Lab #7 – analyzing an ECG on the host PC

Overview:

In this lab, we'll

- use the host PCs to examine the ECG signals that you collected last week
- explore an algorithm to parse an ECG signal and compute heart rate
- understand the difference between processing ECG signals as opposed to an sEMG.

Legal reminder:

This lab involves using your ECG. Legally, this signal represents medical data. As such, federal law says that you have the right to keep it private, and we will of course respect that law. Please refer to our separate legal sheet (which you should have read and signed).

What problem are we solving today?

We'll start with the ECG data that you collected last week – thousands of numbers. For today, we want software on the host to look at these numbers and compute your heart rate. Once we can do this, it would be easy to port the software to run on a PyBoard to be a dedicated heart-rate monitor if you would like to for a final project.

Lab setup – hardware

This lab uses only the PC host computer. It does not use the PyBoard, the scope or any other hardware.

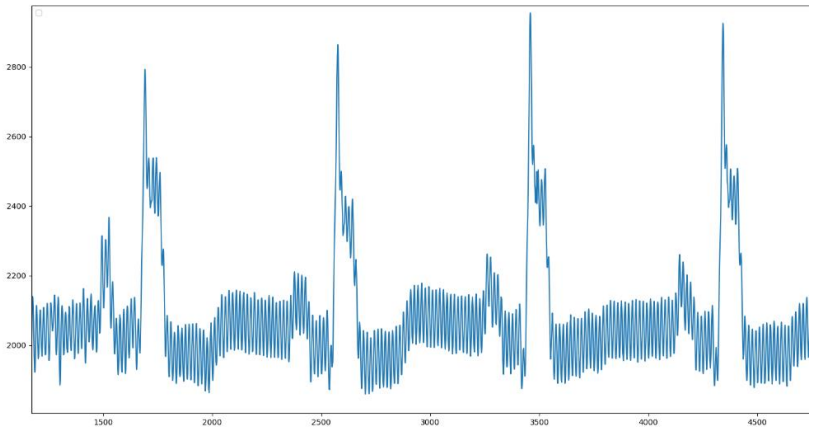
Lab setup – software

The setup for this lab involves first copying a Python program from the class shared folder to your own folder (where you can modify your copy).

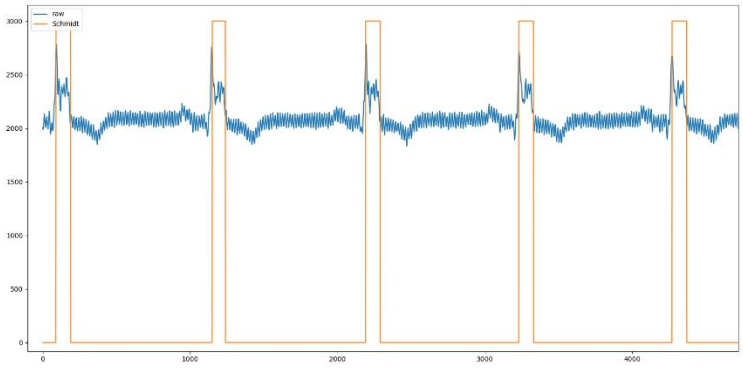
The file locations are the same as usual; the class folder is Q → en1EIY → 2022f → public_html → labs → code. Use the standard Windows copy/paste to grab the file *7_ecg_host.py* and paste it into your own Z-drive folder.

First try

Here's a digitized Lead-III ECG:



Our goal will be to use an algorithm similar to our friendly, trusted sEMG algorithm: a smoothing filter, a second baseline filter, and then the segment-by-segment RMS and Schmidt triggers. We would like to pick the parameters so that each QRS complex results in a single peak of the output, as follows:



Because of the baselining, our final output starts at zero while the initial raw input is offset to be much higher – that's OK. All that matters is that we could use our Schmidt-trigger output to, e.g., pulse an LED or to compute the time between heartbeats and thus compute the heartrate.

It looks easy in the picture above! But vaporware is always easier to write than actual software 😊. We want to get it to work for all of:

- whatever ECGs you recorded last lab from your group
- the sample in Q → en1EIY → 2022f → public_html → labs → code → ecg_8232_ll_ra_1.csv
- the sample in Q → en1EIY → 2022f → public_html → labs → code → ecg_8232_ll_ra_1.csv

You should use the same filtering (i.e., the same values of `f1`, `f2` and `f3`) for all three, though each can have its own Schmidt-trigger levels.

How well can you get that to work? You may have several issues to deal with:

- How well is the second filter doing at finding a nice baseline? Unless you see visible drifting of one heartbeat relative to the previous one, the baseline should be staying pretty constant. I.e., the "baselined" signal should look just like the original except moved to the origin.
- We originally used the RMS (root mean square) algorithm because we wanted to treat negative and positive numbers equally. Do we still want to do that?
- The QRS complex can be very skinny. Is the algorithm really catching it well?

You can definitely get something *similar* to our algorithm to work, but you may have to tweak the code a bit. Feel free to have at it with whatever ideas you can. Here are some debugging steps that may help.

Steps to get some intuition

Visual intuition is your friend, and graphical plots can help a lot. Let's go through the algorithm steps in order. So, first you might work with `f1`. Uncomment the line

```
plt.plot (plot_f1, label="f1")
```

to see the smoothing-filter output. Look at the plots; they should smooth out any high-frequency noise but leave the waveform mostly unchanged. Do they? If not, play with `f1` until they do.

Next, move to `f2`, which controls the baselining filter. The sole purpose of this is to correct any baseline drift – i.e., any slow drifting of the signal up or down. It should *not* substantially change the shape of the ECG signal. To check this out, we can uncomment the line

```
plt.plot (plot_baseline, label="baseline")
```

If the `baseline` signal looks correct, then the `baselined` signal should, again, look very much like the original signal except for being cleaner and for being more-or-less centered around zero. If it doesn't, then play with `f2` until it works as desired.

Once you have the first two filters working, the real cleverness begins! Now comes the interesting part – breaking the signal into segments and taking the RMS value in each segment. You can take a look at this by uncommenting the line

```
plt.plot (plot_rms, label="RMS")
```

It will show you the results of breaking the signal into segments and taking the RMS of each segment. For the sEMG, the purpose of the RMS algorithm was to take a signal that had rapid swings above and below zero and just turn it into its average strength. Is that working well here? If not, what might work better? Similarly, for the sEMG we had a tradeoff of picking the segment size – does our typical .1 second segment size still work well for an ECG?

With that all done, you now should have a waveform that has one big peak (i.e., the QRS complex) per heartbeat. If it has other peaks nearly as large as the QRS, then you might wind up having a hard time picking appropriate Schmidt-trigger values that correctly grab just the one peak.

Questions

1. Explain the "RMS issue" in a bit more detail. Why doesn't using the RMS value work nearly as well for us now as it did for the sEMG signals?
2. Let's say that your T wave is almost as high as your QRS complex, and your algorithm misinterprets it as a second QRS complex. How would that affect your calculated heart rate?
3. Most people find that they need a very different value of `f2` for this lab than they did for the ECG lab. Given what we learned about the frequency domain, can you explain why this is so?

What to turn in:

One lab report per group, contain the answers to the questions above. Include any pictures that might help your explanations. And if anyone has pictures of data that presented unusual or interesting problems, that would be fun to see also.