

# Real-time ECG Acquisition, Digitization, Plotting, and BPM Measurements

This tutorial walks through a working real-time ECG pipeline built around the `ecg_plot_actual.py` we used. It explains the data flow, how to get the hardware and network streaming working, the signal processing used (filters, baseline removal, peak detection), and how to tune and test the system.

**Files referenced:** `ecg_plot_actual.py` (live plotting + BPM computation)

---

## Table of contents

1. Goal & overview
  2. Requirements (software + packages)
  3. Data flow and how the program expects samples
  4. Key parameters in `ecg_plot_actual.py` and what they mean
  5. Signal processing explained (DC removal, notch, bandpass, baseline, smoothing)
  6. Peak detection & BPM algorithm (how peaks are found and BPM computed)
  7. Running the program locally (step-by-step)
  8. Tuning & troubleshooting checklist
- 

## 1. Goal & overview

The program receives single floating point sample values over UDP and displays a scrolling ECG plot in real time while computing a smoothed BPM estimate. The design separates acquisition (UDP stream), buffering (fixed window), filtering (notch + bandpass + baseline removal), peak detection (dynamic threshold + minimum RR), and BPM smoothing.

The advantage of this architecture is that the display remains responsive while the detection logic works on the recent window of data.

## 2. Requirements

Install the Python packages used by `ecg_plot_actual.py`:

```
pip install numpy scipy pyqtgraph PyQt5
```

You will also need a system (Beagle or other board) that streams newline-terminated floating point values over UDP to the machine running the program.

### 3. Data flow and how the program expects samples

- The program binds to a UDP port (default 12345) and expects textual lines that can be parsed as `float`. Example packet payload: "0.1234\n".
- On start the program attempts to `sendto(b"send\n", (BEAGLE_IP, UDP_PORT))` to the upstream device to trigger streaming — this is a convenience only and will silently fail if not applicable.
- The program keeps a circular buffer sized to `FS * WINDOW_SEC` samples and draws the last `WINDOW_SEC` seconds.

#### Recommended simple sender behaviour

- Send samples at a steady rate matching `FS` (default 2000 Hz).
- Send one sample per UDP datagram or send a small newline-separated batch — the program reads text chunks and decodes floats line by line.

### 4. Key parameters in `ecg_plot_actual.py` and what they mean

- `UDP_PORT / BEAGLE_IP`: network parameters.
- `FS`: sampling frequency (Hz). Must match sender. Default 2000.0.
- `WINDOW_SEC`: seconds shown in the scrolling plot (default 10). The buffer length `N = int(FS * WINDOW_SEC)`.
- `NOTCH_F0, NOTCH_Q`: notch filter center and quality factor (e.g., 60 Hz mains notch).
- `MIN_DISTANCE_S`: minimum allowed distance between detected peaks in seconds (~0.65 s default). This prevents double counting.
- `BPM_HISTORY_SIZE`: how many past BPM values are kept for smoothing.

**Note:** If your hardware samples at 500 Hz, set `FS = 500.0` and keep other timing parameters expressed in seconds (the code converts to samples internally).

### 5. Signal processing explained

The processing chain in the program is:

1. **DC centering** — subtract mean over the window to remove offset.
2. **Notch filter** — removes mains interference (60 Hz default). Implemented with `iirnotch` converted into SOS for stability.
3. **Bandpass filter (0.5–40 Hz)** — removes slow drift and high frequency noise while preserving QRS energy.
4. **Median baseline removal** — removes remaining baseline wander using a median filter over a long kernel (200 ms × FS in code) and subtracts it.

All filters are applied using forward-only filtering (e.g., `sosfilt`) to avoid zero-phase operations which would require bidirectional filtering and cause edge artifacts in a rolling buffer.

## 6. Peak detection & BPM algorithm

Use a conservative approach:

- A smoothing window (~50 ms) reduces high frequency noise before peak search.
- A dynamic threshold is computed from the *positive* portion of the center 80% of the window (to avoid edge bias) using a percentile (60th). This adapts to amplitude changes.
- Only positive peaks (ECG R-waves that appear positive in your setup) above the threshold and separated by at least `MIN_DISTANCE_S` are considered.
- The code converts new detections in the newest 2 seconds of the buffer to absolute times (seconds since start) and appends them to `all_peak_times`. Old peaks older than 10 s are removed.
- To compute BPM the code calculates RR intervals from `all_peak_times`, filters RR intervals to a realistic heart rate range (0.63–1.2 s = 95–50 BPM), uses the median RR and converts to BPM. The displayed BPM uses heavy smoothing (IIR + median of recent estimates) to avoid jumpiness.

**Why median RR?** Median is robust to outliers (premature beats or missed detections). The RR range filter reduces false high/low values.

## 7. Running the program locally (step-by-step)

1. Make sure the sending device (or simulator) can reach the PC running the program — check firewalls and routing.

2. Edit the top of `ecg_plot_actual.py` for `BEAGLE_IP` (if you want the program to send the initial `send` trigger) and `UDP_PORT` if needed.
3. Launch the program :  

```
python ecg_plot_actual.py
```
4. Start your ECG data sender (hardware or simulator). You should see a scrolling ECG and the BPM label update.

## 8. Tuning & troubleshooting checklist

- **No data shown:** confirm UDP port and that packets reach the PC (`tcpdump -n udp port 12345` or `wireshark`).
- **Wrong zoom / no QRS shape:** confirm `FS` matches the sampler. If FS too low the QRS will be aliased or too few samples per QRS. Try `FS = 500` and a smaller `WINDOW_SEC` if collecting at lower rates.
- **Too many false peaks:** increase `MIN_DISTANCE_S`, raise the threshold percentile (change `60` to `70`), or increase pre-smoothing window.
- **Missed peaks:** lower threshold, reduce `MIN_DISTANCE_S`, or widen bandpass to include more QRS energy (e.g., upper cutoff to 60 Hz). Also ensure sample alignment doesn't clip R-wave.
- **Strong mains hum:** confirm `NOTCH_F0` (50 or 60 Hz depending on locale) and `NOTCH_Q` are correct.
- **Baseline wander:** reduce median baseline window or implement high-pass filter with a small DC cutoff (e.g., 0.5 Hz is used now). If baseline removal over-compensates, tweak kernel size in the median filter.