IMMMotion

An accelerometer combined with a force-sensor
to measure finger tapping movement (speed, regularity)
May 2011
Floris van Vugt, IMMM Hannover

Design

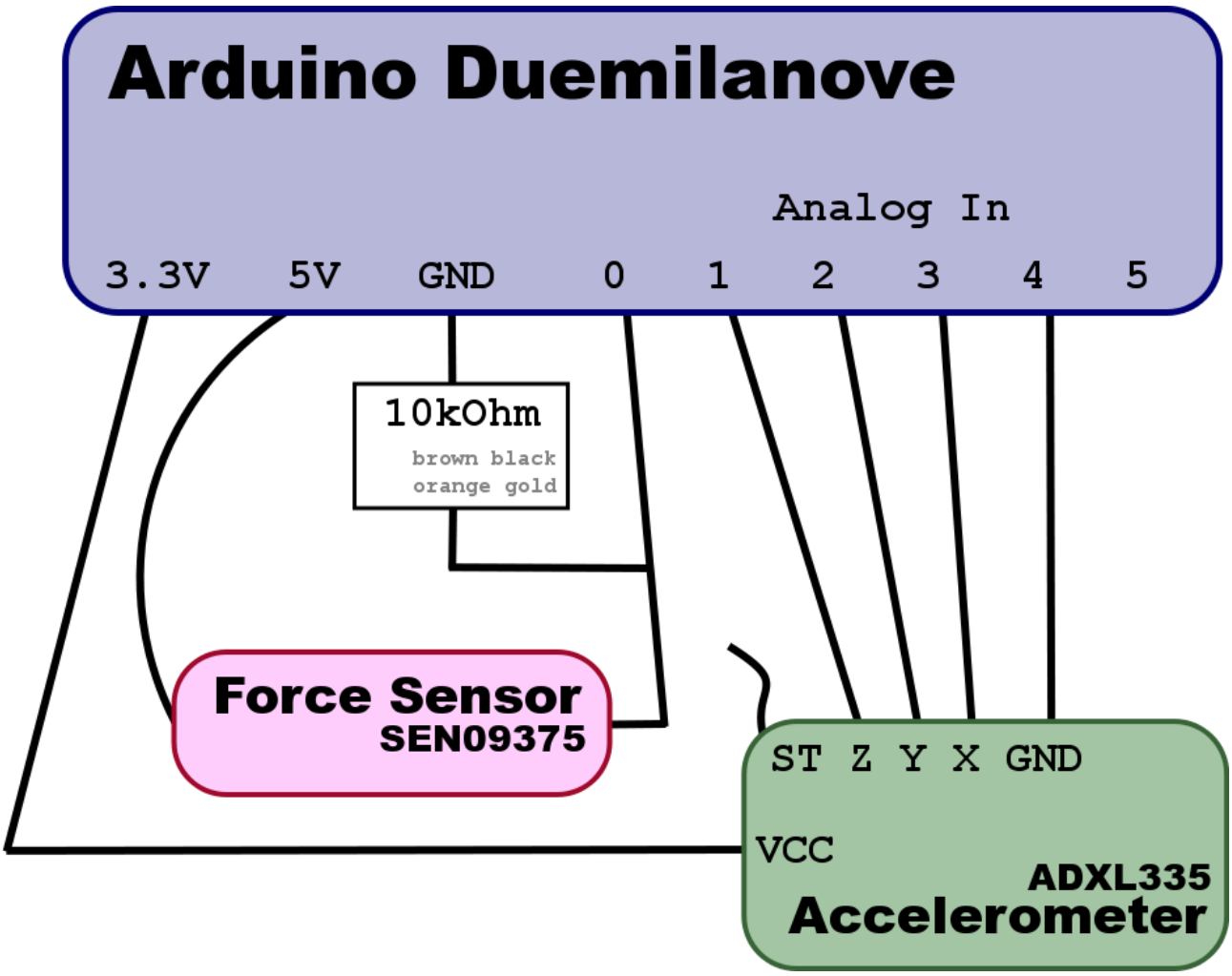
We wanted to have a device that
(1) is cheap, so that we can have multiple of them
(2) is portable, so that we can carry it to several clinics
(3) gives the frequency of a finger tapping movement (i.e. taps per second), as well as regularity and some analysable movement data.

Implementation

We use the Arduino Duemilanove experimentation board, which can be purchased online (www.arduino.cc). It reads the voltage on analog input pins, which it then communicates to a computer via USB. On the computer a program runs that listens to the USB port and logs the data. The Arduino has a force-sensitive-resistor (FSR) hooked up (which changes its resistance when a force is applied) and an accelerometer. The accelerometer is attached to the participant's finger tip and the FSR registers when the participant taps on it.
In this way we can have the tap frequency and perhaps regularity (standard deviation of the inter-onset intervals, sdIOI) based on the FSR data. We furthermore can view the acceleration profile from the accelerometer to determine the number of peaks, height of peaks and other metrics.

Electric Circuit

The arduino is connected as showin the following picture.
We use a tri-axial accelerometer (+-3g) but essentially read only the Z axis.

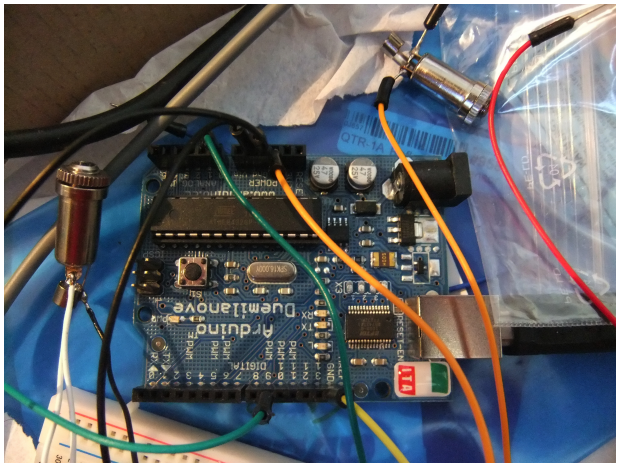


Arduino Program

The arduino is programmed using C, and the programs are transmitted onto it through USB (with a very nice Linux-based interface which you can install in Debian through **apt-get install arduino**)
The program does little more than turning analog pin 4 into a ground pin (which is required for the accelerometer) and then read the FSR data from analog pin 0 and the X,Y,Z acceleration from analog pin 3,2,1 respectively.
It then writes it to the USB port, which acts serially at a baud rate of about 10^6. The data is sent in blocks, starting with one byte “B” for beginning. Then a 2-byte time-code follows (in microseconds, which means that the clock will reset a bunch of times during a trial, but that's fine). Then two two-byte integers give the readings on the FSR (port 0) and on the Z-axis of acceleration (port 1). A python program is used to capture the data.
Reading an analog input port takes arduino about 100 microsecs, which means that when we read both ports and add the time stamp we're getting a data rate of some 3kHz.
Data is stored in a textfile and resampled and smoothed offline.

Generating the metronome signal through arduino

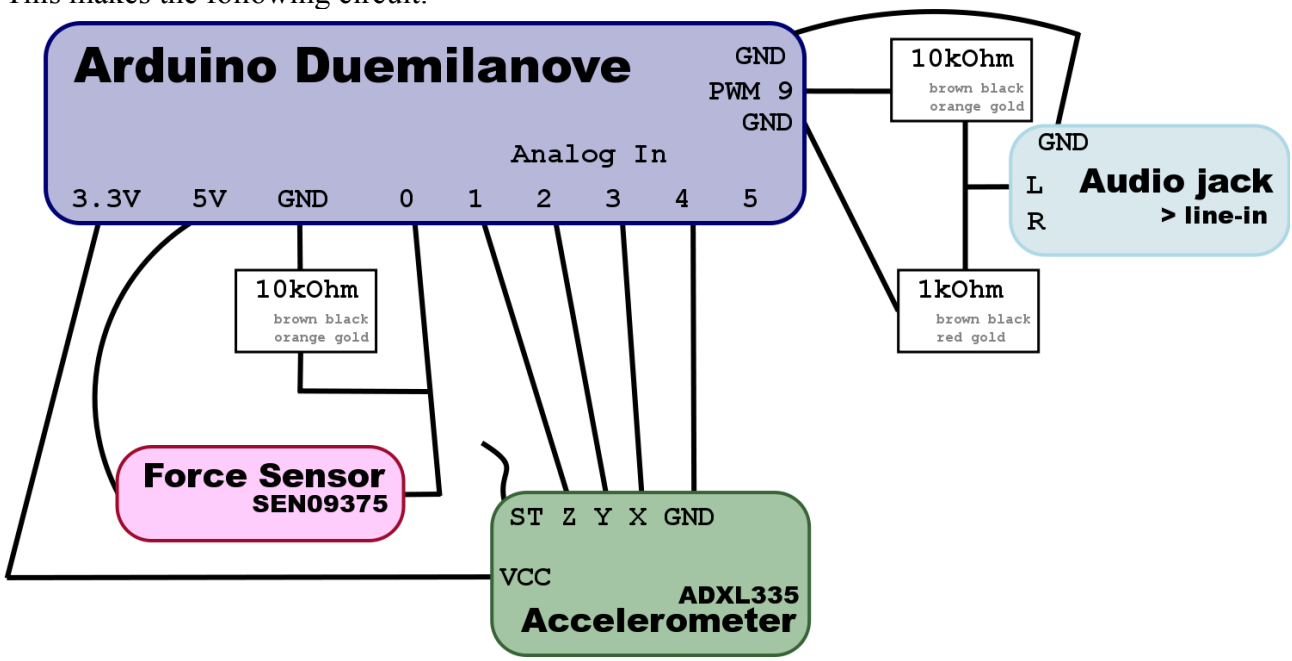
Ok, fast forward two months in time. Now I want to be able to record the metronome signal in the arduino. However, capturing the audio is a bit complicated and has all sorts of noise problems. So now my idea is to use arduino to generate a simple metronome tick at equally spaced times. Since arduino generates the tone by itself, we are sure that the timecodes that he generates along with the tapping data are synchronised.



This metronome tick is written using DDS into a PWM-enabled pin. This connects to a jack connection through a voltage divider. The jack connects to the line-in in the computer, where you can forward it to speakers. In the photo you can see how you can connect the arduino to a stereo audio jack (TSR) plug (this is not the final setting as shown below).

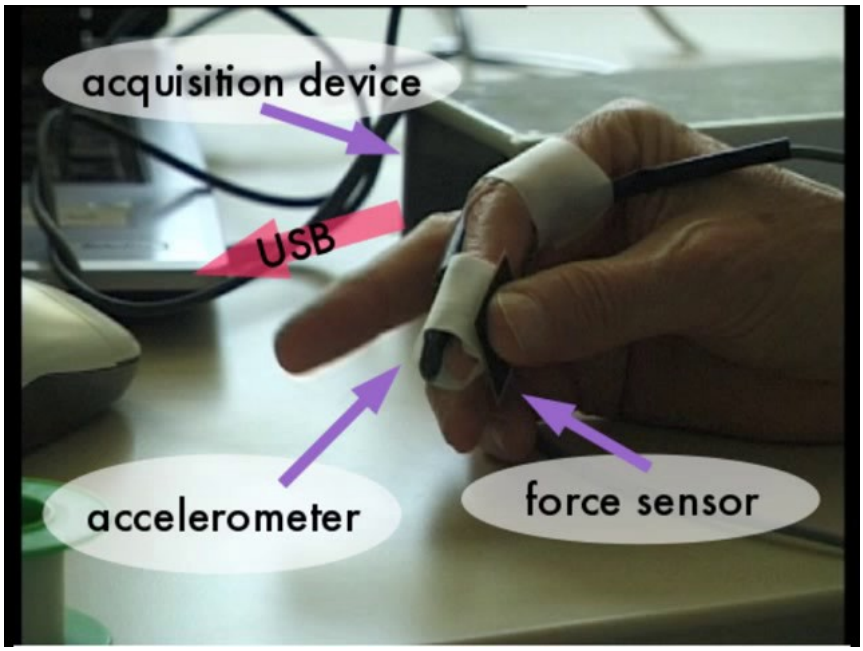
The nice thing is that this leaves no synchronisation problems, since we generate the metronome signal where we do the sampling, so it's enough to just save the metronome signal in arduino and pass it to the computer along with the accelerometer and pressure data. Probably it compromises a bit the capture frequency, but that's no problem since we are without metronome in a really high frequency range anyway (3kHz).

This makes the following circuit:



This is what it looks like in reality:

And you can see Eckart's hand as he is performing the index-to-thumb tapping.



Logbook

I had some trouble recently (March 2012) while compiling a new version of the program that I flash onto the arduino. It did not write the input values correctly to the binary serial out port. It turned out that I had used `Serial.print()` which in the newer version of arduino doesn't write a binary byte as a single bite, but rather as more (when it's an integer). The solution was to replace `Serial.print()` with `Serial.write()` and it all worked. I think this has to do with the transition from Arduino Duemilanove to Arduino Uno.

I managed to get it running on Arduino Uno (April 2012).

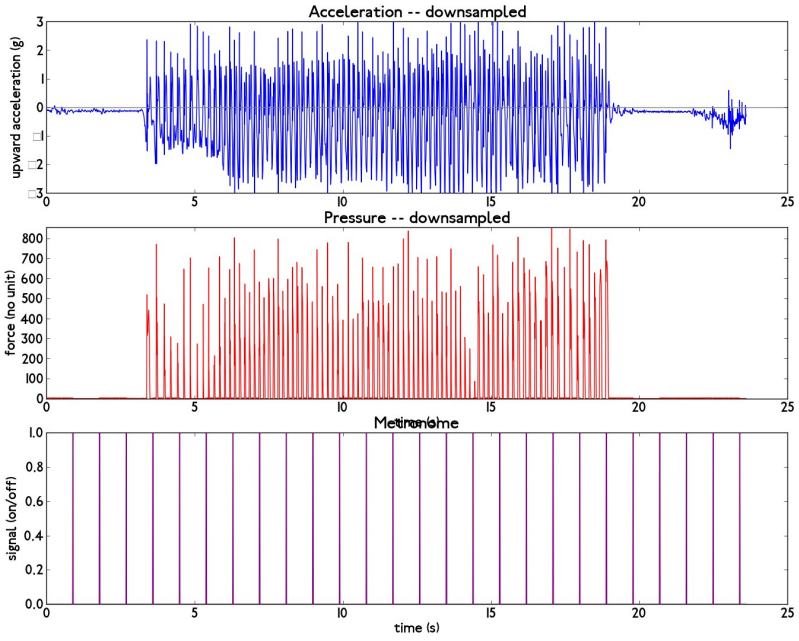
The code is in prog/arduino/sketchbooks/capture_with_metronome
The capture program is in prog/immmotion/capture-gui.py

Capture program

A custom-made python program handles the capturing of the tapping data. It listens to the USB port and reads the binary data serially. The data comes in little packages of several bytes, the first of which is 'B' and the last is 'E', to mark the beginning and the end. In between there are the data read out (one sample), for example a time-value, a reading of the force sensor, and a reading of the accelerometer. This data is saved to a text-file format (space-separated) for later, offline analysis. Note that the time is entered in microseconds (the unit of the arduino internal clock) and hence each sample has its unique associated time-code. This makes that USB-transfer delays play no role, the data comes in in its own sweet time.

The capture program is simple and intuitive, and it enables one to start and stop the capture stream easily.

The captured data can furthermore be previewed, as shown below (note that the data is downsampled to make this preview fast, the actual resolution is much better!):



About 15 seconds of captured data. The first graph shows the accelerometer trace. The second graph shows the data from the force sensor, and it shows clearly the tap points. The third graph is the metronome signal.

Here is a detail of this picture (note that this is me tapping as fast as possible while ignoring the metronome completely):