Heart Sense

Every Beat Counts

Hardware2 Project

Group2 Binary

HEART SENSE

Heart rate variability (HRV) is a **powerful indicator** of stress, recovery, and general health.

Built with Raspberry Pi Pico – **keeping cost under €10** without compromising functionality.

We believe that everyone should be able to access their physiological signals — **in real time, in a simple way.**

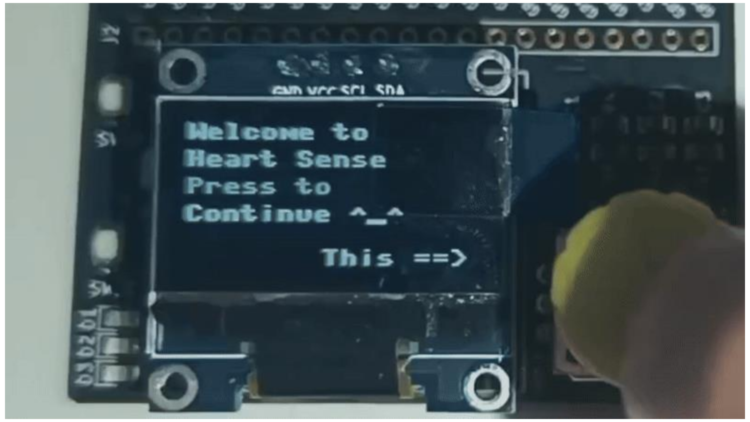
Our Vision:

To make HRV tracking **accessible**, **affordable**, and **interactive** — starting with a compact device that listens to every beat and empowers every user.

User-Friendly Design

Single Rotary Encoder

- ✓ No extra buttons needed
- ✓ Rotate to **navigate menus**
- ✓ Press to **start/stop measuring**
- ✓ **Ideal for elderly and children** — intuitive, minimal, and effortless interaction

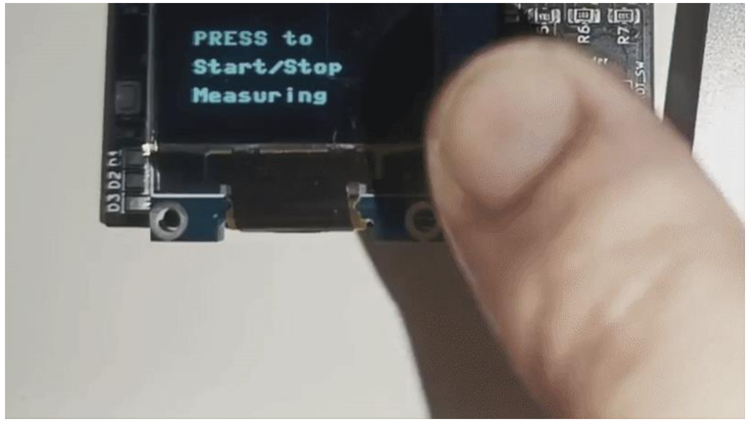


Criteria ★☆☆

Clear Measuring & Live Wave

Real-time interaction + data clarity

- ✓ **Live heart signal** dynamically updating
- ✓ Clear **countdown** during measurement to guide the user
- ✓ Data sent to **PC** via MQTT for deeper analysis

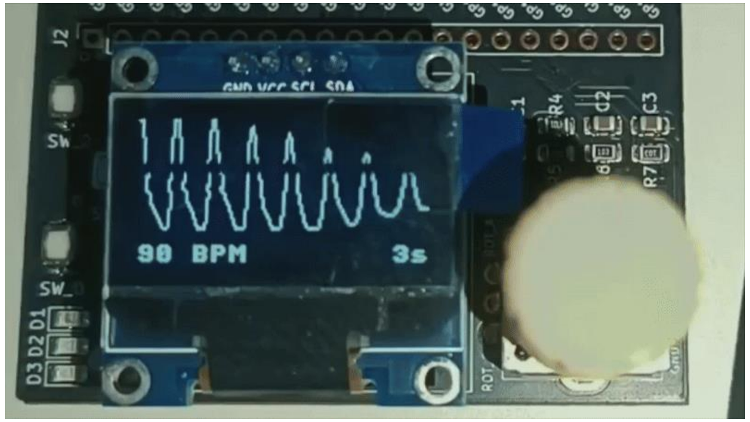


Criteria ★★☆

Stress Level Feedback

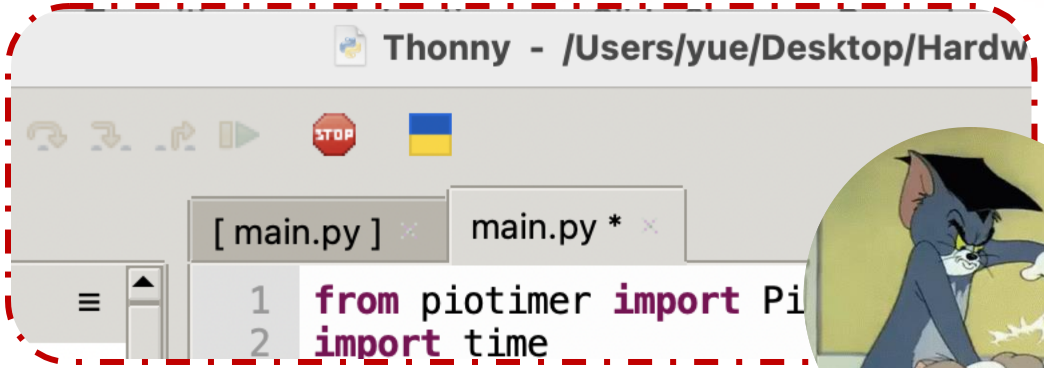
Stress awareness in real time

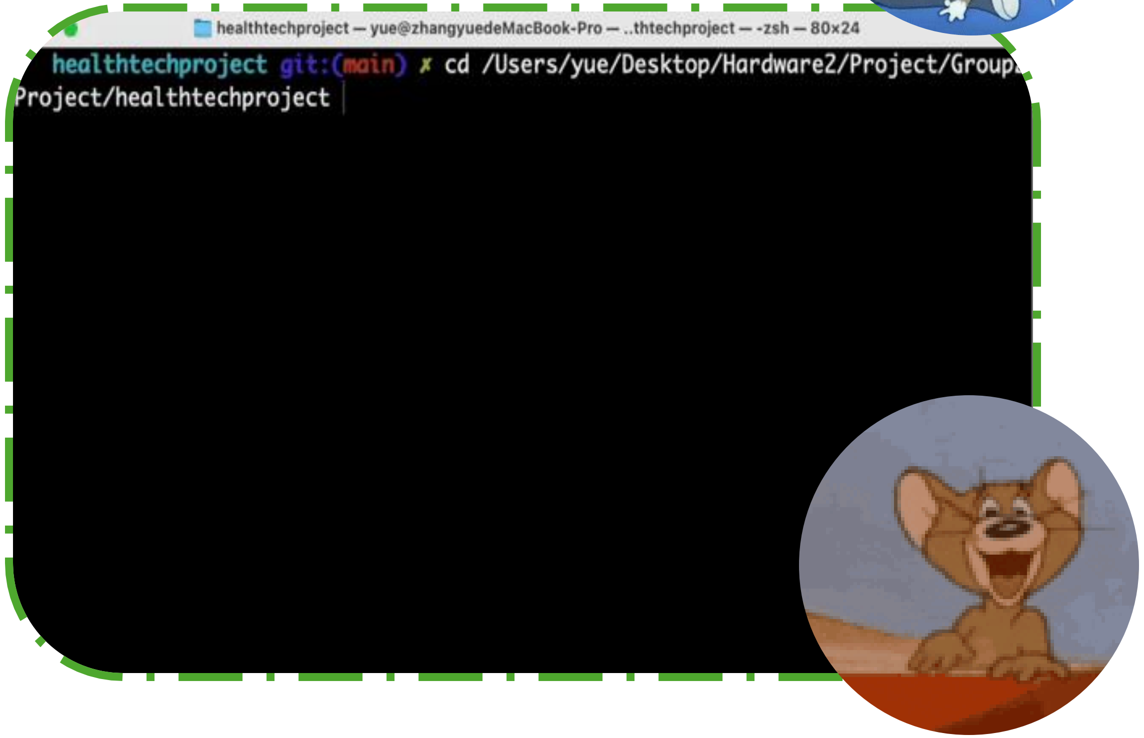
- ✓ Uses SNS/PNS analysis **from Kubios** to estimate stress
- ✓ **Feedback** shown on OLED
- ✓ Results **stored locally** on Pico for future review



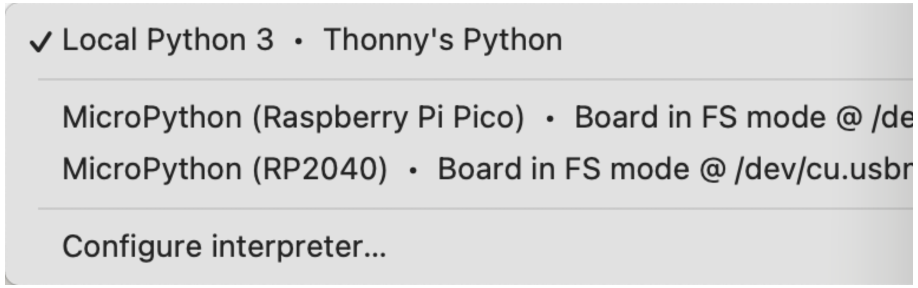
Criteria ★★★

HEART SENSE



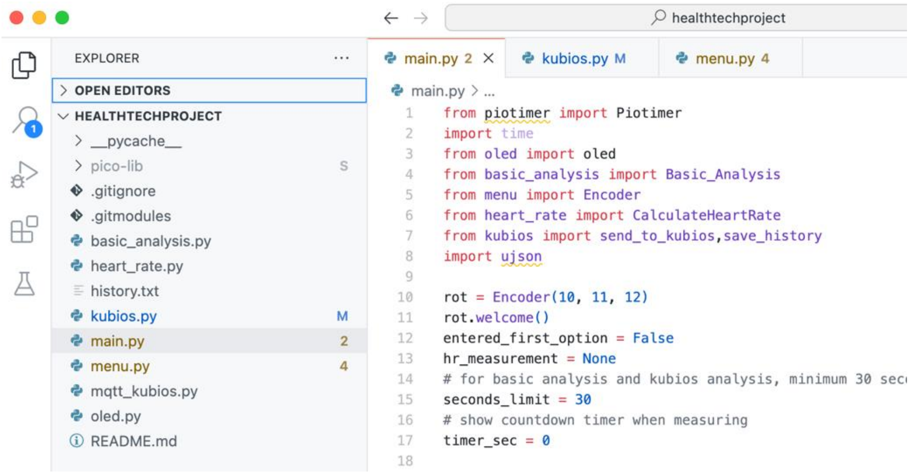


Tired of the clunky Thonny? Magic~



Life saver:

mpremote cp *.py :



Main & Modules

A clean structure: main.py handles logic, other files handle the magic.

main.py

OLED.py	Manages a single OLED instance to avoid I2C conflicts
Menu.py	Contains all functions related to the main menu and navigation
Heart_rate.py	Handles real-time signal acquisition and waveform plotting
Basic_analysis.py	Performs local calculations such as Mean PPI, SDNN, etc
MQTT.py	Manages MQTT connections and data transmission
Kubios.py	Handles communication with Kubios via JSON upload/download

Algorithm & Improvement (1/2)

Smoothing the Plot Curve

- **Separate FIFO queue** stores raw data.
- Every 25 samples take average.
- Added to the main FIFO.

Benefit: *Avoiding sampling rate issues from different sensors.*

```
self.samples_unsmoothed = Fifo(25)
self.samples = Fifo(500)
```

- **Scaled_history list** to store the last 128 scaled samples.
- When new data arrives, the oldest data is removed.

Benefit: *Ensuring a continuous display.*

```
self.scaled_history.pop(0)
self.scaled_history.append
```

- **Uses oled.line(x, y1, x+1, y2, 1)** to connect consecutive data points.
- Instead of oled.pixel()

Benefit: *Eliminating the "pixelated" look.*

```
for x in range(127):
    y1 = 48 - data[x]
    y2 = 48 - data[x+1]
    oled.line(x, y1, x+1, y2, 1)
```

Algorithm & Improvement (2/2)

Improving Detection Accuracy

Double Threshold Setting: both high and low thresholds per second

Benefit: *Reducing the occurrence of a secondary small peak.*

```
self.threshold_high = :
self.threshold_low = i
```

- **Data Slicing for Stability:** Excluding the first 4 intervals when send via MQTT

Benefit: *Removing early, unstable data points at the beginning of measurement.*

```
basic_analysis = Basic_Analysis()
basic_analysis.get_result
(hr_measurement.intervals[4:])
```

- **Heart Rate Validity Check:**
- Only calculate intervals within a valid BPM range (40-200 BPM)

Benefit: *Excluding extreme values at the beginning to ensure the quality of the interval list.*

```
if 40 <= bpm <= 200:
    self.isAnalyzing = False
    display_text = f"{bpm} BPM"
else:
    self.isAnalyzing = True
    display_text = "-- analyzing"
```

Source: https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2016.00653/full

Our algorithm achieved a **close match** (±5 BPM) with smartwatch readings under normal conditions

1. Comparison with Smartwatch Data
Used commercial smartwatches (e.g., Huawei, Xiaomi Watch) to compare BPM values in real time under resting conditions.

2. Stability Test
Tested the device across different durations (1 min, 3 min) to observe consistency in BPM readings.

3. Stress Testing with Noise or Motion
Simulated real-world conditions to ensure the algorithm avoids false peaks caused by small movements or signal noise.

4. Edge Case Detection
Tested the algorithm's response to:

- ✓ Abnormally fast/slow BPM
- ✓ Sudden signal drops
- ✓ Users with irregular waveforms (e.g., dual peaks)

Working with hardware requires patience—sensor noise and wiring issues were frequent.

Importance of threshold tuning and filtering for PPG.

Time synchronization between cloud and device is essential for accuracy.

Visualization helps in debugging and interpretation.

Task distribution and weekly planners helped keep on schedule.

HEART SENSE

Thank you!

Questions?

Hardware2 Project

Group2 Binary

Puntawat Subhamani

Olga Chitembo

Yue Zhang