**PheB Guide: Soft Robotics**

Plant-Human embodied Biofeedback (PheB) is a soft robotics project mimicking natural stimulus for emotional management in confined physical space. PheB is a multi-cell, continuum silicone surface where actuation is driven by air pressure regulation. This guide aims to provide a comprehensive description of its functionality in audio compilation and heart rate monitoring.

**Setting up**

1. **Software: Conda, PyCharm, and Arduino**

Feel free to skip if you already have Python and an IDE. Here are two common tools to start writing and running Python applications.

Conda is a friendly python package and environment management. Follow this guide for installation:

Download Conda — documentation

PyCharm is a common integrated development environment (IDE) for Python. You will likely want to use Conda with it. Follow this guide for configuration:

[Configure a Conda environment in PyCharm (jetbrains.com)](https://www.jetbrains.com/help/pycharm/conda-support-creating-conda-virtual-environment.html?keymap=primary_windows" \l "e417d6ec)

Arduino is utilized for hardware prototyping. Follow this link for installation:

[Download](https://www.arduino.cc/en/software) Arduino

1. **Python Packages**

The following packages are essential for audio compilation.

Run ***conda install \_\_package\_name\_\_*** or ***pip install\_\_package\_name\_\_*** in your terminal.

* *numpy*
* *matplotlib*
* *os*
* *librosa*
* *tkinter*
* *serial*
* *time*
* *re*
* *struct*

1. **Hardware & Arduino**

The project uses *Arduino Mega* for electronics control and *ITV 1000* by SMC for pressure regulation. Wall pressure is set at 40 psi.

**Audio Compilation**

1. **Overview**

The audio compilation is the general analysis of audio data and translation to pressure levels (inflatedness). It consists of three units, each with a different focus.

1. **Audio file compilation (Audio\_cmpl.py)**: PheB responds to an audio file based on amplitude. Audio will be played in synchrony with actuation. Compatible file extensions includes *.wav* and *.mp3*.

Technical implementation: the audio array is a sinusoidal data stream centered around zero. To get the local amplitude, maximums are extracted and averaged to a value per transferring time. The array removes outliers and bounds at a floor and ceiling that best reflects on the robotic surface.

1. **Real-time audio compilation (RealTimeAudio\_cmpl.py)**: Selected microphone will digitalize the surrounding audio volume and translates the value to PheB to achieve an interactive audio-to-visual stimulus transformation.
2. **Manual input (Manual\_cmpl.py)**: PheB reflects manual input by the user. Valid inputs are integers between 0 and 255 (8 bits) and preset tests (eg: t1, t2, t3). This is mainly used for testing regulator performance.
3. **Running the Code: main.py**

The three scripts listed above can be run individually. For convenience, however, a master script main.py integrates all functionalities.

1. Make sure the Arduino is connected to the local device and the script *PheB\_Arduino\_model.ino* is uploaded. Under *Tools*, ensure the *board* and *port* selected are correct.

Note that the Arduino serial monitor should be closed when running python scripts or an error will be raised.

1. Copy the *port* path to **main.py** and assign it to the *com\_port* parameter. No other parameters need to be changed.

To run the script, make sure you are in the correct directory with supporting resources.

1. Plug in the air pressure to the regulators at 40 psi. Ensure the vents are open at the right angles.
2. Run **python main.py** in the terminal. The user will be prompted to an interactive terminal. First, choose one of the three modes described above to run (enter 1, 2, or 3). Multiple sessions can be run consecutively, but not simultaneously. Resources required will be printed on the terminal; the user should respond via the keyboard. When an invalid input is received, the program will disregard it and request again. Enter *quit* or *end* to go back or quit the application.

When running Audio File Compilation, a file window will open. Navigate to and select the desired audio file to load. In the directory *audio* is a collection of example audio files.

At any point when the regulators are actuating, enter *ctrl + c* to kill the current python serial communication process. The program will return to its last state.

1. **Parameters**

To grasp a better understanding of how the code operated, feel free to look into the scripts. Here are some essential parameters that can be tweaked for a different effect. Note: not all parameters are used in all scripts.

* *reg\_min & reg\_max*: the maximum and minimum values regulators receive and act upon.
* *default\_min* & *default\_max*: default values to gauge loudness of a real time sound.

* *transfer*\_rate: rate of serial communication from python to Arduino.
* *dtime*: the temporal period in which the maximum is sampled for further processing in audio file compilation. Default: 0.1 (seconds)
* *freq*: frequency in which an audio file is sampled to an array. Default: 22050
* *input\_freq*: rate of sampling real-time audio from microphone. Default: 44100.
* *blocks*: rate of serial communication in real-time audio compilation.
* *buffer\_size* (seconds)/*window\_size* (frames): the size of buffer/window is inversely proportional to how much the program responds to rapid changes in audio.
* *baud\_rate*: rate of transferring data in serial communication. Default: 9600.

1. **Plotting & Visualization**

The function *plot\_general(y, x, sampling\_frequency)* is a good tool to visualize any audio time series. Inject them in the code where desired.

Chart

Description automatically generated

*Figure 1. The Original audio of example ocean wave*

Chart, histogram

Description automatically generated

*Figure 2. Modified audio*

Chart, histogram

Description automatically generated

*Figure 3. Final data*

**Ocean Wave & Heart Rate**

1. **Overview**

The heart rate model is solely in Arduino. The behavior is traced by a sinusoidal ocean wave equation simulated in MatLab.

A heart rate sensor is attached to the user. An increase in heart rate roughly signals an activated sympathetic system: stressful and alert. A decrease in heart rate signals an activated parasympathetic system: relaxed and energy conserving. An elevated heart rate leads to increased pressure amplitude and ocean wave frequency, and vice versa. PheB reflects the user’s physiology in the biophilic environment, which allows user to monitor their own body to a healthy state, enhancing positive emotions.

1. **Running the code: heartrate\_sensor.ino**
2. Plug in the air pressure to the regulators at 40 psi. Ensure the vents are open at the right levels.
3. Make sure the Arduino is connected to the local device and the script *heartrate\_sensor.ino* is uploaded. Under *Tools*, ensure the *board* and *port* selected are correct.
4. A heart rate sensor should tag on the earlobe of the user. Heart rate should update roughly every five seconds. If not detected, default values are used.

Open the serial monitor to see the current regulator and heart rate values. Note that the regulators act on a range from 0-255 (8 bits).

Turn off the application by turning off the power of the Arduino.

*Guide written by: Ian Lee 05/2022*