthy individual. Having it automated allows it to be easier to access. It provides a more common way of accessing the ability see a person’s heart rate.  One Arduino user, Saddam, did a project very similar to this (circuitdigest.com). It was the same except that he did not use MATlab to interface with the Arduino, and he used a heartbeat module which tracked the voltage, like our pulse sensor, as well as flashed for each beat, like our LED light set up to the Arduino. Saddam’s setup featured an LCD screen hooked up to his Arduino, with the code being written using downloaded Arduino software.  The motivation for the approach using MATlab iis to find a simple way to calculate a heart rate with a simple code, but also having an accurate reading. This approach, using MATlab, is more user-friendly for someone who is just learning to code in MATlab and hasn’t had any formal experience working with an Arduino. All one has to do with this setup is learn how to set up the physical components of the Arduino and then the rest of the interfacing is done in MATlab.

b. Methods (10 pts)

---The code starts by reading a pin from the Arduino.  The Arduino tracks the voltage change by using the pulse sensor that is connected to the finger.  Two while-loops are used that the rest of the code is contained within. The first while loop shuts off when the voltage goes beyond our voltage threshold (4.5).  The second loop shuts off when voltage is no longer being registered by the Arduino (i.e the pulse sensor was taken off). The elapsed time is tracked using the tic-toc function.  A matrix was made with the first column being the elapsed time and the second being the voltage read. Variables were made to track three voltage readings from the Arduino in a row.  If the middle value is greater than the previous and next value, a peak was logged. Each time this happened, a variable “peakflag” was made, which initiated the code to make the Arduino flash the LED.  The voltage readings and elapsed time were then plotted, and the BPM was calculated and displayed.



c. Results (10 pts)

--The program worked relatively well. The code takes a little while to get working, but once it gets a firm read on the pulse, the code runs close to what is actually calculated. The data in the table below was calculated with the program first, and then with a radial pulse. The pulse found by the program was close to the radial pulse calculated by the group members. Each time the program was run, the pulse calculated was close to the actual calculated value.

|  |  |  |
| --- | --- | --- |
| **Group member** | **MATLAB BPM** | **Radial pulse BPM** |
| **1** | **57.38** | **62** |
| **2** | **65.3** | **60** |
| **3** | **68.05** | **62** |

d. Discussion (10 pts)

---Some of the shortcomings experienced were that the ‘trough’ data wouldn’t work all that well for the program written. This was solved by using a “voltage\_threshold\_min” instead of an entire “if/end” statement for the trough part of the code. Overall, the program was not affected too much considering the program still works accurately.  However, one could see where improvements could be made. In the aforementioned Saddam’s setup, the software used to code came directly from the Arduino website. It was designed to interface with the Arduino, and one might assume that more accurate readings could be obtained, or, at the very least, the Arduino more easily accessed with simpler code.  For instance, Saddam used a function “millis” which tracked the time between heartbeats directly from the Arduino. It is logical to assume that the time component would match up more consistently to the user’s actual heartbeat. The heartbeat module likely flashed much closer to the precise time of actual heartbeat, while there was a delay in the LED flash for this project.  Another shortcoming, was an aesthetic component. The group was hoping to get the graph to just display 10 second increments at a time, but could not figure out how. The pulse sensor used was not the best, and therefore had some problems of its own. If one were to press too hard on the pulse sensor, the data was compromised. Like a FitBit, since the pulse sensor uses an LED light to read the heart rate, it is not the most accurate reading. This device has the potential to go in many directions. Like the heart rate monitoring of a FitBit, this device will provide an easy way to calculate rate heart in a moment of time. This device could have the small things fixed up and then be sold to a larger company to use in devices like a FitBit, or in an even simpler application would be to sell the device as is, with a few minor fixes, to provide people with a simple and cheap way to calculate their heart rate.

e. Conclusion (5 pts)

---In this project, MATlab was used to interface with an Arduino.  An LED light was set up to the Arduino to simulate heartbeat. MATlab plotted the heartbeat against time and displayed an average heartbeat of the last 10 beats, while the light flashed for each beat.  The setup worked quite well and, within reason, accurate results were obtained. However, more accurate methods could be used for the same simulation.

**Appendix**

Acknowledgements-

Mr. Toich

Saddam: <http://circuitdigest.com/microcontroller-projects/heartbeat-monitor-project-using-arduino>

Code-

clear

clc

a=arduino(); %initializes Arduino

pulsePin='A0'; %where pulse monitor is plugged into board

b(1)=readVoltage(a,pulsePin); %reads voltage from pulse clip

counter=0; %initialize variables

elapsedtime=0;

row=1;

peak=zeros(10,1);

pulseinterval=zeros(10,1);

k=1;

j=1;

x=1;

while x~=0; %This loop terminates if the voltage goes above what is desired

while b~=0; %while loop ends when voltage is no longer registering, pulse sensor taken off

   tic; %starts timer

   peakflag=0; %initializes pulse interval timer

   data(row,1)=counter; %assigns column 1 in the matrix to be the time

   data(row,2)=b; %assigns column 2 in the matrix to be the voltage from the pulse monitor

   prevVal=b;   %the following variables are created to read voltage 3 times in a row

   b=readVoltage(a,pulsePin);

   currVal=b;

   b=readVoltage(a,pulsePin);

   nextVal=b;

   voltage\_threshold0=2.6; %mV voltage threshold, above this value are the pulse peaks

   voltage\_threshold\_min=2.0; %mV minimum voltage threshold to identify troughs

%This if-statement is made to track peaks, if the middle variable is greater than the previous                     and next variable, a peak is registered. “k” becomes 1 greater each time a peak is logged, and the peak is added to the “k” element of variable “peak”.  Variable “peakflag” is set to 1 to be used later.

   if b > voltage\_threshold0 & nextVal<currVal & currVal>prevVal;

       k=k+1;

       peak(k)=currVal;

       peakflag=1;

  end

%This if-statement is used for when the voltage read is above the desired max, x is set to 0 which ends the while-loop containing all the code.

   if b>4.5;

       x=0;

   end

%At this point in the code, all needed voltage readings have been logged, so toc is used to finish the time count

   elapsedtime=toc;

   counter=counter+elapsedtime; %keeps accumulating time

%This if-statement is used after each time a peakflag is set to 1, or when a peak was registered.  The following code turns the LED on and off for each peak, j increased by 1 and the “j” element of pulseinterval becomes the accumulated time, the if-statement ends by shutting off “peakflag”

  if peakflag==1;

       writeDigitalPin(a,'D11',1); %turns on LED if a peak is found

       writeDigitalPin(a,'D11',0); %turns off LED

       j=j+1;

       pulseinterval(j)=counter;

       peakflag=0;

   end

%At this point in the loop, all the necessary components have been obtained for plotting and displayed BPM.  “Row” becomes 1 greater, and “data” puts voltage and time into columns 1 and 2 using “row” as the row number, so for each iteration of the while loop, the next row of data is obtained.  Column 1 of “data” is plotted against column 2, or voltage vs time. The axes are properly labeled and function “hold on” is used to continually plot data for each iteration of the loop.

   row=row+1;

   pause(0.0001)

   plot(data(:,1),data(:,2),'r-')

   xlabel('time')

   ylabel('mV')

   hold on

%Once k and j are greater than or equal to 10, or when 10 or more peaks have been logged, variable “P” is created to track the last 10 peaks of each iteration of the loop.  The number of peaks is set to variable “pulsecount”, which should be 10. The total accumulated time of each of those peaks is logged as “interval”. Variable “I” is set to subtract the total elapsed time over the last 10 peaks.  Finally, “BPM” is set to pulsecount (or the number 10) divided by I/60. This sets the units to beats per minute, and the result is printed.

if k>=10 && j>=10;

P=peak((k-9):k);

pulsecount=length(P);

interval=pulseinterval((j-9):j);

I=pulseinterval(j)-pulseinterval(j-9);

BPM=pulsecount/(I/60);

fprintf('The BPM is %.2f\n',BPM)

end