PSYCHOGEAR, YET ANOTHER PSYCHOPHYSICS LIBRARY

Diego Dall'Alba, Marco Vicentini, and Debora Botturi
Computer Science Department, University of Verona, Verona, Italy
{*dallalba, vicentini, botturi*}*@metropolis.sci.univr.it*

Abstract

This paper presents PsychoGear, a new library of psychophysics methods, which can supplements haptic, visual, and audio stimuli. We have developed a modular framework that arises from the decomposition of an experiment in a series of psychophysics methods, each one composed of a variable number of trials. Constant stimuli, staircases, QUEST, PEST, and Maximum Likelihood adaptive procedures for psychophysics measurement are natively implemented in our library. Implementation details such as trial presentation, stopping rules, generation of the next stimulus, and data collection are natively handled by the framework. By this way the user (researcher or student) can be concentrated on the design of the experiment. Our library is very easy to use with simple and efficient C++ code, or with a dedicated GUI to create a XML configuration file of the entire experiment.

Only few works in the literature bring together physiological, perceptual and cognitive factors, such as velocity, force, reaction time, mental workload, EMG measurement, and transmission delay. The main reason is that experiments testing a number of factors are not trivial, and the design implementation has to carefully take over the synchronization and the interface between different stimuli. We have experienced the difficulties to implement a multi-factorial experimental design, even harder when the subject has to interact with the environment where there are noisy sensors, delayed stimuli and complex devices. Recent experiments (Vicentini & Botturi, 2008) show the needs of multi-factorial design, in which several cognitive factors have to be evaluated in order to model the human perception in haptic system.

We are working on the development of a new psychophysics library to control all the aspects of the experimental setup, maintaining the rigor of the psychophysics methodology within an haptic system. The library will be able to control the experiments by governing stimuli generation and synchronization, data acquisition from different sensorial channels and data logging.

Our approach follows the research line described in Anderson (2001), that is, to combine multiple psychophysics measurements within a full factorial design. We are working on a modular approach that arises from the decomposition of an experiment in a series of psychophysics methods, each one composed by a variable number of trials. For each trial we consider the presentation of a stimulus, the hardware source, the psychophysics procedure type (classical or adaptive) and the logging of the subject's response.

The focus of the approach described is on the infrastructure, not on a specific method. We want to provide a framework in which including a new experimental method is quick and easy. The user can define the design and choose the parameters without dealing with the underlying code because we have decoupled the programming phase from the utilization one.

Moreover, we pursue the code reusability and all the implementations such as trials,

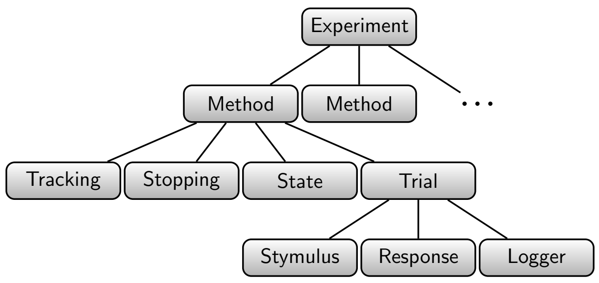


Figure 1: Experimental Psychophysics Library Object Hierarchy.

stopping rules, generation of the next stimulus, and data collection are stored in the framework.

With this library the user can concentrate on the design of the experiment putting together methods he/she needs, the programmer can rely on the framework to implement new details, without even know the psychophysiscs purpose. Finally, we point out in this library the possibility to have multi-factorial design and, an easy device integration.

PsychoGear

Starting from the ideas of the Psychophysics Toolbox (Brainard, 1997) for MATLAB, the PsychoPy–Psychophysics software in Python (Peirce, 2007), and the GroovX framework (Peters, 2008), we are looking at providing new classes and methods to cover several aspects, from stimulus presentation and response collection, using at the same time classical and adaptive procedures, to data analysis, such as psychometric function fitting.

The library code is written in C++ and it works directly under the most common operating system (Linux, Mac OSX, and Windows). The component interface and the code management are very simple, therefore the implementation of new parts is straightforward even for programmers with little experience. Our framework supports C-style code, very useful when dealing with hardware devices; it also supports high level features, such as data abstractions and object-oriented programming.

The modular structure allows an easy software maintenance and debug. We use compiled code, also because it is faster than the interpreted, thus real-time is possible, even in highly complex cases, such as haptic experiments with multi-modal stimuli.

The major effort in the library development is the management of haptic experiments, that is possible to consider as a broad class that include the visual experiments. We support multi-factorial design, hardware independency and component exchange and different type of stimuli and responses. Therefore the library is able to visualize visual stimuli, supports different haptic devices and, can communicate with further devices through a generic TCP communication protocol.

The basic set of components for a psychophysics paradigm is made of an update rule for the characteristics of the stimulus, which should balance between different dynamic parameters, and a stopping criterion. Hence, if the rules of the experiments are set out, there is no need for complicated control procedures and the execution control can be very simple. Moreover, in literature we can find a limited number of tracking strategies and stop rules, thus we ensured that once implemented it is possible to easily reuse them in any experiment.

With our library, once a psychophysics method has been implemented, it can be

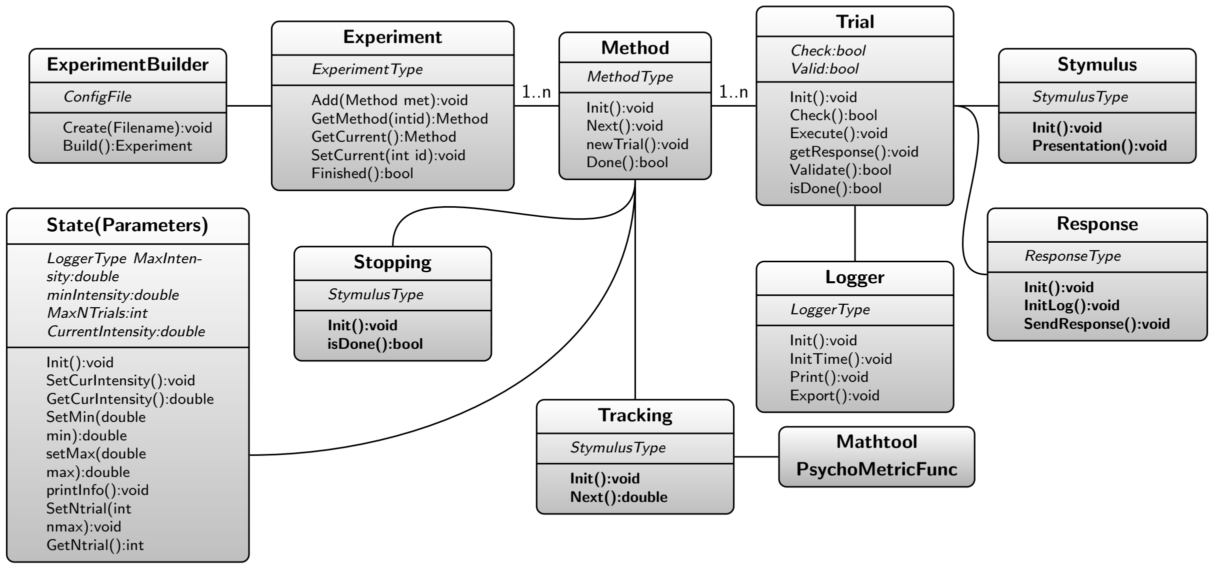


Figure 2: Experimental Psychophysics Library UML Class Diagram.

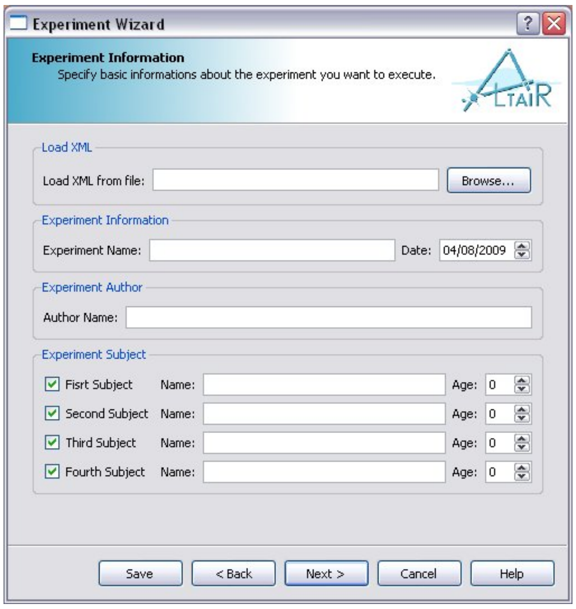
reused in a totally different scenario, with others methods, criteria and devices by simply instantiating a new set of objects. We provide the implementation of the most common psychophysics paradigm, the ones discussed in (Leek, 2001), and we count on the sharing of classes between users.

A stimulus presentation always requires some low level code, that communicates with the hardware device. In our library the support for each type of hardware device is of immediate use and only a new Stymulus class has to be developed. The use of the library is twofold: the implementation of the classes and the use of them. The coding part requires programming capabilities (i.e. a "programmer"), the use focuses on the choice of proper parameter values.

Class Structure

In Fig. 1 we show the object hierarchy: an *Experiment* contains an arbitrary number of methods. Each *Method* contains a *State* class, which contains all the parameters needed for a correct execution of all the sub-component, a *Tracking* class, that controls the stimuli intensities, a *Stopping* class, that checks the end of the method, and a *Trial* class. *Trial* is composed by a *Stymulus* class, which deals with stimulus presentation, a *Response* which collects user responses and a *Logger* for data logging. Fig. 2 depicts a simplified UML class diagram drawing the PsychoGear framework. Each class has a simple and easy to use interface.

The main component of this framework is the *Method* class that supervises all the different control aspects of the experiment procedure. We have to present a sequence of stimulus intensities, that are known a priori or settled at run time, and for this purpose the *Tracking* class gives basic commands to control the evolution of the stimuli in the procedure. A very important aspect is the procedure ending management. The *Stopping* subclass checks the end of the procedure and also allows to easily change between different stopping rules. Moreover following the psychophysics definition, *Trial* class is implemented with a *Stymulus* subclass, that manages the stimulus presentation, and a *Re-*


Experiment Wizard

Experiment Information

Specify basic informations about the experiment you want to execute.

Load XML

Load XML from file:

Experiment Information

Experiment Name: Date: 04/08/2009

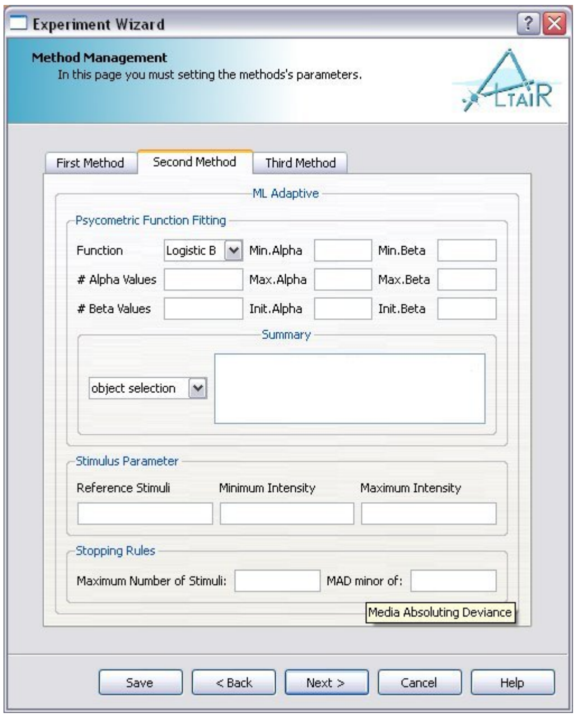
Experiment Author

Author Name:

Experiment Subject

✓ Fisrt Subject	Name:	Age:	0
✓ Second Subject	Name:	Age:	0
✓ Third Subject	Name:	Age:	0
✓ Fourth Subject	Name:	Age:	0

(a) Experiment initial setup


Experiment Wizard

Method Management

In this page you must setting the methods's parameters.

First Method | Second Method | Third Method

ML Adaptive

Psycometric Function Fitting

Function	Logistic B	Min.Alpha		Min.Beta	
# Alpha Values		Max.Alpha		Max.Beta	
# Beta Values		Init.Alpha		Init.Beta	

Summary

object selection

Stimulus Parameter

Reference Stimuli | Minimum Intensity | Maximum Intensity

Stopping Rules

Maximum Number of Stimuli: MAD minor of:

Media Absoluting Deviance

(b) ML method parameters tuning

Figure 3: Views of the graphical user interface to create XML configuration file

sponse subclass, which controls the responses acquisition. In addition, a *Logger* subclass is implemented to save all the meaningful parameters that describe user performances.

Besides the psychophysics procedure uses some statistical tools, for example probability distribution and random number generator, therefore the *Mathtool* class is implemented to contain all the auxiliary mathematical functions.

It is common to design a real experiment which involves more than a single psychophysics procedure (for example, two staircase procedures with random stimulus presentation). So we need to control which procedure has to be executed, how many trials have to be presented, when the entire experiment finishes. To manage this type of design, the *Experiment* class manages different methods defining a global end criteria and a switching rule between methods in every step of stimulus presentation. The specific parameters of the classes involved in the experiment are stored in the *State* class. This class contains all the parameters in the same place, so it is easy for every sub-class to access useful sharable information.

When an experiment is clearly designed, it may be seen as a set of psychophysics procedures and parameters (i.e. stimulus intensities, starting values, stopping rule, presentation time). Especially during the pilot testing, the user needs to tune some parameters to fit the current design to the goal of her/his study.

To make this operations easier, we set a unique XML configuration file, where the user can easily choose the components and set the parameters of the experiment. The *ExperimentBuilder* class reads the configuration file and initializes the experiment with the correct sub-classes and parameters.

Features

Typically, building and managing an experiments requires to write hundreds lines of code. Our library permits to obtain the same result with few XML lines.

A frequent problem is that parameters of the experiment are spread all over the code, making the section decouple more difficult. In our framework, thanks to the modularity and the global parameters setting, each extension can be implement with minimal modifications to the existing component, and the modification is always limited to a precise set of components.

Once the main classes are developed, the experiment execution and control is very simple and it needs few line of code, as you can see in the next example:

```
int main()
{
    // Create an Experiment Builder with the proper configuration file
    ExperimentBuilder builder("CONFIG.XML");
    // Create Experiment from parameters in config file
    Experiment *exp = builder.Build();
    Method *cur;
    //Execute Experiment while finish is not reached
    while (!(exp ->Finished()))
    {
        //Select the current method
        cur = exp->GetCurrent();
        //Execute all trials
        while(cur->isDone())
        {
            // New trial from current method
            cur -> newTrial();
            // Invoke traking algorithm to refresh current method parameters
            cur -> Next();
        }
    }
}
```

Parameters can be adjusted with no code change or code recompilation. A typical method to define the experimental parameters is to use a plain text file, in which storing the needed information. A XML file is the closest solution to a plain text, but it also provides a clear description of each inserted parameter; a XML file increases the readability and makes easier changes and updates. Indeed, the XML configuration file stores all the parameters needed for correct execution and with a simple text editor it is possible to change them.

We also provide an intuitive graphical user interface to create XML configuration file in a simple e comfortable way. The interface guides the user in the creation of complex experimental setup with a step by step procedure, that helps to choose the correct component and suitable parameters. In Fig. 3(a) we show an initial experiment setup where users can load a previously created XML file or insert a new experiment, author and, subjects informations. In Fig. 3(b) we show a configuration form for Maximum Likelihood method parameters (in the example we have an Experiment with three different methods, i.e. constant stimuli, ML and staircase).

Discussion

In this paper we have presented a psychophysics library that proposes an easy to use structure, taking advantage from the experience of other implementations and the needs of the haptic perception experiments. The novelty of this library is the native management of the psychophysics procedures and of the haptic system; moreover some problems from the existing libraries are solved in this implementation ensuing the software engineeing suggestions (i.e. object oriented code organization, haptic device support). The library architecture is meaningful for the organization and the development of every experimental design,which is even more important in haptic research, given the complexity of the experiments designs.

In the near future we plan to extend the implementation of the basic components, adding, for example, the support for all the fundamental adaptive methods. Once a wide set of example will be developed we will use them as an "how to" for our library, to allow an easy and fast startup time for new users.

We are planning to widen the hardware device support and to improve their integration with other source of stimulus devices, i.e. audio/video, with a unified synchronization protocol. We are also introducing support for OpenGL to improve the rendering of visual stimuli, and we are working on support for OpenCL to obtain faster parallel computation (i.e. in psychophysic functions fitting evaluation). Moreover our graphical user interface will be improved implementing the drag and drop of the components during the creation procedure and allowing the real–time managment of the experiment execution, broadening the PsychoGear library use.

Acknowledgments

This work was partially founded by the AccuRobAs project under EU's 6th Framework Programme (contract IST-045201).

References

Anderson, N. H. (2001). *Empirical direction in design and analysis.* Mahwah, NY: Lawrence Erlbaum Associates.

Brainard, D. H. (1997). The Psychophysics Toolbox. *Spatial Vision*, *10*, 433-436.

Leek, M. R. (2001). Adaptive procedures in psychophysical research. *Perception & Psychophisics*, *63*(8), 1279-1292.

Peirce, J. W. (2007). PsychoPy-Psychophysics software in Python. *Journal of Neuroscience Methods*, *162*(1-2), 8-13.

Peters, R. J. (2008). *The groovx framework v. 1.0a1.* Retrieved July 21st, 2008, from `http://ilab.usc.edu/rjpeters/groovx/`

Vicentini, M., & Botturi, D. (2008). Overshoot effect in stiffness perception tasks during hand motion with haptic device. In M. Ferre (Ed.), *Haptics: Perception, devices and scenarios* (Vol. 5024/2008, p. 189-198). Berlin: Springer.