University College London

GS-PAT: Board field format specification

Multi-Sensory Devices lab

Diego Martinez Plasencia
10-1-2020

Change Log

Date	Change
30/03/2020	Initial version (Diego Martinez Plasencia)
05/02/2021	Format Update – Public (Roberto A. Montano Murillo)

Table of Contents

1 About this document:

Acoustophoretic setups make use of our acoustic boards. However, not all boards are exactly the same, and there is the possibility that boards can change in the future. For instance, each transducer provides a slightly different phase offset, which is calibrated after manufacturing and which is specific for each transducer in each board. Other parameters include the local position of each transducer, their amplitude or their hardware ID

Your specific board will come with an accompanying ***configuration file*** containing these parameters. These need to be provided in runtime for the correct operation of the boards.

The current file describes the format we are currently using, for our internal maintenance (e.g. how our calibration software needs to safe files), and for understanding of the implementation.

We first describe the parameters considered and their purpose, and then describe a very simple LL0 grammar to describe it …

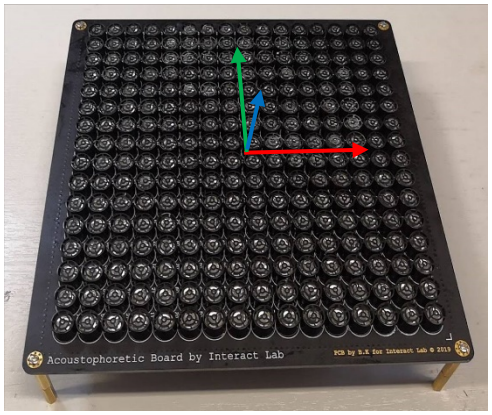
2 Parameters contained in the file:

Each of your boards are identified by a unique ID, which matches the COM port number detected when you connect your board to your computer (you can check this in the *Device Manager*) and also matches the name of your **configuration file**. More specifically, for a board with ID (i.e. COM port ***n***), the name of the file would be:

*Board_**n**.pat*

This section describes the parameters contained and describes their purpose:

1. **Hardware ID:** We use an USB connection (FTD2) to achieve high bandwidth communication with the boards. These connections are bound to the COM port, but have a hardware ID that does not match such port number (e.g. our first board with ID 1 had hardware ID "FT4TKZL8"). This field contains such name, making this mapping transparent to the user.
2. **Number of transducers:** Our current implementation assumes boards with 256 transducers per USB connection, but this could be subject to change in the future. This field contains the number of transducers in your board.
3. **Transducer positions:** The placement of the transducers could also be subject to change (e.g. non-square boards), so this field describes the position of each of the transducers in coordinates **local to the board**. We use the system of reference in Figure 1a (XYZ axis encoded as R-G-B arrows) and meters as our unit.
4. **Transducer IDs to PIN IDs:** Transducers are logically arranged as a linear array. For instance, in our current boards transducer 0 is placed in the top left corner of the board, while transducer 255 would be located in the bottom right. However, the hardware can (and does) use a different arrangement, describing how transducers are connected to the pins in the circuitry (see documentation on *Board Manufacturing*). This field contains a mapping



describing, for each transducer ***t*** (e.g. ordered from top left to bottom right in our current boards) its associated hardware PIN **p** (e.g. ordered to match hardware requirements). This is required by our driver (*AsieInho*) to send update packages to the boards according to their actual PIN layout.

5. **Phase correction (per PIN ID):** As indicated before, each transducer has a different response which must be considered for optimum use of the board. This field contains the static phase offset correction required. Please note that this is stored in degrees for each transducer, but the order is that of **the PIN ID of the board** (e.g. not the nice top-left to bottom right transducer order).
6. **Amplitude correction (per PIN ID):** The maximum amplitude of each transducer also shows some variance, which we store here. Particularly, this field contains the pressure in Pascals delivered by each transducer at 1m distance. Again, this is stored according to **PIN ID**

3 Grammar

3.1 Lexical level:

Our grammar makes use of the following low level lexical categories, which can all be easily parsed with conventional functions (e.g. *fscanf()*):

- **float**: Single precision floating point number
- **integer**:
- **string**: Null terminated character string

The description also makes use of other conventional separators, which we identify using conventional C notation (e.g. **'\n'**, **','**), in boldface, to indicate they are lexical categories.

3.2 Syntactic level:

This section describes the syntax of our grammar, with clauses identified in *italics* and conventional notation for regular expressions. Semantic actions (between curly brackets) simply imply storing the values read and are mostly omitted, except for relevant cases.

File → *HardwareID NumTransducers TransducerPositions PINMapping PhaseCorrection* [*AmplitudeCorrection*] ***eof***

HardwareID	→ **string '\n'**	
NumTransducers	→ **integer '\n'**	{*n*:= **integer** }
TransducerPositions	→ [**'(' float',' float ',' float ')' ','**]^n **'\n'**	
PINMapping	→ [**integer ','**]^n **'\n'**	{if (**integer** > 255 || **integer** <0) parse_error}
PhaseCorrection	→ [**integer ','**]^n **'\n'**	{if (**integer** > 360 || **integer** <0) parse_error}
AmplitudeCorrection	→ [**float ','**]^n **'\n'**	{if (**float**<0) parse_error}