The choice of software and hardware in psycholinguistics: review and opinion

A escolha de software e hardware na psicolinguística: revisão e opinião

Thiago Oliveira da Motta Sampaio
Language Acquisition, Processing and Syntax Lab – LAPROS
Universidade Estadual de Campinas, Campinas, São Paulo / Brasil
thiagomotta@iel.unicamp.br

Resumo: Nos últimos anos, diversos *softwares* foram criados para auxiliar a elaboração de experimentos em ciências cognitivas. A oferta de *softwares* de simples utilização deveria facilitar o trabalho dos iniciantes, porém, acabou trazendo novos problemas e dúvidas. Que *software* usar? Qual deles é o mais adequado ao meu estudo e por quê? Através de uma revisão sobre computação, linguagem de programação e técnicas de apresentação de estímulos visuais, este artigo pretende fomentar a discussão a respeito (i) dos diversos tipos de *softwares* para estimulação, (ii) da importância de conhecer os detalhes técnicos do *hardware* utilizado e (iii) da compatibilização *hardware*-*software*-método como uma variável a ser controlada durante o desenvolvimento do protocolo experimental.

Palavras-chave: psicolinguística; ciências cognitivas; linguagens de programação; métodos.

Abstract: In recent years, several software have been designed to aid in the development of experiments in cognitive sciences. The offer of user-friendly software should help beginners in their initial studies; however, it has brought new problems and questions. Which software

eISSN: 2237-2083
DOI: 10.17851/2237-2083.25.3.971-1010

should one use? Which one is more appropriate for my research and why? The present paper brings a quick and panoramic review of computer science, programming languages, and the presentation of visual stimuli. Through these three topics, I intend to promote a discussion (i) on the main types of software for stimulation in cognitive sciences, (ii) on the importance of being attentive to the hardware specifications, and (iii) on some compatibility issues between software-hardware methods as independent variables in our experiments.

Keywords: psycholinguistics; cognitive sciences; programming languages; methods.

Received on December 5, 2016

Accepted on March 27, 2017

1. Introduction

The following study will not present a hypothesis to be tested by an experimental method. This is a paper on methods, aimed at questioning the routine of a cognitive scientist who works with experimental psychology/psycholinguistics, especially that which concerns the development of experiments on a computer.

First, it is important to emphasize that the drafting of an experiment on the computer does not necessarily require advanced knowledge of computer programming. There is a wide range of software for this task for both advanced users and beginners in computer programming; therefore, grants to buy software are not necessarily a problem. For the majority of experiments, one need not shell out hundreds of dollars for the acquisition of software, since there are many open source options, which are fairly efficient and easy to learn. Nevertheless, for some reason, these types of software are not well known among Brazilian researchers.

The second question that I wish to present in this article is that, despite the broad offer of applications, we should not completely trust the software in the task of the communication between the user and the machine. Contrary to recent operating systems, the software do not hide from the user the configurations that are impossible to be performed by

the hardware, nor do they provide the necessary information about the behavior of the hardware during the test. Many times we believe we have total control of our variables without realizing that the computer is not properly executing the tasks that we asked it to execute.

After presenting my main questions, I intend to begin a discussion about these two aspects. Section 2 follows by problematizing the common view of computer programming knowledge; section 3 aims to clarify what computer programming languages are and how they work; section 4 presents a wide range of software for experimental design in cognitive sciences, among programming languages, toolboxes, as well as paid and free software with graphic user interface (GUI); section 5 discusses the problems involved in the controlled visual experiments that stem from the lack of knowledge about the hardware used in its application. I will close this paper with some final considerations.

2. A quick discussion on software and hardware

Computer Science has a rule that seems to predict the rhythm of technological progress: Moore's Law (1965). This law demonstrates that computers increase their complexity exponentially, doubling their processing capacity every two years. If we use this rule to look back to the past, we can see that the beginning of computer programming occurred in the 1960s, exactly with the invention of the first chip.

Through Moore's Law, it is possible to predict that even the most enthusiastic of users are unable to keep up with technological progress in its totality. While the quantity of information increases over time, people take on more responsibilities, which requires a certain amount of time to update themselves on all fronts of technology.

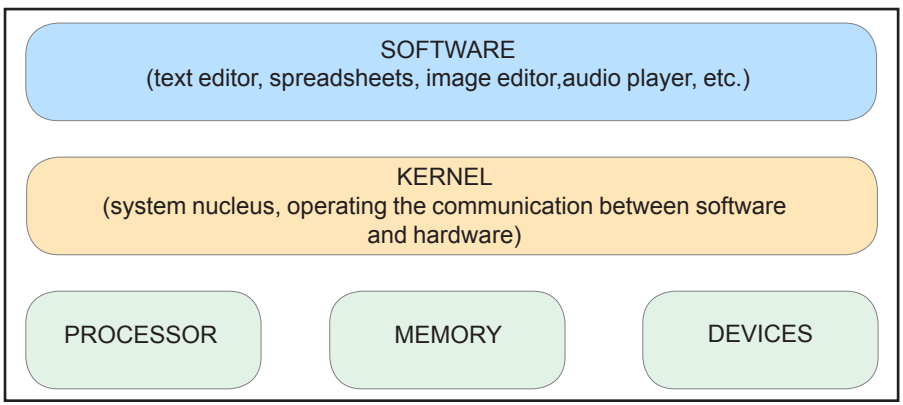
In this context, we are invaded by the idea that the children of today are digital natives and naturally learn to use advanced technologies that even the technology enthusiasts are no longer able to dominate. However, on many occasions, this topic was discussed in personal conversations and even at a round table discussion at the 3rd Meeting for Scientific and Cultural Dissemination, held at the University of Campinas, and it was found that this is only a half-truth.

It is undeniable that there are (i) young users who truly take advantage of the accessibility to new technologies, to dominate their resources and become excellent programmers. On the other hand,

what we observed in most of cases is that (ii) the software (especially proprietary software) have become increasingly available both financially, as they have become cheaper, and user friendly. This brings an illusion of technological inclusion, in which many less advanced users are able, with relative ease, to perform tasks that would be highly complex for computer enthusiasts only a few years ago.

This can be observed quite clearly when we verify the evolution of mobile operating systems. Ever-increasingly popular, these systems, which today nearly every child has in his/her pocket, have reached the point of dividing or even substituting functions that were carried out exclusively by expensive and inaccessible computers. The result is that many of these people who considered themselves to be experts in information technology have nothing more than a vast knowledge about how to use the wide range of software available on the market. These users are able to use many types of computer programs in a skillful manner through its user interface, in the *front-end* (Figure 1). However, they often have an extremely limited knowledge about the communication between software and hardware, and of troubleshooting, the *back-end*.

FIGURE 1 – Kernel (nucleus) of the system: the bridge between software and hardware



Note: Kernel is the center of an operating system, its nucleus. It is responsible for serving as a bridge between the software and the hardware of a computer. Front-End users (interface-users) are generally limited to the knowledge of the software, without the need to understand the inner workings of the computer at the other levels. We can make an allusion to the visible part of an iceberg, when the major part of the rock is submerged and outside of our field of vision.

It is not difficult to find an enthusiast from the 1990s or the beginning of the 21st century who has had to deal with innumerous incompatibility issues of a new hardware (ex. sound and video cards), or the issues of installing a compatible hardware, but without the right driver[1] to carry out the communication between the operating system and the computer.

When we install Windows or Linux for the first time, they generate generic drivers so that the hardware functions in minimal configurations. The drivers for each component are searched after installation so that we can use the machine in its full potential. Formerly, we should manually search for drivers and install them. By passing through these experiences and searching for solutions, users ended up gaining at least the basic notions of hardware-software communication.

Today, both the hardware and the software, as well as the communication between them, have become more efficient, allowing the operating systems to hide the options that the hardware does not support (much of this due to the presence of the correct drivers for the installed hardware), which avoids part of the more basic problems that we faced in the 1990s. In addition, the most recent versions of the systems already come with a library of the most commonly used drivers. When the system do not contain the proper driver for the devices, they offer the option of an automatic search. Moreover, the user can always find the driver on the manufacturer's webpage, in CDs or in flash drives that come with the device.

Mobile devices can also suffer from the underuse of its functions caused by generic drivers. One key example is the quality of photos of a smartphone. Contrary to the iOS, which have standardized their hardware and forces third-party developers to use the official applications, Android needs to adapt itself to the diverse types of hardware from different brands. Each brand has an optimized photo application for their devices, with a wide range of filters that improve the performance of the images in the post-processing. Third-party applications, however, use a generic driver to directly access any camera, which has consequences on the image quality.

[1] Software containing instruction of communication between the operating system and the hardware.

The knowledge on how the computer receives the inputs may not be necessary for some users, since everything seems to be properly working. By contrast, this knowledge can be important for us to make better choices of software, to solve some technical issues and, especially, when we wish to strictly control how the hardware presents the stimuli and collects behavioral data.

The first step towards discussing the communication between the software and the hardware is the discussion on what the computer programming languages are and how they work. To achieve this, I will limit myself to those which are commonly used by researchers in cognitive sciences, which includes psycholinguistics, in the main American and European laboratories.

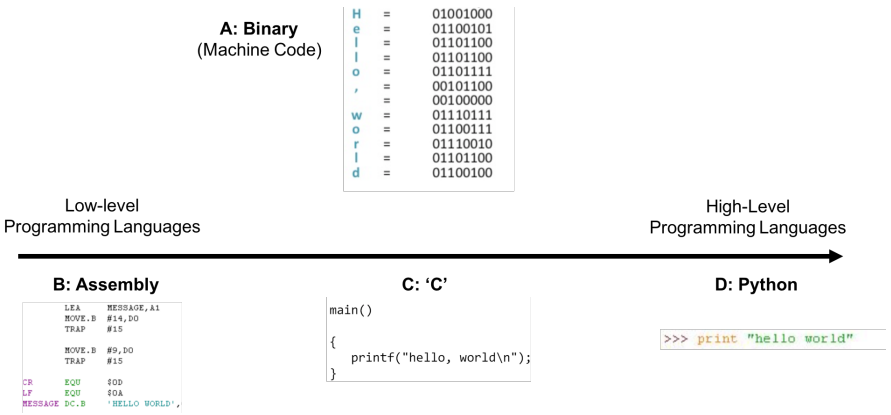
3. What are the Computer Programming Languages

Classic computers[2] receive input in the form of electric current inflows, corresponding to the numbers 0 (turned off) and 1 (turned on), which we call *bits*. This makes it possible to distinguish two pieces of information. To increase the amount of information to be processed, computers were developed in such a way as to work in an eight-bit (1 byte) system. Each sequence of eight bits can be related to different information. For example, the representation of the number 8 corresponds to the sequence 00111000, while the letter "e" (lower case) corresponds to the sequence 01100101. This binary code, known as the machine code, is the only one understandable by the machines. However, giving instructions to the computer directly in binary code is not the easiest way of programming. For this reason, computer programming languages (hereinafter called CPL) arose in an attempt to make it more accessible.

[2] In a nutshell, classic computers are different than quantum machines. Please notice that the classic machines work with the transmission of bits, which can vary between the off (0) and on (1) states. The quantum machines work with so-called qubits, or quantum bits, in which the information, in addition to the binary states, can be in a position overlaying 0 and 1 (vectors). This will have an effect on the quantity of information transmitted each time and in the manner of transmission and reading of the information by the quantum computers.

The CPLs are classified according to their level of abstraction. The low level languages have an operation that is closest to the group of instructions supported by the computers, which demands a greater learning curve. The high level languages are closer to human language, using functions in syntactic and semantic relationships that are easily memorized and learned (Figure 2).

FIGURE 2 – CPLs and levels of abstraction



Note: The first step in a computer programming course is to write a code that makes the machine show the phrase "Hello World" on the screen. Figure A corresponds to the letters in binaries; Figure B uses an Assembly to exhibit this phrase; Figure C presents the same command in C language; Figure D executes the exhibition of the text in Python. Please note that the writing of the code becomes easier from A to D. The arrow exemplifies the concepts of languages of low-level abstraction (closer to the binaries) and those of high-level abstraction (closer to human languages). The manner in which we should write and organize the functions and arguments is called syntax.

Assembly is a prime example of a low-level language. Actually, Assembly is not exactly a specific CPL, but rather the name given to the single language of each processor, containing a legible and memorizable structure of the group of instructions that a computer can perform. More advanced programmers can use Assembly to give instructions to the computer or even to develop applications. The drawback of the development in Assembly is that they are not intelligible to other

computers. Moreover, programming in Assembly is still a task for experts in the field.

Computer programming only became popular with the development of higher-level languages. By contrast, as in natural language, when two interlocutors do not know a common language, a translator/interpreter is necessary to establish communication. This generates a processing cost and, consequently, an increase in the computer's response time.

A language is a software that allows us to create a sequence of steps/functions called 'algorithm'[3]. This algorithm will be read and executed by the computer, however, computers only understand binaries. For these instructions to be understood by the hardware, the language needs to be **'compiled'**, which means, translated to binaries what, for instance, is done by C and C++, or **'interpreted'**, which means, transforming the command lines into a byte code, which will be interpreted by a virtual computer, which is done, for instance, by Java and Python.

One of the most commonly used languages today is 'C'. As it is created with the aim of developing operating systems, C is considered the *lingua franca* of computer programming, much in the same way as English is among natural languages. Hardware manufacturers, in addition to Assembly, generally also write a code that maps the functions of its devices directly to C, thus facilitating the work of developers. Despite the practicality introduced by this language, computer programming was still relatively restricted. In addition, many users have quite specific needs that could be carried out in a simpler fashion and in a different programming logic than that used by C. Thus a wide range of high-level languages began to appear, such as Python, R, Matlab, and Java. The advantage of these languages is that all allow us to execute each step of the code throughout the programming to verify if it will function properly in practice. This in turn facilitates (i) the identification and localization of errors, which are normally indicated by the console itself (screen where the code is written) during programming and (ii) the learning, due to its

[3] An algorithm, in general, can be defined as an ordered sequence of steps that lead to a specific result; hence, it is a finite sequence. An algorithm does not necessarily need to be something mathematical or computational.

immediate feedback and identification of the syntactic arguments through different colors, as can be seen in the example of Python (Figure 2).

Java is widely used, since, in addition to being free, it was heavily marketed by its developer, Sun, which today is owned by Oracle. The market then began to demand prior knowledge of Java to hire computer programmers, which led universities to offer technology courses on Java. Although it has lost market space more recently, Java is the programming language adopted by Google to develop applications for Android systems. The Java is interpreted and translates the code to a virtual machine (*Java Virtual Machine*, JVM). This virtual computer is like an emulator that simulates a machine in any computer, preventing the code from having to be recompiled.

Python works in a similar way, using a virtual machine that is installed in any computer and allows it to be executed in any computer, regardless of the operating system. This favors the portability of its codes and makes it one of the preferred languages of those who work with experimental methods.

Matlab is a proprietary software based in C and Java. It was created with the aim of facilitating programming based in data matrixes. Though the codes written in Matlab run directly within the software, Matlab contains a compiler and a runtime that allows us to run our codes outside of its interface. One free option to Matlab is GNU Octave, whose syntax is quite similar to that of Matlab, which facilitates migration. Another option with a similar syntax is Julia (BEZANSON *et al*., 2014), which, in some tasks, presents an excellent performance.

By contrast, R is well-known by all linguists and other researchers who work with statistics, corpus analysis, or data mining. It is a free software developed to facilitate the work with numerical and statistics data, which is widely used in cognitive sciences. Psycholinguists commonly use R to analyze data due to its programming logic and to the diverse, freely distributed scripts for this purpose, but nothing hinders it from being used for the design and the application of experiments.

In sum, we have thus far observed that the processor contains an architecture that receives a type of information to execute an algorithm. This information can be elaborated by means of a programming language that, in addition to facilitating the tasks of the developer, can also be translated to the machine through the compilers and interpreters, in

exchange for a performance loss related to the processing cost of the algorithm and of the translation method. Now that we have a basic understanding of the communication between the hardware and the software, we will now continue our discussion on software developed for the creation and presentation of stimuli in cognitive sciences.

4. Software for experiment design in Cognitive Psychology

One of the greatest difficulties of a student who decides to work with the experimental method is learning how to control the stimuli and psychometric data-collection properly. Moreover, I have noticed that, in Brazil, most experiments are carried out on proprietary software that costs more than a thousand dollars and could just as easily have been carried out on free software. For this reason, I propose a change in the relationship of Brazilian psycholinguists with software for experimental design.

In this light, we have come across four basic problems that, though in no specific order, will be considered and discussed throughout this section:

(i) **Choice**: the variety of software available for experimental design;

(ii) **Familiarity** with the task: the lack of familiarity with more basic concepts on hardware-software interface and programming;

(iii) **Learning**: the learning curve; and

(iv) **Portability**: the software options can be limited according to the operating system.

As regards the choice, for some decades now, the options available to develop psychometric tests were scarce, forcing beginners to use the resources available in their laboratory. This facilitated the choice factor but affected the learning factor, given that we needed to become familiar with the available software. Today, we have seen a growth of software for experimental design, providing us with more and more options.

Today, it is usual for American and European labs to have three to four options available to facilitate the work of their researchers and visitors that may have experience in one or more of them. When choosing

a post-doc, it is also usual for laboratories to demand experience in one or two types of software, corresponding to those in which their researchers work with. This brings a uniformity in the way in which the research is conducted in the lab. On the other hand, what should be an advantage, in certain cases, ends up becoming a problem.

Today, dozens of software can be used to develop experiments. Some of these offer us the creative freedom of *Turing complete*[4] languages, including C, Presentation, Java, R, Python, and Matlab, together with its toolboxes. Others give us the advantage of the learning curve at the expense of the freedom of creation in proprietary software with a GUI, such as E-Prime, Paradigm, and SuperLab. Still others combine the GUI, the freedom of programming, as well as the portability between different operating systems, such as PsychoPy and OpenSesame, both free.

The most pressing question falls on the beginners. Still unexperienced and having to divide their attention among undergraduate studies, scientific training, and proposals for grants and for the M.A. program, they can present even more difficulties, both concerning the choice and the getting used to a software during their first experiments. At this first moment, even those who believe they have a good level of knowledge, will most likely opt for the most practical option, regardless of whether it is in fact more practical for him/her, or whether it is the best option for his experiment.

As seen above, programming languages allow us to create everything that our skill as a programmer allows. In this light, it is perfectly possible to use it to create any kind of experiment, from the simplest and recurrent to the more complex, with completely new methods. Even so, writing all of the necessary commands for communication between software and hardware, in addition to rewriting everyday functions, can hinder and prolong this work. To facilitate the task, many research groups with skills in computer programming have developed software to make life easier.

---

[4] According to Alan Turing, a computer programming language should contain (i) a means of repetition or of conditional jump and (ii) an end, allowing for the generation and reading of a result from the programmed algorithm. Upon attending to these conditions, the language is called *Turing Complete*. *Turing Complete* languages allow us to program everything that our programming skills allow.

The following subsections present and discuss some of these options. Technical comparisons, such as accuracy and precision in psychometric data-collection or their processing speed, however, are outside of the scope of this paper, especially since these measurements can change according to the hardware used. To guarantee that the times presented are not altered due to problems in the system and/or hardware, it is necessary to use external measures (PLANT *et al*., 2004; PLANT; TURNER, 2009). If these comparisons are interesting for you, I would suggest beginning by reading the battery of tests conducted by Garaizar *et al*. (2014), comparing the E-Prime 2, PsychoPy, and DMDX, or the article by De Leeuw & Motz (2015), which compares the response times of PsychToolbox 3 with those of jsPsych running in browsers and facilitating the viability of web-based experiments.

4.1 Experimental software options #1: *Toolboxes*

A toolbox is a group of functions written for specific purposes. With these tools, we do not need to code all communication between the hardware and the software. We simply need to use the existing functions and to define their parameters, which facilitates and automatizes the more common tasks, in our case, of the presentation of stimuli and of data collection. The graph below shows us some of the toolboxes used for experiments in cognitive sciences:

GRAPH 1 – Some toolbox options used for experimentation in cognitive sciences

For Python	
• *ExPyriment*	Krause & Lindermann (2014)
• *PyGame*	Shinners (2011)
• *Vision Egg*	Straw (2008)
For C	
• *PsyToolKit*	Stoet (2010)
For Java	
• *PsychJava*	*www.psychjava.com*[5]
For Matlab	
• *Psychtoolbox 3*	Kleiner *et al*. (2007)
For JavaScript (Web-based Experiments)	
• *jsPsych*	De Leeuw (2014)
• *JATOS*	Lange; Kühn; Filevich (2015)

ExPyriment (KRAUSE; LINDERMANN, 2014) was drawn up to be a universal platform. It has functions for behavioral and neurophysiological experiments, and it is multiplatform, thanks to Python. Its idea is to have a structured, linear logic that is easy to transpose the experimental design to the code. One of its advantages is the possibility of running on a version for Android. In its site, it is possible to find tutorials and demos to be studied and used in other experiments (see Annex).

The developers of *Vision EGG* (STRAW, 2008) were searching for a simpler way to use Python programming for graphic processing, especially for 3D graphics. Therefore, they use OpenGL API[6] and develop the toolbox as an interface for visual stimuli experiments.

[5] The PsychJava website has been offline for some time now. I was unable to verify the reason. As it was already incorporated into other software, I believe that the project has been discontinued.

[6] The term API (*Application Programming Interface*) refers to a group of algorithms created by the developer of a software to allow other types of software, or a code created by the user him/herself, to use or modify some hidden functions of the application in question.

PyGame (SHINNERS, 2011) has a different proposal. Originally created for gaming development, its own code is responsible for tasks that demand higher processing power, such as video and audio processing. These tasks work in a different way than the written code, in order to achieve the best performance possible. These characteristics made it a good tool for visual and auditory experiments.

One of the most widely used toolboxes today is *Psychtoolbox 3*, or PTB-3, (KLEINER *et al*., 2007), developed for Matlab and GNU Octave. The PTB-3's proposal is to provide functions that create an interface between Matlab and the hardware. This allows a better control, accuracy, and precision of visual and auditory stimuli, despite being a language that is more distant from the hardware (high-level abstraction). Thanks to these characteristics, I would suggest PTB-3 as a great tool to begin programming pshychological experiments. The PTB also has an interface with the graphic API *OpenGL*. PTB3 also have some functions written by the hardware manufacturers, such as the *EyeLink Toolbox*, provided by the SR Research for the development of tests in their eye-tracking equipment.

Although Matlab has versions in different platforms, it is quite likely that some codes need to be slightly modified so as to become compatible with a new operating system, such as keyboard mapping. Nonetheless, because it has been widely adopted, PRB-3 is constantly updated to correct bugs, to increase its functions, and to improve its performance and compatibility. Furthermore, the toolbox contains *PsychJava* that have not yet been launched on the market.

4.2 Experimental software options #3: Web-based experiments

If you have the need to conduct a massive (online) experiment or a web-based experiment, there is the possibility of using languages to develop webpages, such as HTML5, CSS, and JavaScript. Another option is Flash, which, for many technical, strategic, and market, reasons, has been rejected by the market.

These languages also have their toolboxes, facilitating the task of drawing up psychophysical tests on the web. One very recent toolbox is *jsPsych* (DE LEEUW, 2014), for *JavaScript*. jsPsych provides some demo codes that can be reused for other types of tests. Those that already work with JavaScript, CSS, and HTML5 will most likely find it easy to

develop experiments with jsPsych. Another tool for online studies also works on JavaScrip: *JATOS* (*Just Another Tool for Online Studies*), from Lange, Kühn & Filevich (2015).

There are also options in other languages. Developed for C, the *PsyToolKit* was created by Gijsbert Stoet for the creation and application of behavioral experiments. PTK is similar to an interpreted high level language for experimental purposes. It has a double compiler that transmutes the code to C during programming, and then the compiler transforms this into the machine's language. Since its 1.4 version, it has been possible to interpret it in the Java virtual machine (JVM). This toolkit also contains a web interface that allows one to create and run web-based experiments, in addition to a graphic interface for the creation and application of online questionnaires (*PsyQuest*). In its site, it is possible to find many tutorials and demos of the more popular experimental paradigms (see Annex).

Although some tests can be easily transferred to web platforms, I am still a bit skeptical concerning the uniformity of the algorithm between different hardware. Some visual and auditory stimuli can vary greatly according to the monitor, speakers, headphones, and hardware used, in such a way that each participant is stimulated by hardware of different quality and processing power. Different computers and browsers can have bugs or present the experiment and collect chronometric data in a different manner. In addition, we do not have a good control over the environmental conditions, such as the noise and lighting level of the room, or over who the participants are in cases in which this information is relevant to the interpretation of the data.

As regards the reaction times, De Leeuw & Moritz (2015) conducted a battery of tests, comparing the performance of jsPsych with that of PsychToolbox 3, and defend the use of JavaScript even for chronometric tests. By contrast, Reimers & Steward (2014) compared tests in JavaScript and in Flash. The authors argue that both can be useful tools for experiments. In recent years, however, Flash has been excluded from the web environment, which leads me to believe that, even though it is still useful, it is just a matter of time that the tests written in Flash will no longer be viable. Even so, Flash generates files that are quite heavy in relation to other software, which can compromise the performance in older and modest machines.

Another interesting option to conduct experiments is mobile software development, which has become an increasingly used tool. Experiments for cell phones or tablets can be developed directly in Java (Android) or Swift (iPad), or in some specific softwares, as we will see in the following sections. For iPad, there is still the option to develop it in *PsyPad*, created and maintained by Andrew Turpin (TURPIN; LAWSON; MCKENDRICK, 2014).

4.3 Experimental software options #3: Languages for cognitive experimental design

Toolboxes have facilitated the work of developing cognitive experiments in many programming languages. However, if the programmers create their own languages to facilitate their own tasks, such as R and Matlab, the researchers have also created languages that facilitate the development of experiments in cognitive sciences.

This is the case of *Psychology Experiment Building Language* (PEBL; MUELLER; PIPER, 2014), based on *C++*, which is free and is designed specifically for the development of experiments with text, image, audio, and video stimuli. This language is available for Windows and MacOS, and its use consists of the creation and edition of the demo text files that contain the necessary codes so that the parser – of the programming language – presents the stimuli and collects behavioral data (see Annex). Another free option is DMDX (FORSTER; FORSTER, 2003), which is commonly used for visual experiments.

Other softwares of this type were developed for commercial purposes and are, therefore, paid. One of the most used proprietary software in recent decades is *Presentation*, created by *Neurobehavioral Systems*. *Presentation* contains two proprietary languages: *Scenario Description Language* (SDL) and *Program Control Language* (PCL), based on C and Basic, both of which are used to draw up the visual stimuli, trials, and stimulation routine. Currently, *Presentation* counts on a module that allows for programming in Python.

4.4. Experimental software options #4: *Graphic User Interface* (GUI)

Despite the resources introduced by programming languages and toolboxes, all of this still involves programming, which, for some, is still considered a task for specialists. Beginners and professionals who are more experienced in cognitive sciences, who had no formal education in programming logic, show great resistance to the need to code their experiments. For them, some types of GUI software were developed, making the development process more visual, thus diminishing the need for programming skills, as well as diminishing the learning curve necessary to create the first tests.

One of the most famous types of GUI software is Psyscope (COHEN *et al*., 1993), which is widely used by psycholinguists. Psyscope contains a graphic interface, with *drag-and-drop* objects, which allow one to view and organize experiments in a visual logic of a tree diagram. The lines indicate the relations between functions, lists, and other objects, each with a realm of internal options that give us freedom for the configuration and personalization of our tests.

The current version of Psyscope support Tobii eye-tracking devices. Although it runs natively on Intel processors,[7] Psyscope is still exclusive to MacOS, which represents a disadvantage, especially as regards equipment prices. Nevertheless, one advantage is the fact that it is free and that it is possible to learn the system in only a few days. A new version, still in the beta phase, contains an interpreted code editor. This change should allow for the identification of errors through the code, in a much more simplified manner. To the researchers who wish to test the new version, one need to contact Luca Bonatti, one of the developers responsible for the forum (see Annex).

For Windows, the software that is the most like Psyscope is E-Prime, a proprietary software. E-Prime also contains a *drag-and-drop* interface in which it is possible to organize and view the experiment; however, its logic simulates a timeline, in which lists and functions succeed one another. Its 3.0 version was launched in December 2016 with

[7] One of the main reason for the incompatibility between the Mac and PC software was the use of different processors. Today, Apple computers also use Intel processors, which, for example, allow Windows to be installed on a Mac, as well as the so-called Hackintoshes, which consist of the installation of MacOS in PCs. For this reason, today, Psyscope could be ported to Windows.

the feature of running experiments on tablets. Due to the recent launching, the commentaries about E-Prime in this paper refer to version 2.

Another similar software is *Paradigm*,[8] which contains a similar logic and support Python scripting. It has the advantage of allowing the creation of experiments that can be saved in *DropBox* to be presented by an iPad version. Both E-Prime and Paradigm rely on support from manufacturers. The prices, however, are a great disadvantage.

Other software have been developed by the engineers of equipment manufacturers, in such a way as to guarantee an efficient software-hardware communication. This is the case of eye-trackers. To cite only the main manufacturers, the Tobii equipment can be used through the Tobii Studio, a software with a timeline logic that allows us to organize our visual stimuli. The SMI eye trackers have an entire suite to design, apply, and analyze the data. EyeLink eyetrackers have the *Experiment Builder* and a toolbox for Matlab (PTB-3) and Python.

The main advantage of GUI software is the learning curve. Generally, a beginner is able to learn it in just a few days. However, one disadvantage is the fact that they are highly focused on their main goals: building and running an experiment. Hence, though they do create many types of highly powerful and advanced algorithms, they usually do not allow us to go beyond the functions that have been pre-set by their developers.[9]

Some other kinds of software have been appearing on the market, offering a graphic interface that facilitates the visualization of the sequence of algorithms mixed with the potential of a high-level language. Fortunately, the two options that I know of are free and multiplatform:

---

[8] In the beginning of 2016, *Paradigm*'s sales were stopped due to the death of its only developer, Bruno Tagliaferri. The company was bought by Josh Pritchard at the end of the same year, resuming its sales and support.

[9] Excluding Psyscope, technically, E-Prime and Paradigm can be expanded through *InLine*. This tool allows one to insert coding from a specific language within the code generated by the GUI. *InLine* commands are the way we can access some of the software's hidden functions, aimed to extending the possibilities offered by the GUI. In this manner, the *InLine* language is not used to create a completely new code with functions that have not, in some way, been inserted by the software developers. When a language is inserted within other software for this purpose, they are known as *Script Languages*, which create *scripts*, which is different from the code that the software creates at the end of the development process and which contains these scripts.

PsychoPy and *Open Sesame*, both based on Python language and scripting, respectively.

PsychoPy[10] (PEIRCE, 2007, 2009) contains a graphic interface that allows one to draft and view the organization of a large part of its experiment. Broadly speaking, it contains two timeline windows, one of the experiment and another of each screen. This software runs on a pyglet backend, interfacing between Python and OpenGL API.

These characteristics brought PsychoPy users the dream of running it on RaspberryPi, a minicomputer developed by the RaspberryPi Foundation in the United Kingdom (Figure 3). These computers are extremely cheap. Its most powerful version (v. 3, Model B) costing less than 40 dollars and, its simplest version, (v. Zero) less than 20 dollars. Due to its price, these computers have become more popular in all types of projects that involve technology. However, due to the incompatibility between pyglet and the hardware, PsychoPy was incompatible with RaspberryPi. This panorama may soon change. Fortunately, last year, the first OpenGL experimental drivers for the platform were launched, making it possible for PsychoPy to be used in these small computers. According to tests performed by Mark Scase and published in the PsychoPy forum[11] in February 2016, the application of experiments is still unfeasible. But it is still possible to code experiments on RaspberryPi and apply them in another computer.

---

[10] PsychoPy is usually classified as a *toolbox*. I do not disagree; however, the fact that it contains a GUI makes it have more interested people within its non-specialist public than that of traditional toolboxes, and for this reason, I preferred to categorize it among the GUI software in this paper.

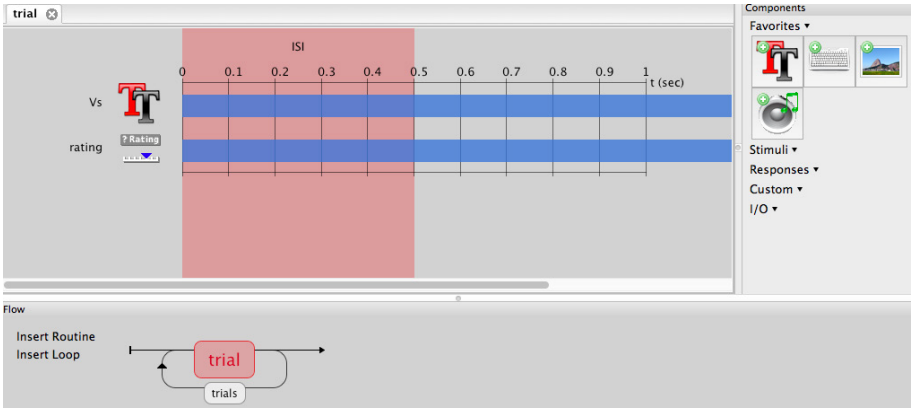
[11] "PsychoPy on RaspberryPi": https://groups.google.com/forum/ - !topic/psychopy-users/1mPwJqDVy1c

FIGURE 3 – A RaspberryPi 3 Model B, in a protective case (copyrighted photo)

Despite the fact that it is easy to learn and use the Builder View,[12] it still seems simpler to configure some variable within the Coder View. Moreover, some functions may just not be available in the GUI. For example, it does not contain a table editor in its interface, such as E-Prime and Psyscope, even though these editors are quite limited. In this sense, it is necessary to create our tables in an external software, such as Excel. This is generally a standard procedure for some programmers and simple for beginners, incurring no added difficulty. Still, among the users of GUI software, it is usually pointed out as one of its weak points.

[12] Excellent PsychoPy tutorial in Portuguese, reported by Prof. Mahayana Godoy (UFRN): <www.youtube.com/watch?v=W8cpnARvtNw>.

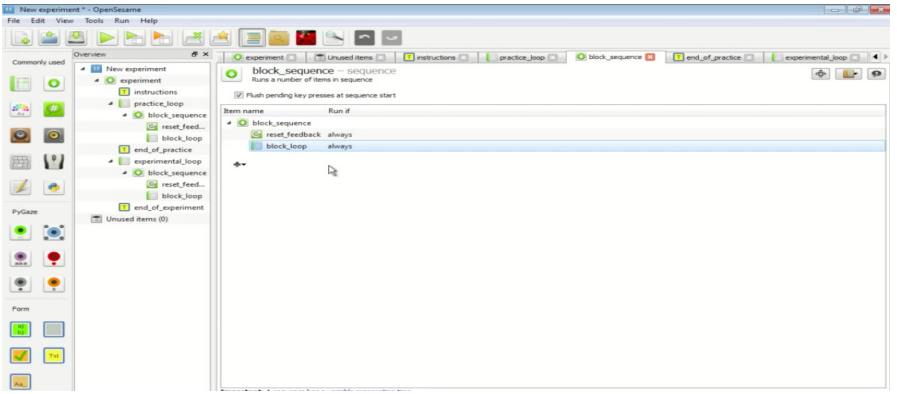
FIGURE 4 – Capture of the PsychoPy graphic interface



Note: In addition to the GUI interface, it also contains a Python programming console.

Open Sesame (MATHÔT *et al*., 2012), though it has a more complete interface than PsychoPy, still requires some hardware-software knowledge to be correctly used (or an attentive reading of their manuals and tutorials). For instance, we should be careful when inserting stimulus duration, since it depends on the screen refreshing rate - as we will discuss in the next section – or the choice of the backend, according to the type of experiment and of hardware we have [pyglet, pygame, xpyriment, or droid]. If pyglet is not necessary, Open Sesame can become a good tool to be used in RaspberryPi. This software can also be considered a free option to E-Prime, given that its interface bears some similarity to proprietary software.

FIGURE 5 – Capture of the Open Sesame graphic interface screen



5. How the software and hardware can influence perception

The previous section brought a realm of alternatives for the development of experiments. On the other hand, our attention should not merely be limited to the software. Much in the same way as Chomsky proposes the difference between competence and performance, separating what we know from what we in fact do in language, we can also transpose the dichotomy to the software-hardware interface. The software allows us to send a command so that the machine will perform a given task, but is the hardware capable of performing it?

5.1 Mental Chronometry and visual stimulation: the case of apparent movement

After efforts from Helmholtz in Physiological Psychology and from its revival by Donders and Cattell in Experimental Psychology, Mental Chronometry is recognized as a tool for the analysis and measurement of cognitive processes. Psycholinguistics commonly uses chronometric protocols in visual and auditory modalities, such as in lexical decision task, priming, self-paced reading, perceptions tests, among others. Many tests, however, depend on temporal accuracy on a millisecond scale, and to achieve this, it is necessary to have a notion how our equipment works, such as computer screens.

Before entering into the details of how computer screens work, we need to understand two visual illusions that were of utmost importance in the history of their development. The first is the *Phi Phenomenon* (WERTHEIMER, 1912), which occurs when we make many lamps available, one beside another, and turn them on and off successively. This action blocks the mind from perceiving the turning off and on of the lamps, creating the illusion that the light moves from one lamp to another.

The second is the *Beta Movement* illusion, described by Kenkel (1913). If we present a sequence of slightly similar images – such as a doll in different positions – at a specific speed, our mind don't perceive them as static images, but as a moving scene. These two phenomena are grouped in a kind of illusion known as *Apparent Movement*.

These illusions are responsible for our capability of watching the series of frames known as movies and cartoons and of playing videogames. Two questions were posed for the techniques of presenting images with apparent movement: (i) create materials with a larger number of images to result in a better experience or (ii) create materials that maintain the acceptable experience in the least expensive manner possible?

In silent films, the images were presented in a sequence of frames registered in celluloid films at 14 to 26 frames per second (fps), which was enough to give the illusion of movement. By contrast, this movement was normally considered to be irregular, giving the sensation of skipping. In this sense, it can be said that the *threshold* to the *beta movement* is approximately 15fps. Then, movies started to be created and presented at a higher rate, from 18 to 23 fps. Later, this rate was raised to a constant of 24 fps, given that this is the minimum rate for videos to be properly synchronized with sound (READ; MEYER, 2000).

5.2 Why use CRT screen?

Television was invented in the 1950s, and the movies were brought into the household. These devices were enormous and heavy due to the technology of the day. There are elements that emit radiation through the absorption of energy. This is the case of phosphorous, which is used both in fluorescent objects, which emit visible radiation while absorbing radiation from other sources, as well as in phosphorescent

objects, which continue to emit visible radiation for some time after the absorption. TV screens are phosphorescent and absorb the electrons emitted by a large Cathode Ray Tube (CRT), responsible for the size and weight of these devices.

The older computer screens follow the same technology. In CRT screens, each frame is constructed pixel per pixel in sequential form, beginning at the first point in the upper left corner to the last in the lower right corner of the screen, all in a matter of milliseconds. At this moment, the computer receives a signal from the screen, indicating that the current frame is over and the next frame begins to be constructed. This signal is called the *retrace signal* (COHEN; PROVOST, 1994). So as to prevent us from perceiving the change of the frames, the screen blinks for 1.5 milliseconds, while the rays that illuminate each pixel of the screen return to the upper left corner to begin the set-up of the next frame (PEIRCE, 2009). The frequency at which a screen is able to change from one frame to another has become known as the *refresh rate*. This term, in part, substituted the fps in the descriptions found in the manuals.

In TV devices, the refresh rate was defined according to the local electricity. In the US, it works at 60Hz, whereas the energy supply in Europe works at 50Hz. The refresh rate indicates how fast a device will be able to update the frame every second. Thus, the TV devices in Europe present a new frame every 20ms (1/50), while in the US, the televisions worked faster and were able to present a new frame at every 16.7ms (1/60). The computer screens, following the technology of televisions, generally work at 60Hz.

Though they consume a large amount of energy, CRT screens present an excellent response (in approximately milliseconds, μs), as well as an excellent angle of vision, which allows people in different positions to have a very similar psychophysical experience of the image. For this reason, many important centers of cognitive sciences resist recent technology and insist on presenting visual stimuli exclusively in CRT screens.

5.3 Are modern screens a good option?

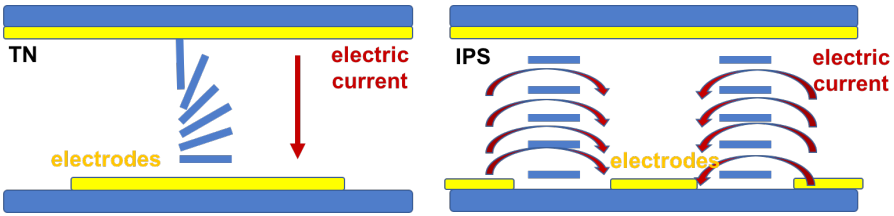
The technology of screens has evolved, based on the monochromatic screens used in clocks and some older laptops. We usually call this technology *Liquid Crystal Display* (LCD) or *Thin Film Transistor* (TFT).

Liquid crystal is a transparent substance but, upon receiving electric current, shows its structure and becomes opaque, blocking the passage of light. In TFT-LCD screens, the liquid crystal is spread between two transparent and polarized filters in opposite directions (HOOGBOOM *et al*., 2007). To form the image, the transistor emits an electric current capable of alternating the LCD configuration, making the molecules turn up to 90º vertically. For this reason, this technology is called *Twisted Nematic* (LCD-TN), due to the twisted arrangement of the liquid crystal molecules that are positioned perpendicularly to the screen (Figure 6). The crystal molecule movement guides the rays of light in the formation of light and colors, according to the image to be exhibited.

Some of the LCD-TN advantages are (i) the size of the screen, (ii) its response time (few milliseconds), and (iii) its low price today. By contrast, its viewing angle is quite restricted due to the angulation of the liquid crystal molecules. This results in a low fidelity of colors, brightness, and contrast of the image. LCD-TN monitors are not recommended for visual stimulation, because it is difficult to ensure that two participants will have the same psychophysical experience. In LCD-TN, brightness, colors, and contrast can be drastically altered, simply by making a subtle movement sideways.

When searching for solutions, the LCD-IPS (*In-Plane Switching*) technology was developed. This new method is able to make the liquid crystal molecules turn horizontally rather than vertically, in turn positioning themselves parallel to the screen. This change diminishes the distortion of the image, increases its viewing angle, and produces a great fidelity of color. Its weakness, however, is its response time, which is much slower than that of TN screens. Initially, this became one of the weakest points of the technology, creating the *Ghosting* effect. This problem has not been solved yet, but today, due to a lack of options, it is still the LCD screen most commonly used for experiments (Figure 6).

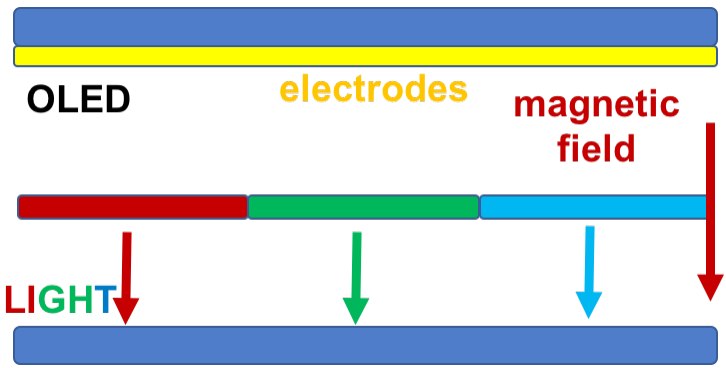
FIGURE 6 – Operational models of the LCD-TN and LCD-IPS monitors



Note: Adapted from Sharp manuals.

With the scarcity of CRT screens on the market, some studies discuss the potential of *Organic Light-Emitting Diode* (OLED) and Plasma Display screens to substitute them (ITO *et al*., 2013; RICHLAN *et al*., 2013). As regards OLED, these screens are constructed with two or three layers of carbon materials that emit light when exposed to an electromagnetic field (Figure 7). The first layer is responsible for conducting the electricity, while the final layer is responsible for emitting light. This light is produced by three filters, responsible for the RGB color system and the brightness of the light is proportional to the force of the magnetic field.

FIGURE 7 – Operational model of an OLED monitor



Note: Based on the Visionox model.

OLED screens contain a higher fidelity of gamma and colors, in addition to having a better brightness and an excellent field of vision, turning around 170-180 degrees, which avoids the common distortions of LCD screens. In addition, they present a better response time, placing it ahead of its competitors in terms of usability for visual experiments. Its great disadvantage is its price. Since it is a new technology, it is still quite expensive on the market. Cooper *et al*. (2013) have already pointed out OLED as excellent substitutes for the CRT screens.

5.4 Some questions concerning screen technology, the design and application of psycholinguistic tests

Currently, the majority of computer screens work at 60Hz (16.7ms/frame), although it is possible to find faster ones working with a refresh rate of up to 200Hz (5ms/frame), especially for gaming. Nevertheless, the experience of video also depends on the capacity of the hardware to process each image within this refresh rate, through a reasonable processor, a good and video card, as well as a reasonable free RAM. It is also important that the background tasks, such as antiviruses, software updates, and notifications, be turned off when running experiments. This will avoid unnecessary consumption of computer resources that can make the tasks run slower than usual, altering both the perception as well as the measurements.

If we wish to achieve a millisecond accuracy, it is of utmost importance that we have a notion of the competence and performance of the machine. To give an example of why, I will base my arguments on the experimental protocol of covert priming, in which the experimenter presents the prime word for only a few milliseconds in such a way that the word can be read, but it remains unconscious.

Consider an experiment of masked covert priming conducted by Garcia (2013), in which the experimenter wants to present the prime word for 38ms covered by a mask (a sequence of *), which is presented for 50ms before and after the prime word. Now consider that this test is applied in one of the most common screens, working at 60Hz (such as the MacBook White used in the study). Upon using a software with a graphic interface, such as Psyscope (used by the author) or Open Sesame, an experimenter will indicate the duration of the presentation of the stimulus in the corresponding field: 38ms. However, this simply

means that, at 38ms, the computer will send the command for the screen to refresh the image. In practice, the stimulus will only be truly changed in the next refreshing time, that is, in a multiple of 16.7ms (1s / 60Hz). In this scenario, we can see that the last refreshing before the command would have been around 33.4ms, the time of two frames. Hence, the next refreshing will be at 33.4ms + 16.7 = 50.1ms. That means that both the prime word (38ms) and its mask (50ms) would in fact be presented during the same time.

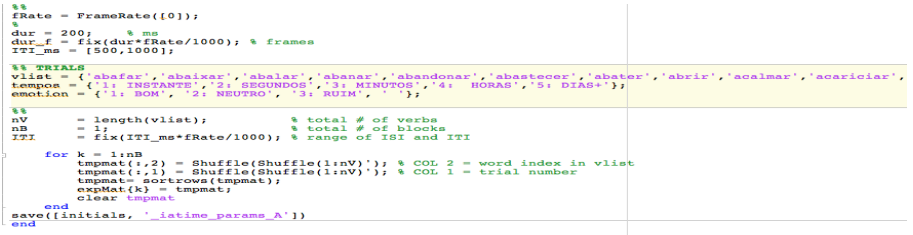
Initially, this is not necessarily an enormous problem for the majority of experiments, when we work with relatively long durations, such as 300 or 400ms. Nonetheless, it is important to beware of this problem, since, in the case of the covert priming and of other tests whose stimulus are presented for few frames, the refresh rate of the screen can represent the difference between the participant having a conscious perception or not. In addition, the researcher will present his/her methods, describing a 38ms prime word, which is not true. If the hardware and software configurations are not explicitly described in the methods, no one can identify the error.

The problem may be even greater. While many user manuals specify that their screens work at 60Hz, in its detailed technical specifications, it is not rare to find that some of them in fact work within a range of 60-75Hz. It is unclear, however, the moments in which the screen works within a specific band, if the refreshing has a variable timing, depending on the type of image, or if it has a configuration panel that allow us to control the refresh rate within this band. Many studies on methods of visual stimulus presentation have developed work on these questions. The majority agree that (i) when working with LCD screens, it is necessary to conduct precision tests and that (ii) relying only on the refreshing rate is not a reliable method (PLANT; TURNER, 2009; ELZE, 2010a, b; BAUER, 2015).

Concerning the frame rating control, programmers are aware of these questions, since knowing the screen refresh rate is essential for the code to work. This can be observed in the matlab/psychtoolbox code developed by Sampaio & van Wassenhove (2013), used and published by Sampaio (2015), and illustrated in Figure 8. In this code, there is a calculation referent to the number of frames presented by the hardware (variable "*dur_f*") in order, finally, to adapt the indicated time (variable "*dur*") and report the number of frames presented. This type of calculation

is usually called *adaptive synchronizer*. For GUI users, however, these details may never be noticed.

FIGURE 8 – Capture of the screen with part of the code written in Matlab by Sampaio & van Wassenhove (2013), using functions from PTB 3



```
%%
fRate = FrameRate([0]);
%
dur = 200;      % ms
dur_f = fix(dur*fRate/1000); % frames
ITI_ms = [500,1000];

%% TRIALS
vlist = {'abafar','abaixar','abalar','abanar','abandonar','abastecer','abater','abrir','acalmar','acariciar',
tempos = {'1: INSTANTE','2: SEGUNDOS','3: MINUTOS','4:  HORAS','5: DIAS+'};
emotion = {'1: BOM', '2: NEUTRO', '3: RUIM',' '};

%%
nV      = length(vlist);            % total # of verbs
nB      = 1;                        % total # of blocks
ITI     = fix(ITI_ms*fRate/1000); % range of ISI and ITI

    for k = 1:nB
        tmpmat(:,2) = Shuffle(Shuffle(1:nV)'); % COL 2 = word index in vlist
        tmpmat(:,1) = Shuffle(Shuffle(1:nV)'); % COL 1 = trial number
        tmpmat= sortrows(tmpmat);
        expMat{k} = tmpmat;
        clear tmpmat
    end
save([initials, '_iatime_params_A'])
end
```

PsychoPy also has an adaptive synchronizer using a function to test the refreshing rate of the screen and to calculate the duration of each frame, as illustrated in figure 9. Nevertheless, it is important to be aware since, in some cases, it is unable to identify the refresh rate and it will use the standard 60Hz. E-Prime 2 contains a diagnostic tool that also performs this synchronization and gives us information about the hardware's capacity (SCHNEIDER *et al*., 2002). The developers of Open Sesame, in tutorial videos,[13] call attention to this point by recommending the use of multiples of the fps rate decreased by 5ms, to avoid delays. By contrast, Paradigm and other types of software are able to report the number of frames and/or, their duration. However, it is unclear if they have a synchronizer.

FIGURE 9 –PsychoPy native adaptive synchronizer

```
# store frame rate of monitor if we can measure it
expInfo['frameRate'] = win.getActualFrameRate()
if expInfo['frameRate'] != None:
    frameDur = 1.0 / round(expInfo['frameRate'])
else:
    frameDur = 1.0 / 60.0  # could not measure, so guess
```

[13] www.youtube.com/ceebassmusic

Psyscope also does not make it clear if it has a synchronizer, but Cohen & Provost (1994, p. 446) indicate the existence of another control method, the *retrace synching*. Normally, the computer sends a new frame to the monitor according to the refreshing rate. The monitor, therefore, waits until the frame is over to refresh the image, as we saw above. With the *retrace synching*, PsyScope waits for the *retrace signal* to send the next frame, which guarantees that the indicated time in the results is the exact time of the onset of the stimulus. The exact duration of each frame and, consequently, of the stimulus, can be calculated with the information of the retrace synching.

All of these questions show us that, to develop psychophysical and psycholinguistic experiments, it is not enough to have knowledge of a specific software. It is also necessary to have a notion of what steps the hardware should follow and how accurately it is capable of executing these steps, in such a way that it becomes possible for us to think about how to overcome future problems or deviations in the precision and accuracy of the acquired data. GUI software are quite useful, as they simplify the task of developing an experiment. On the other hand, they allow us to run experiments without the need to understand what, in fact, is being done with the variables that we defined. This can lead us to believe that a visual or temporal variable is duly controlled when, actually, it is not.

After this discussion about tests running in a controlled machine, I wonder about the precision and accuracy of the data acquired by web-tools running in different settings.

6. Final Considerations

Upon completing this study, I believe I have achieved two main goals. The first of these is the discussion and presentation of the diverse types and options of software that can be used for experiments in cognitive sciences in general, which includes psycholinguistics. There is a large realm of software in different platforms and with different levels of learning curves that could be circulated more widely in Brazil, in turn increasing the contact of students with experimental design. The second objective is the discussion on problems of method that can be easily overcome if we have the knowledge of what happens in the machine when we run our test.

6.1 But which software should I use?

One of the main questions that can be made after this discussion is: "Which software should I use?" I believe that my contribution in this paper was that of presenting a wide range of options and their main characteristics, in such a way that you have some basis before choosing one of them. In a more practical manner, though previously DMDX, Presentation and Psyscope were some of the most widely used; today I can see that the most popular software among programmers are C and its toolboxes, PyGame (Python), and Psychtoolbox 3 (Matlab). Among non-programmers, E-Prime and PsychoPy appear, to me, to be the most popular in American and European laboratories. In Brazil, E-Prime has become quite popular in recent years among non-programmers, followed by Paradigm, due to its more accessible price. Among the open source options, I see few articles using PsychoPy and DMDX.

For those who are beginning their academic career, I believe it is quite reasonable to recommend PsychoPy. This recommendation is based on five factors: (i) it is an extremely simple software with a clean graphic interface, which contains only what is necessary; (ii) it offers the possibility of continuing to use the same software after learning Python, in its coder view; (iii) it has an adaptive synchronizer, offering a greater reliability regarding the acquired data; (iv) it is quite popular abroad, and it is possible to exchange experiences and codes with many researchers worldwide; and (v) it is free. It is important to note that this indication is merely a personal opinion of a software that I consider to be extremely practical and reliable for the great majority of cases.

Particularly, I have had great success in using PsychoPy in undergraduate psycholinguistics classes. This experience has helped students lose their fear of developing experiments due to their lack of knowledge in programming, to be able to apply and analyze their own tests in only a few classes, and, consequently, to have a more hands-on experience with what is in fact psycholinguistic experimentation, thus increasing their interest in the field. Moreover, the Psycholinguistics Laboratories in Brazil generally pay more than one thousand dollars for each license, which, with only a little more information, can be easily substituted by free software solutions. Though free, all of these types of software have discussion groups that work as collective support between users and responsible developers.

In addition to PsychoPy, the options in JavaScript appear to be excellent options for web-based experiments. Although I am still not comfortable conducting chronometric experiments in these platforms, the comparisons performed by De Leeuw & Moritz (2015) seem to be consistent. Nevertheless, I wonder if it is possible and how to avoid chronometric imprecision regardless of the processing differences of participant's computers. For this reason, though I recommend them, I do suggest being careful with these platforms if you are working with subtle differences in the physical stimulation, such as difference in images, in light, or color intensity, or with presentation times, such as in covert priming experiments.

6.2 Being aware of software and hardware configurations

My second goal was to show that it is necessary to have a comprehension of the software-hardware interface so that we can be sure that we correctly control the psycholinguistic stimulation. Only telling the computer what we want it to do does not necessarily mean it is capable of executing it. Without prior knowledge of the hardware's capability or without the use of accurate external measurements, it is impossible to perceive that the machine is not controlling the times as we had indicated.

Still in this scope, these questions show the importance of offering a detailed description of the software (such as its version), the code (to be shared), and the hardware used in the design and application of the tests. Many types of software may not have been tested in some versions of an operating system, especially those that have been recently released. For this reason, we are unable to update the operating systems of the laboratory computers before we are aware of its full compatibility with the software.

Some types of software present problems with some pieces of hardware, such as the video card, but we rarely pay attention to the warnings from the developers regarding these questions. Moreover, many times we describe that our stimuli are presented for a specific duration that is clearly impossible to be presented by an ordinary computer/screen. Since we do not have the obligation of knowing all of the incompatibility issues of our hardware and software, I cannot say it is wrong. For this reason, it is mandatory to present our methods and computers in such

a way that someone who has more knowledge on methods can easily identify inconsistencies between what has been described and what the computer can do. Being aware of these details prevents some of the main factors that lead to problems in the replication of results, a theme that has been in constant debate, such as in the survey of Open Science Collaboration (2015), published in *Science*, which later led to the article of "A manifesto for reproducible science" (MUNAFÒ *et al*., 2017), published in *Nature*.

I would like to reiterate that the detailed and careful description of the main information of our hardware, software, and the development of the stimuli, as well as sharing the codes, are essential factors for the viability of the experimental method, whose efficacy and validity is founded exactly upon the systematic reproduction of these methods and of their results reported by different researchers throughout the world.

I believe that this paper will, in some way, encourage researchers to be more aware and to take greater descriptive care when reporting their tests and results.

Acknowledgements

I would like to thank Virginie van Wassenhove, Douglas Bemis, Jansen Oliveira, Daniela Cid de Garcia, and Julia Cataldo Lopes for their discussions about programming languages, experimentation, and methods, as well as Renata Passetti who, only a few days before sending this article, inspired me to review the software for web-based experiments, making this study more inclusive. I would like to thank Leticia Kolberg and the students from the Psycholinguistics course in 2016/2, in the classes on methods, which helped me to raise some of the examples used throughout the text. Thanks also go to the reviewers of this article who, with their suggestions on its re-structuring, made, as much as possible, this work a bit more accessible. This review has been developed with the financial support of FAEPEX 519.292 project and FAPESP 2016/13.920-9 grant.

References

BAUER, B. A Timely Reminder About Stimulus Display Times and Other Presentation Parameters on CRTs and Newer Technologies. *Canadian Journal of Experimental Psychology*, Société Canadienne de Psychologie, v. 69, n. 3, p. 264-273, 2015. Doi: https://doi.org/10.1037/cep0000043

BEZANSON, J.; EDELMAN, A.; KARPINSKI, S.; SHAH, V.B. Julia: a fresh approach to numerical computing. *ArXiv*, 2014. Available at: <arxiv.org/abs/1411.1607>. Access: Nov. 28, 2016.

BRAINARD, D. H. The psychophysics toolbox. *Spatial Vision*, Brill Online, n. 10, p. 433-436, 1997.

COHEN, J.; MACWHINNEY, B.; FLATT, M.; PROVOST, J. PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods, Instruments & Computers*, Springer Link, v. 25, n. 2, p. 257-271, 1993.

COHEN, J.; PROVOST, J. *PsyScope: User Manual 1.0*, Carnegie Mellon University, 1994. Available at: <psy.cns.sissa.it/psy_cmu_edu/PsyMan.pdf>. Access: Nov. 28, 2016.

COOPER, E. A.; JIANG, H.; VILDAVSKI, V.; FARRELL, J. E.; NORCIA, A. M. Assessment of OLED displays for vision research. *Journal of Vision*, Association for Research in Vision and Ophthalmology, v.13, n. 16, p. 1-12, 2013.

DE LEEUW, J. R. jsPsych: A JavaScript library for creating behavioral experiments in a web browser. *Behavior Research Methods*, Springer, v. 47, n.1, 1-12, 2014. Doi: https://doi.org/10.3758/s13428-014-0458-y

DE LEEUW, J. R.; MOTZ, B. A. Psychophysics in a Web browser? Comparing response times collected with JavaScript and Psychophysics Toolbox in a visual search task. *Behavior Research Methods*, Springer, v. 48, n.1, p.1-12, 2015. Doi: https://doi.org/10.3758/s13428-015-0567-2

ELZE, T. Misspecifications of Stimulus Presentation Durations in Experimental Psychology: A Systematic Review of the Psychophysics Literature. *PLoS ONE,* São Francisco, Califórnia, v. 5, n. 9, 2010a.

ELZE, T. Achieving precise display timing in visual neuroscience experiments. *Journal of Neuroscience Methods*, Elsevier, n. 191, p. 171-179, 2010b.

FORSTER, K. I.; FORSTER, J. C. DMDX: A Windows display program with millisecond accuracy. *Behavioral Research Methods*, Springer, n. 35, p. 116-124, 2003.

GARAIZAR, P.; VADILLO, M.A.; LÓPEZ-DE-IPIÑA, D.; MATUTE, H. Measuring Software Timing Errors in the Presentation of Visual Stimuli in Cognitive Neuroscience Experiments. *PLoS ONE*, São Francisco, Califórnia, v. 9, n. 1, 2014.

GARCIA, D.C. *Elementos estruturais no acesso lexical:* o reconhecimento de palavras multimorfêmicas no português brasileiro. 2009. 108 f. Thesis (Masters in Linguistics) – Faculdade de Letras, Universidade Federal do Rio de Janeiro, 2009.

HOOGBOOM, J.; ELEMANS, J. A.W.; ROWAN, A. E.; RASING, T. H. M.; NOLTE, R. J. M. The development of self-assembled liquid crystal display alignment layers. *Philosofical Transactions of The Royal Society A*, The Royal Society Publishing, n. 365, p. 1553-1576, 2007.

ITO, H.; OGAWA, M.; SUNAGA, S. Evaluation of an organic light-emitting diode display for precise visual stimulation. *Journal of Vision,* Association for Research in Vision and Ophthalmology, v. 13*,* n. 7, p. 1-21, 2013.

KENKEL, F. Untersuchungen über den Zusammenhang zwischen Erscheinungs-grobe und Erscheinungsbewegung bei einigen sogenannten optischen Tauschungen. *2. Zeitschrifi fur Psychologie,* Göttingen, v. 67, p. 358-449, 1913.

KLEINER, M.; BRAINARD, D.; PELLI, D. What's new in Psychtoolbox-3? *Perception*, v. 36, n. 14, p. 1-26, 2007.

KRAUSE, F.; LINDERMANN, O. Expyriment: A Python library for cognitive and neuroscientific experiments. *Behavior Research Methods*, Springer, v. 46, n. 2, p. 416-428, 2014. Doi: https://doi.org/10.3758/s13428-013-0390-6

LANGE, K; KÜHN, S.; FILEVICH, E. Just Another Tool for Online Studies (JATOS): An easy solution for setup and management of web servers supporting online studies. *Plos One*, São Francisco, Califórnia, v. 7, n. 10, 2015.

MATHÔT, S.; SCHREIJ, D.; THEEUWES, J. OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavioral Research Methods*, Springer, v. 44, n. 2, p. 314-324, 2012. Doi: https://doi.org/10.3758/s13428-011-0168-7

MEDINA J. M.; WONG, W.; DÍAZ, J. A.; COLONIUS, H. Advances in Modern Mental Chronometry. *Frontiers in Human Neuroscience*, Frontiers, v. 9. 256, p. 5-7, 2015.

MOORE, G. E. Cramming more components onto integrated circuits. *Eletronics*, v. 38, n. 8, p. 114-117, 1965.

MUELLER, S.T.; PIPER, B.J. The Psychology Experiment Building Language (PEBL) and PEBL Test Battery. *Journal of Neuroscience Methods*, Elsevier, n. 222, p. 250-259, 2014.

MUNAFÒ, M. R.; NOSEK, B. A.; BISHOP, D. V. M.; BUTTON, K. S.; CHAMBERS, C. D.; DU SERT, N. P.; SIMONSOHN, U.; WAGENMAKERS, E. J.; WARE, J. J.; IOANNIDIS, J. P. A. A manifesto for reproducible science. *Nature Human Behaviour*, Springer Nature, n.1, 2017.

OPEN Science Collaboration. Estimating the reproducibility of psychological science. *Science*, American Association for the Advancement of Science, v. 349, n. 6251, 2015.

PEIRCE, J. W. PsychoPy - Psychophysics software in Python. *Journal of Neuroscience Methods*, Elsevier, v. 162, n. 1-2, p. 8-13, 2007. Doi: https://doi.org/10.1016/j.jneumeth.2006.11.017

PEIRCE, J.W. Generating stimuli for neuroscience using PsychoPy, *Frontiers in Neuroinformatics*, Frontiers, v. 2, 10, 2009.

PLANT, R. R.; HAMMOND, N.; TURNER, G. Self-validating presentation and response timing in cognitive paradigms: How and why? *Behavior Research Methods, Instruments, & Computers*, Springer Link, n. 36, p. 291-303, 2004.

PLANT, R. R.; TURNER, G. Millisecond precision psychological research in a world of commodity computers: New hardware, new problems? *Behavior Research Methods*, Springer, v. 41, n. 3, p. 598-614, 2009. Doi: https://doi.org/10.3758/BRM.41.3.598

READ, P.; MEYER, M. P. *Restoration of motion picture film*. Oxford: Butterworth-Heinemann, 2000. (Series in Conservation and Museology)

REIMERS, S.; STEWARD, N. Presentation and response timing accuracy in Adobe Flash and HTML5/JavaScript Web Experiments, *Behavior Research Methods*, Springer, v. 47, n. 1, p. 309-327, 2014.

RICHLAN, F.; GAGL, B.; SCHUSTER, S.; HAWELKA, S.; HUMENBERGER, J.; HUTZLER, F. A new high-speed visual stimulation method for gaze-contingent eye movement and brain activity studies. *Frontiers in Systems Neuroscience,* Frontiers, v. 7*,* n. 24, 2013.

SAMPAIO, T. O. M.; VAN WASSENHOVE, V. Self-paced Reading tests for GNU Octave/Matlab [software computacional], 2013. Access: March 30, 2017. Available at: <http://www.thiagomotta.net/uploads/7/0/5/2/7052840/spr_tests_-_octave-matlab_13.zip>. Access: Mar. 30, 2017.

SAMPAIO, T. O. M. *Coerção aspectual*: uma abordagem linguística da percepção do tempo. 2015. 398f. Dissertation (PhD in Linguistics) – Faculdade de Letras, Universidade Federal do Rio de Janeiro, 2015.

SCHNEIDER, W.; ESCHMAN, A.; ZUCCOLOTTO, A. E-Prime user's guide. Pittsburgh, PA: Psychology Software Tools, 2002. Available at: <step.psy.cmu.edu/materials/manuals/users.pdf>. Access: Nov. 28, 2016.

SHINNERS, P. *PyGame - Python Game Development* [computer software], 2011.

STOET, G. PsyToolkit - A software package for programming psychological experiments using Linux. *Behavior Research Methods,* Springer, v. 42, n. 4, p. 1096-1104, 2010. Doi: https://doi.org/10.3758/BRM.42.4.1096

STRAW, A. D. Vision Egg: An Open-Source Library for Realtime Visual Stimulus Generation. *Frontiers in Neuroinformatics*, Frontiers, v. 2, n. 4, 2008.

TURPIN, A.; LAWSON, D.J.; MCKENDRICK, A.M. PsyPad: a platform for visual psychophysics on the iPad. *Journal of Vision – Methods*, The Association for Research in Vision and Ophthalmology, v.14, n.16, 2014.

WATSON, A. B. *Handbook of Perception and Human Performance*. New York: Wiley, 1986.

WERTHEIMER, M. Experimentelle Studien über das Sehen von Bewegung. *Zeitschrift für Psychologie*, Göttingen, v. 61, n. 1, 161-265, 1912.

APPENDIX

Source of computer software cited in the text

a) Open source Multiplatform software:

C: www.open-std.org/jtc1/sc22/wg14

ExPyriment: www.expyriment.org

JATOS: www.jatos.org

Java: www.java.com

JsPsych: http://www.jspsych.org

Julia: julialang.org

Open Sesame: osdoc.cogsci.nl

Octave: www.gnu.org/software/octave

PEBL: pebl.sourceforge.net

PsyToolKit: www.psytoolkit.org

PsychJava: psychjava.com*

PsychoPy: psychopy.org

Psychtoolbox 3 (p/ Matlab e Octave): psychtoolbox.org

PsyPad: www.psypad.net.au

PyGame: pygame.org

Python: www.python.org

R-Project: www.r-project.org

Scilab: www.scilab.org

VisionEgg: visionegg.org

* The PsychJava website has been off-line for some time now. I was unable to discover the reasons why this site is offline. As it has already been incorporated in other types of software, I believe that the project has been discontinued.

b) Multiplatform Proprietary software:
Matlab: www.mathworks.com
SuperLab: www.cedrus.com/superlab

c) Open Source for MacOs X:
Psyscope: psy.ck.sissa.it

d) Proprietary software for Windows:
E-Prime: www.pstnet.com/eprime.cfm
Paradigm: paradigmexperiments.com
Presentation: www.neurobs.com

e) RaspberryPi: www.raspberrypi.org