Revista de Estudos da Linguagem, Belo Horizonte, v.25, n.3, p. 971-1010, 2017

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

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

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,1 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 the last few years, several softwares have been designed to help the development of experiments in cognitive sciences. The offer of user friendly software would help beginners in their first tests. However, it brought new problems and questions. Which software should one use? Which one is more adequate for my research and why? The present paper brings a quick and panoramic review on computer science, programming languages and on the presentation of visual stimuli. Through these three topics, I intend to promote a discussion (i) on the main types of software

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

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-method as independent variables in our experiments.

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

Recebido em: 5 de dezembro de 2016.

Aprovado em: 28 de março de 2017.

1. Introdução

O trabalho a ser apresentado a seguir não irá apresentar uma hipótese para testá-la através do método experimental. Este é um artigo de métodos que tem por objetivo problematizar algumas questões rotineiras de um cientista cognitivo que trabalha com experimentação: o desenvolvimento de experimentos no computador.

Primeiramente é preciso dizer que para elaborar um experimento no computador não é necessário ter um conhecimento muito avançado em computação. Existem diversos softwares especializados nesta tarefa tanto para usuários avançados quanto para os jovens cientistas em formação. Verba para software também não é necessariamente um problema. Para a maior parte dos experimentos, não precisamos desembolsar centenas de dólares para a aquisição de software proprietário, visto que existem diversas opções livres e com uma curva de aprendizagem bastante rápida. Ainda assim, por alguma razão, estes softwares não são tão difundidos entre os pesquisadores da área.

A segunda questão que trago neste artigo é que, apesar da grande oferta de aplicações especializadas, não é aconselhável deixar toda a tarefa de comunicação entre usuário e máquina por conta do software. Ao contrário dos sistemas operacionais mais atuais, os softwares de experimentação não escondem do usuário as configurações impossíveis de serem realizadas pelo seu hardware, nem dão informações precisas sobre o comportamento do hardware durante o teste. Muitas vezes acreditamos ter controle total sobre nossas variáveis sem nos darmos conta de que o computador não está executando exatamente a mesma tarefa que pedimos para ele executar.

Apresentadas minhas questões principais, pretendo iniciar nas próximas seções uma discussão sobre estes dois aspectos. A seção 2, a seguir, problematiza o conhecimento em computação do senso comum. Em seguida, a seção 3 visa esclarecer o que é e como funcionam as linguagens de programação. A seção 4 apresenta diversos tipos de software criados especialmente para experimentação em ciências cognitivas, entre linguagens de programação, toolboxes e softwares com interface gráfica pagos e gratuitos. A seção 5 discursa sobre problemas de controle de experimentos visuais originados do desconhecimento sobre o hardware usado em sua aplicação. Por fim, fecho este artigo com algumas considerações finais.

2. Uma rápida discussão sobre software e hardware

As Ciências da Computação possuem uma regra que parece prever o ritmo do avanço tecnológico: a Lei de Moore (1965). Esta lei demonstra que computadores aumentam exponencialmente sua complexidade, dobrando a capacidade de processamento a cada 2 anos. Se utilizarmos esta regra para olhar para o passado, verificamos que o início da computação teria ocorrido na década de 60, exatamente quando o 1º chip foi inventado.

A partir da lei de Moore, é possível prever que até mesmo usuários entusiastas não conseguem acompanhar o avanço da tecnologia em sua totalidade. Enquanto a quantidade de informação aumenta com o passar dos anos, nossas vidas adquirem mais responsabilidades que nos tomam o tempo necessário para nos atualizar em todas as frentes da tecnologia.

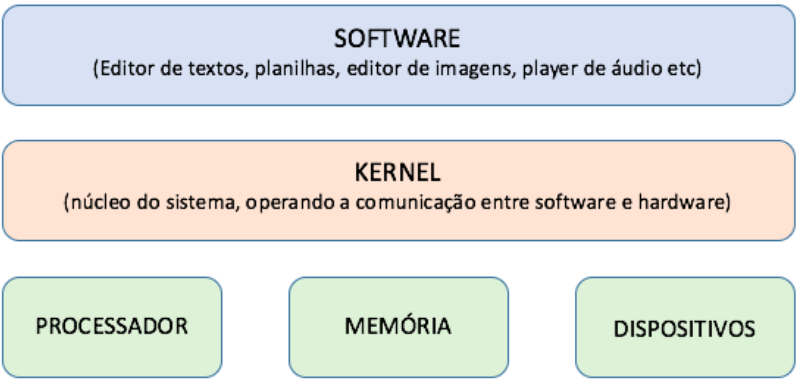
Neste contexto, somos invadidos pela ideia de que as crianças de hoje são nativas digitais e aprendem naturalmente a utilizar tecnologias avançadas que mesmo os entusiastas já não conseguem mais dominar. Porém, em diversas ocasiões, conversas pessoais e até mesmo em uma mesa durante o 3º Encontro de Divulgação Científica e Cultural na Unicamp, foi discutido e constatado que esta é uma meia verdade.

É inegável que existem (i) jovens usuários que realmente aproveitam a maior disponibilidade dos conteúdos digitais para dominarem tecnologias avançadas e se tornarem excelentes programadores. Por outro lado, o que observamos na maioria das vezes é que (ii) os softwares (especialmente os pagos) ficaram cada vez mais acessíveis tanto financeiramente, se tornando mais baratos, quanto na facilidade de utilização (*user friendly*). Isso resulta em uma ilusão de inclusão

tecnológica, na qual muitos usuários menos avançados conseguem, com relativa facilidade, realizar tarefas que seriam bastante complexas para o entusiasta de anos atrás.

Isso pode ser observado com mais clareza ao verificarmos a evolução dos sistemas operacionais dos dispositivos móveis. Cada vez mais populares, estes sistemas que, hoje, quase toda criança tem no bolso, chegaram ao ponto de dividir ou mesmo de substituir algumas funções que, antes, eram realizadas exclusivamente por computadores pessoais caros e inacessíveis. O resultado é que muitas daquelas pessoas que consideramos *experts* em tecnologia da informação possuem nada mais do que um vasto conhecimento sobre como utilizar os diversos softwares disponíveis no mercado. Estes usuários conseguem utilizar diversos tipos de programas computacionais de forma hábil através de sua interface de usuário, no *front-end* (Figura 1). Porém, eles muitas vezes possuem um conhecimento extremamente limitado sobre a comunicação do software com o hardware e da resolução de questões básicas de hardware, o *back-end*.

FIGURA 1 – Kernel (núcleo) do sistema: a ponte entre software e hardware



Nota: O centro de um sistema operacional é o seu núcleo (*kernel*), responsável por servir de ponte entre o software e o hardware de uma máquina. Usuários de interface (*front-end*) geralmente se limitam ao conhecimento do software, não tendo a necessidade de compreender o funcionamento da máquina nos demais níveis. Podemos fazer uma alusão à parte visível de um iceberg, quando a maior parte da rocha está submersa e fora de nosso campo de visão.

Não é difícil encontrar um entusiasta dos anos 90 ou do início do século que tenha lidado com as inúmeras questões de incompatibilidade entre uma nova peça de hardware (ex. placas de som e de vídeo) e a placa mãe de seu computador, ou entre um hardware compatível com sua placa mãe, mas sem a disponibilidade de um controlador para realizar a comunicação entre o sistema operacional e sua nova peça de hardware (*driver*[1]).

Quando instalamos o Windows ou o Linux pela primeira vez, eles geram drivers genéricos para que o hardware funcione em suas configurações mínimas. Os drivers específicos de cada peça são procurados após a instalação, para que possamos aproveitar o potencial máximo da máquina. Antigamente, tanto a busca quanto instalação dos drivers eram realizadas de forma manual e, ao passar por estas experiências e buscar por soluções, os usuários acabaram obtendo noções básicas sobre a comunicação entre hardware e software.

Hoje, tanto o hardware, quanto o software, assim como a comunicação entre eles, se tornaram mais eficientes permitindo que o sistema operacional simplesmente esconda muitas das opções de configuração que o hardware não suporta (muito disso devido à presença dos drivers corretos para o hardware instalado), o que evita uma grande parte dos problemas mais básicos que enfrentávamos na década de 90. Além disso, as últimas versões dos sistemas já possuem uma biblioteca dos drivers mais utilizados. Quando os sistemas não possuem o driver de sua peça, eles oferecem a opção de busca automática. Além disso, você sempre pode recuperar o driver na página da fabricante ou nos CDs que acompanham a peça.

Os aparelhos móveis também podem sofrer com a subutilização de suas funções por conta de drivers genéricos. Consideremos a título de exemplo a qualidade das fotos de um smartphone. Ao contrário do iOS, que roda em hardware padronizado e obriga os desenvolvedores de software a usarem os aplicativos e *drivers* oficiais da empresa, o Android precisa se adequar a diversos tipos de hardware de diferentes fabricantes. Por esta razão, cada fabricante possui um tipo de aplicativo de fotos otimizado para o seu aparelho, com diversos filtros que melhoram o desempenho da câmera no pós-processamento. Os aplicativos de terceiros, porém, para não terem o trabalho de criar um aplicativo diferente (ou *drivers*

[1] Software contendo instruções de comunicação entre o sistema operacional e o hardware.

diferentes que aumentariam o tamanho do aplicativo) para cada modelo existente de telefone, geram um driver genérico para acessar diretamente a câmera, o que diminui a qualidade de imagem mesmo nos aplicativos mais usados e famosos.

Conhecer a forma como o computador recebe os inputs pode não ser necessário para todos os usuários desde que tudo funcione aparentemente bem. Por outro lado, este conhecimento pode ser importante para fazermos melhores escolhas de softwares para determinados fins, para resolvermos determinados tipos de problemas técnicos e, especialmente, para quando desenvolvemos nossos experimentos e queremos controlar rigorosamente a forma como o hardware apresenta os estímulos aos participantes e coleta as respostas comportamentais.

O primeiro passo para discutir a comunicação entre software e hardware será o levantamento sobre o que são as linguagens de programação e como elas funcionam. Para isso, me limitarei àquelas que são comumente utilizadas por pesquisadores em ciências cognitivas, o que inclui a psicolinguística, nos principais laboratórios americanos e europeus.

3. O que são as Linguagens de Programação?

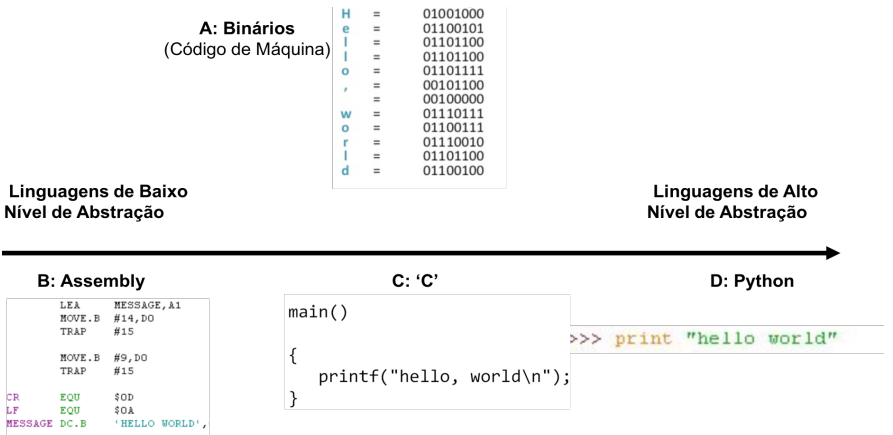
As máquinas clássicas[2] recebem input em forma de fluxos de corrente elétrica, correspondendo aos números 0 (desativado) e 1 (ativado), o que chamamos de *bits*. Isso nos dá a possibilidade de distinguir até duas informações por vez. Para aumentar o número de informações a serem processadas, os computadores foram desenvolvidos de modo a trabalhar com um conjunto de oito bits (1 *byte*). Agora, cada sequência de 8 bits pode ser relacionada a uma informação diferente. Por exemplo, a representação do número 8 corresponde à sequência 00111000 e a letra 'e' (minúsculo) corresponde à sequência 01100101. Este código binário, conhecido como código de máquina, é o único que os processadores são capazes de compreender. Porém, dar instruções ao

[2] Grosso modo, máquinas clássicas se opõem às máquinas quânticas. Repare que as máquinas clássicas funcionam com a transmissão de *bits*, que podem variar entre os estados desligado (0) ou ligado (1). As máquinas quânticas funcionam com os chamados *qubits*, ou *bits* quânticos, na qual a informação, além dos estados binários, pode estar em uma sobreposição de 0 e 1 (vetores). Isso terá efeitos na quantidade de informação transmitida por vez e na forma de transmissão e leitura de informação pelos computadores quânticos.

computador diretamente em binários é impraticável. Por isso, surgiram as linguagens de programação, doravante LP, visando agilizar a tarefa do desenvolvedor.

As LPs são classificadas de acordo com seu nível de abstração. As linguagens de mais baixo nível têm um funcionamento mais próximo do conjunto de instruções suportado pelos processadores das máquinas, o que exige uma maior curva de aprendizagem. As linguagens de mais alto nível são mais próximas da linguagem humana utilizando funções e relações sintáticas e semânticas de mais fácil memorização e aprendizagem (Figura 2).

FIGURA 2 – Linguagens de programação e nível de abstração



Nota: O primeiro passo de um curso de programação é escrever um código que faça a máquina exibir a frase "*Hello World*" no monitor. A Figura A corresponde apenas às letras do texto em binários; A Figura B utiliza um Assembly para realizar esta exibição; a Figura C apresenta o mesmo comando na linguagem C; A Figura D, executa a exibição do texto em Python. Repare que a facilidade de escrita do código aumenta de A até D. A seta maior exemplifica os conceitos de linguagens de baixo nível de abstração (mais próximo dos binários) e as de alto nível (mais próximos da linguagem humana). A forma como devemos escrever e organizar as funções e argumentos é chamada de sintaxe.

Como exemplo de linguagem de baixo nível é necessário citar o Assembly. O Assembly não é exatamente uma linguagem específica, mas o nome que se dá a uma linguagem única de cada processador,

contendo uma forma legível e memorizável do conjunto de instruções que a máquina pode realizar. Por esta razão o Assembly é também conhecido por 'linguagem de montagem'. Programadores mais avançados podem utilizar o Assembly para dar instruções para a máquina ou mesmo para desenvolver aplicações. O porém do desenvolvimento em Assembly é que eles são ininteligíveis as outras máquinas e seus códigos só podem ser executados pelo modelo de processador para o qual foram escritos. Além disso, a programação em Assembly ainda é uma tarefa para experts na área.

A programação só viria a se popularizar com o desenvolvimento de linguagens de alto nível de abstração. Por outro lado, assim como em linguagem natural, quando dois interlocutores não são fluentes em uma língua em comum, é necessário um tradutor/intérprete para que a comunicação se estabeleça. Isso gera um custo de processamento e, por consequência, um aumento no tempo de resposta da máquina.

Uma linguagem é um software que nos permite criar, de forma lógica, uma sequência de passos/funções chamada 'algoritmo'.[3] Este algoritmo será lido e executado pela máquina, esta, que só entende binários. Para que estas instruções sejam compreendidas pelo hardware, a linguagem precisa ser **'compilada'**, traduzida para a linguagem de máquina (ex. C e C++), ou ser **'interpretada'**, transformando suas linhas em um código binário (*byte code*) que será interpretado por uma máquina virtual (ex. Java e Python).

Uma das linguagens mais utilizadas hoje é o 'C'. Por ter sido criado com o objetivo de desenvolvimento de sistemas operacionais, softwares que precisam tirar todo o potencial das máquinas, o C é considerado a língua franca da programação, assim como o inglês entre as línguas naturais. As fabricantes de hardware, além do Assembly, geralmente escrevem também um código que mapeia as funções de seus dispositivos diretamente para o C, facilitando o trabalho dos desenvolvedores. Apesar da praticidade introduzida por esta linguagem, a programação ainda era relativamente restrita. Além disso, muitos usuários têm necessidades bastante específicas que poderiam ser realizadas de forma mais simples e em uma lógica de programação diferente da utilizada pelo C. E assim

[3] Algoritmo pode ser, grosso modo, definido como uma sequência ordenada de passos que levam a um determinado resultado, logo, se trata de uma sequência finita. Um algoritmo não precisa necessariamente ser algo matemático ou computacional.

surgiram diversas linguagens de mais alto nível, das quais podemos citar o Python, o R, o Matlab e o Java. A vantagem destas linguagens é que todas permitem que executemos cada passo do código ao longo da programação para verificar se ele irá funcionar corretamente na prática. Isso facilita (i) a identificação e a localização de erros, que normalmente são indicados pelo próprio console (tela de escrita do código) durante a programação, e (ii) a aprendizagem, devido ao seu *feedback* imediato e identificação dos argumentos sintáticos através de diferentes cores, como podemos observar no exemplo do Python, na Figura 2.

O Java é bastante utilizado pois, além de ser gratuito, teve uma grande ação de marketing realizada pela sua desenvolvedora Sun, hoje pertencente à Oracle. O mercado passou a exigir conhecimentos de Java para contratar programadores, o que levou as universidades a leciona-la nos cursos ligados à tecnologia. Embora tenha perdido força recentemente, o Java é a linguagem de programação adotada pela Google para desenvolvimento de aplicativos para Android. O Java é interpretado e traduz o código para uma máquina virtual (*Java Virtual Machine*, JVM). Esta máquina virtual é uma espécie de emulador que simula uma máquina específica em qualquer computador, evitando que o código tenha que ser recompilado.

O Python funciona de forma semelhante ao JAVA, utilizando uma máquina virtual que é instalada em qualquer computador e permite que ele seja executado em qualquer máquina, independente do sistema operacional. Isso favorece a portabilidade de seus códigos e o torna uma das linguagens preferidas de quem realiza experimentação.

O Matlab é um software proprietário com base em C e em Java. Ele foi criado com o objetivo facilitar a programação baseada em matrizes de dados. Embora os códigos escritos em Matlab rodem diretamente dentro do software, ele possui um compilador que nos permite tornar estes softwares independentes, capazes de rodar fora de sua interface. Uma opção gratuita ao Matlab é o GNU Octave. Sua sintaxe é bastante semelhante à do Matlab, o que facilita a migração. Outra opção com sintaxe semelhante é o Julia (BEZANSON *et al*., 2014) que, em algumas tarefas, apresenta um excelente desempenho.

Já o R é bastante conhecido de qualquer linguista e outros pesquisadores que trabalham com estatística, análise ou mineração de dados para a criação de corpus. Trata-se de um software livre desenvolvido para facilitar o trabalho com dados numéricos e estatísticos, sendo

amplamente utilizado em ciências cognitivas. Psicolinguistas costumam utilizar o R para análise de dados devido à sua lógica de programação e aos diversos códigos distribuídos gratuitamente para tal, mas nada impede que ele seja utilizado para desenho e aplicação de experimentos.

Em resumo, observamos até aqui que o processador possui uma arquitetura que recebe um determinado tipo de informação para executar um algoritmo. Essa informação pode ser elaborada através de uma linguagem de programação que, além de facilitar a tarefa do programador, também pode ser traduzida para a máquina através de compiladores e interpretadores, em troca de uma determinada perda no desempenho do algoritmo. Agora que temos um conhecimento básico sobre a comunicação entre hardware e software, seguimos com nossa discussão sobre os softwares desenvolvidos para a criação e apresentação de estímulos em ciências cognitivas.

4. Softwares para experimentação em ciências cognitivas

Uma das maiores dificuldades de um aluno que escolhe trabalhar com o método experimental é aprender como controlar devidamente a estimulação e a coleta dos dados. Além disso, reparo que, no Brasil, boa parte dos experimentos que são realizados em software proprietários de mais de mil dólares poderiam ser portados com facilidade para software livre. Por este motivo, proponho uma mudança na relação dos psicolinguistas com os softwares especializados em experimentação.

Neste caminho, nos deparamos com quatro problemas básicos que, embora sem uma ordem específica, serão considerados e discutidos ao longo desta seção:

(i) **Escolha**: a grande oferta de softwares de estimulação existentes hoje no mercado;

(ii) **Familiaridade** com a tarefa: a falta de familiaridade com conceitos mais básicos na relação entre hardware e software e com linguagem de programação;

(iii) **Aprendizagem**: a curva de aprendizagem em algumas plataformas de experimentação; e

(iv) **Portabilidade**: a diferença entre os sistemas operacionais que, frequentemente, limita a escolha de softwares.

No que diz respeito à escolha, há algumas décadas, as opções disponíveis para elaborar testes psicométricos eram escassas, obrigando os iniciantes a utilizar os recursos já disponíveis em seu laboratório. Isso facilitava o fator escolha mas afetava o fator aprendizagem, visto que precisamos nos acostumar com os softwares que eram disponibilizados. Hoje, vivenciamos um crescimento nas opções de softwares especializados em experimentação, nos trazendo cada vez mais opções de escolha.

Hoje é comum os laboratórios americanos e europeus disponibilizarem de três a quatro opções para facilitar o trabalho de integrantes e visitantes que, porventura, tenham experiência anterior em algum deles. Ao solicitar um posdoc, é também comum os laboratórios exigirem experiência em um ou dois softwares específicos, correspondentes àqueles nos quais seus integrantes mais trabalham. Isso traz uma uniformidade na forma como o laboratório desenvolve suas pesquisas. Por outro lado, o que deveria ser uma facilidade, em certos casos, se torna um problema.

Hoje existem dezenas de softwares que podem ser utilizados para a elaboração de experimentos. Alguns destes softwares nos oferecem desde a liberdade criativa das linguagens *Turing complete*[4] como C, Presentation, Java, R, Python e Matlab e suas toolboxes (caixa de ferramentas). Outros nos dão a facilidade de aprendizagem e de uso em detrimento da liberdade de criação nos softwares proprietários com interface gráfica (GUI, de *Graphic User Interface*), como o E-Prime, o Paradigm e o SuperLab. Outros ainda chegam a combinar a praticidade da GUI, a liberdade da programação e, também, a portabilidade entre sistemas operacionais, como o PsychoPy e o OpenSesame, ambos em software livre.

O maior questionamento na escolha de um software cai em cima dos iniciantes. Ainda inexperientes e tendo que dividir suas atenções entre graduação, iniciação científica e pré-projetos para obter bolsas e para ingressar em um mestrado, eles poderão apresentar ainda mais

[4] Mais tecnicamente, existem diversos tipos de linguagens. Segundo Alan Turing, uma linguagem de programação deve possuir (i) uma forma de repetição ou de salto condicional e (ii) um fim, possibilitando a geração e a leitura de um resultado do algoritmo programado. Ao atender estas condições, a linguagem é chamada de *Turing Complete.* Linguagens *turing complete* nos permitem programar tudo o que nossa habilidade nos permitir.

dificuldades tanto na escolha, quanto na curva de aprendizagem e na habituação a um software durante seus primeiros testes. Neste momento, mesmo aqueles que consideram ter um bom nível de conhecimento, muito provavelmente optarão pela opção mais prática, independente de ela ser, de fato, mais prática para ele, ou de ser a melhor opção para o tipo de teste que irá aplicar.

Como vimos anteriormente, as linguagens de programação nos permitem criar tudo o que nossa habilidade como programadores permitir. Desta forma, é perfeitamente possível utilizá-las para criar qualquer tipo de experimento, desde os mais simples e recorrentes até experimentos mais complexos e com métodos completamente inéditos. Ainda assim, escrever todos os comandos necessários para a comunicação entre software e hardware, além de reescrever funções corriqueiras podem dificultar e alongar este trabalho. Para facilitar a tarefa, diversos grupos de pesquisadores com habilidades em programação desenvolveram softwares que facilitam a vida do programador-experimentador.

Nas próximas subseções apresento e discuto algumas destas opções. Comparações técnicas entre cada uma delas, como exatidão temporal na coleta de dados ou velocidade de processamento, porém, estão fora do escopo deste trabalho, especialmente porque estas medições podem mudar de acordo com o hardware utilizado. Para garantir que os tempos apresentados não foram alterados por conta de problemas de sistema e/ou de hardware é necessário a utilização de medidores externos (PLANT *et al*., 2004; PLANT; TURNER, 2009). Caso estas comparações sejam interessantes para você, sugiro iniciar pela bateria de testes realizada por Garaizar *et al*. (2014), comparando os softwares E-Prime 2, PsychoPy e DMDX, ou o artigo de De Leeuw & Motz (2015) que comparam os tempos de resposta do PsychToolbox 3 com o do jsPsych rodando em navegadores e advogando pela viabilidade de experimentos cronométricos via internet.

4.1 Opções de software experimental #1: As *toolboxes*

Uma das opções que nos auxiliam no desenvolvimento dos experimentos são as *toolboxes*. Uma toolbox consiste em uma ‘caixa de ferramentas’, contendo um conjunto de funções previamente escritas para determinados fins. Com estas ferramentas, não precisamos realizar toda a comunicação entre hardware e software, bastando utilizar estas funções e definir seus parâmetros em nosso código, o que facilita e

automatiza as tarefas mais comuns, no nosso caso, tarefas de apresentação de estímulos e de coleta de dados. O quadro abaixo, nos mostra algumas das toolboxes utilizadas para experimentos em ciências cognitivas, o que inclui a psicolinguística e a neurociência da linguagem:

QUADRO 1 – Algumas opções de toolboxes utilizadas para experimentação em ciências cognitivas

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

O *ExPyriment* (KRAUSE; LINDERMANN, 2014) foi elaborado para ser uma plataforma universal de experimentação. Primeiramente por ter funções para experimentos comportamentais e neurofisiológicos, segundo por ser multiplataforma, graças ao Python. Sua ideia é ter uma lógica estruturada de forma a facilitar a transposição do desenho experimental para o seu código. Uma de suas vantagens é a possibilidade de rodar uma versão específica para *tablets* e *smartphones* android. Em seu site é possível encontrar tutoriais e uma série de códigos modelo para serem estudados e utilizados na programação de seu teste (ver Anexo).

[5] O site do PsychJava está fora do ar há algum tempo. Não consegui informações sobre os motivos da queda do site. Como ele já estava incorporado em outros softwares, acredito que o projeto esteja parado.

Os desenvolvedores do *Vision EGG* (STRAW, 2008) buscavam uma forma mais simples de utilizar a linguagem de programação multiplataforma Python para o processamento gráfico, especialmente em 3D. Eles então aproveitam o conjunto de bibliotecas de funções (API[6]) gráficas conhecido como *OpenGL* e desenvolvem o VisionEGG como uma interface entre os dois, visando a experimentação com estímulos visuais.

O *PyGame* (SHINNERS, 2011) tem uma proposta diferente. Originalmente criado para desenvolvimento de jogos, sua ideia é que o seu próprio código seja responsável por tarefas que demandam maior poder de processamento como a renderização de imagens e processamento de áudio. Estas tarefas serão abstraídas de forma diferente do código escrito pelos desenvolvedores, garantindo sempre o melhor desempenho possível para o jogo. Estas características o tornaram uma boa ferramenta para desenvolvimento de experimentação, sendo considerado uma boa plataforma para estimulação visual e auditiva, que demandam mais processamento da máquina.

Uma das toolboxes mais utilizadas atualmente é a *Psychtoolbox 3*, ou PTB-3, (KLEINER *et al*., 2007), desenvolvido para Matlab e GNU Octave. A proposta do PTB-3 é fornecer funções que façam interface entre o Matlab e o hardware para fins de experimentação. Isso permite um maior controle e confiabilidade cronométrica das estimulações visuais e auditivas apesar de ser uma linguagem mais distante do hardware (de alto nível de abstração). Estas características a tornam uma excelente ferramenta para programadores iniciantes. O PTB também tem interface com a API gráfica *OpenGL*, além de ter funções escritas por fabricantes de hardware como a *EyeLink Toolbox*, fornecida pela SR Research para desenvolvimento de testes em seus equipamentos de rastreamento ocular.

Embora o Matlab tenha versões nas diferentes plataformas, é bastante provável que alguns códigos precisem ser ligeiramente modificados para se tornar compatíveis com um novo sistema operacional

[6] O termo API (*Application Programming Interface*) se refere a um conjunto de algoritmos criados pelo desenvolvedor de um software para permitir que outros softwares, ou um código criado pelo próprio usuário, utilizem ou modifiquem algumas funções ocultas da aplicação em questão.

como, por exemplo, o mapeamento do teclado.[7] Apesar disso, devido à grande adoção, o PTB-3 é constantemente atualizado para corrigir bugs, para aumentar suas funções e para melhorar seu desempenho e compatibilidade. Mais além, a toolbox contém funções do *PsychJava* que ainda não possui distribuição pública.

4.2 Opções de software experimental #3: Opções de experimento via Web

Caso você tenha a necessidade de realizar um experimento em massa ou, por alguma outra razão, a experimentação via web seja interessante para você, existe a possibilidade de utilizar linguagens específicas para desenvolvimento de páginas web, como o HTML5, o CSS e o JavaScript. Outra opção é o Flash que, por diversas razões técnicas e, muito provavelmente também, estratégicas e de mercado, se tornou indesejável pelo mercado.

Estas linguagens também possuem suas toolboxes, facilitando bastante a tarefa de elaboração de testes psicofísicos na web. Uma toolbox bem recente é o *jsPsych* (DE LEEUW, 2014), para *JavaScript*. O jsPsych disponibiliza alguns modelos que podem ser reutilizados para outros tipos de testes, o que facilita bastante o seu uso. Aqueles que já trabalharam com JavaScript, CSS e HTML5, linguagens desenvolvidas para criação de páginas web, provavelmente terão facilidade em desenvolver experimentos com esta toolbox. Outra toolbox com o mesmo objetivo e utilizando a mesma linguagem é o *JATOS* (*Just Another Tool for Online Studies*), de Lange, Kühn & Filevich (2015).

Também temos opções em outras linguagens. Desenvolvido para C, o *PsyToolKit* foi criado por Gijsbert Stoet para criação e aplicação de experimentos comportamentais. A ideia deste *toolkit* é ser uma linguagem de nível de abstração maior que o do C, tendo um compilador duplo que transmuta o código para C durante a programação para, em seguida, o compilador do C transformá-lo em linguagem de máquina. Desde sua versão 1.4 é possível interpretá-lo na máquina virtual Java (JVM). Este *toolkit* também possui uma interface web que permite criar e rodar experimentos via web, além de uma interface gráfica para criação e aplicação de questionários online (*PsyQuest*). Em seu site é possível

[7] A mudança no código das teclas no PsychToolbox 3 pode ser observada nos códigos anexos de Sampaio (2015). Dos 4 experimentos desenvolvidos com a *toolbox*, 2 foram aplicados em PCs e 2 em Mac.

encontrar diversos tutoriais e modelos dos paradigmas mais populares em experimentação (ver Anexo).

Embora alguns testes possam ser facilmente portados para plataformas Web, ainda sou bastante cético no que diz respeito ao controle da estimulação. Alguns estímulos visuais e auditivos podem variar bastante de acordo com o monitor, caixas de som, fones e hardware utilizados. Máquinas e navegadores diferentes podem apresentar o experimento e coletar os dados cronométricos de forma diferente. Além disso, não temos um bom controle sobre as condições ambientes e sobre quem são os participantes do teste em casos em que estas informações sejam relevantes na interpretação dos dados.

No que diz respeito aos tempos de reação, De Leeuw & Moritz (2015) realizaram uma bateria de testes comparando o desempenho do jsPsych com o PsychToolbox 3 e advogam a favor da utilização de JavaScript inclusive para testes cronométricos. Já Reimers & Steward (2014) comparam testes em JavaScript em Flash. Os autores argumentam que ambos podem ser ferramentas úteis para experimentação psicofísica. Nos últimos anos, porém, o Flash vem sendo excluído de ambiente web, o que me faz acreditar que, mesmo que ainda seja uma ferramenta útil, é possível que, em breve, testes escritos em Flash deixem de ser viáveis. De qualquer forma, o Flash gera arquivos bastante pesados em relação aos outros softwares, o que pode comprometer o desempenho em máquinas mais antigas e menos potentes.

Outra opção interessante para realizar experimentos na web e em massa é o desenvolvimento de testes para tablets, que vêm se tornando uma ferramenta cada vez mais explorada. Experimentos para *tablets* podem ser desenvolvidos diretamente em Java (android) ou Swift (iPad), além de poderem ser desenvolvidos em outros softwares livres ou proprietários, como veremos nas próximas seções. Para iPad, ainda existe a opção de desenvolvê-lo no *PsyPad*, criado e mantido por Andrew Turpin (TURPIN; LAWSON; MCKENDRICK, 2014).

4.3 Opções de software experimental #3: Linguagens específicas para experimentos cognitivos

As toolboxes facilitaram muito o trabalho de desenvolvimento de experimentos cognitivos em diversas linguagens de programação. Porém, se os programadores criam linguagens próprias para facilitar suas próprias

tarefas, como o R e o Matlab, os pesquisadores em ciências cognitivas também criaram linguagens que facilitam a apresentação de estímulos.

Este é o caso do PEBL (*Psychology Experiment Building Language*; MUELLER; PIPER, 2014), baseado em *C++*, gratuito, desenhado especificamente para a elaboração de experimentos com estímulos de texto, imagens, áudios e vídeos. Esta linguagem está disponível para Windows e MacOS e sua utilização consiste na criação e na edição de arquivos de texto modelo que contém os códigos necessários para que o *parser* da linguagem de programação apresente os estímulos e colete os dados indicados pelo programador (ver Anexo). Outra opção gratuita é o DMDX (FORSTER; FORSTER, 2003), bastante utilizado para experimentos visuais.

Outros softwares deste tipo foram desenvolvidos por empresas e são, portanto, pagas. Um dos softwares proprietários mais utilizados nas últimas décadas é o *Presentation*, elaborado pela *Neurobehavioral Systems*. O Presentation contém duas linguagens proprietárias, a SDL (*Scenario Description Language*), e a PCL (*Program Control Language*), baseadas em C e em Basic, ambas utilizadas para elaborar os estímulos visuais, trials e o roteiro da estimulação. Atualmente, o *Presentation* conta com um módulo que permite a programação em Python.

4.4. Opções de software experimental #4: *Graphic User Interface* (GUI)

Apesar das facilidades introduzidas pelas linguagens de programação, pelas toolboxes e também pelas linguagens mais direcionadas à experimentação, tudo isso ainda envolve o ato de programar que, para alguns, ainda é considerado uma tarefa de especialistas. Iniciantes e profissionais mais experientes em ciências cognitivas, que não tiveram formação em lógica de programação, possuem uma enorme resistência à necessidade de programar seus experimentos. Para eles, foram elaborados alguns softwares que oferecem uma interface gráfica (GUI), que torna o processo de desenvolvimento mais visual, diminuem a necessidade de habilidades de programação e, assim, diminuem também a curva de aprendizagem necessária para criar seus primeiros testes.

Um dos softwares GUI mais famosos é o *Psyscope* (COHEN *et al*., 1993), bastante utilizado para experimentação em linguagem. O Psyscope possui uma interface gráfica, com objetos *drag-and-drop*, que permitem visualizar e organizar experimentos em uma lógica visual de

diagrama arbóreo. As linhas indicam as relações entre funções, listas e objetos do experimento, cada um com uma gama de opções internas que nos dão enorme liberdade de configuração e personalização do nosso teste.

A versão atual do Psyscope conta com suporte aos aparelhos de rastreamento ocular da Tobii. Embora rode nativamente em processadores Intel,[8] o Psyscope ainda é exclusivo do MacOS, o que se configura em uma desvantagem, especialmente no que diz respeito ao preço do equipamento. Apesar disso, ele tem como vantagem o fato de ser gratuito e de que é possível dominá-lo em questão de dias. Uma nova versão, ainda em fase beta, possui um editor de código interpretado. Esta mudança deve permitir a identificação de erros e a alteração de determinadas funções de forma muito mais simplificada, via código. Aos experimentadores que desejarem testar a nova versão, basta entrar em contato com Luca Bonatti, um dos desenvolvedores e responsáveis pelo fórum (ver Anexo).

Em ambiente Windows, um dos softwares mais próximos ao Psyscope é o E-Prime, de código proprietário. O E-Prime também possui uma interface *drag-and-drop* na qual é possível organizar e visualizar o experimento, porém, sua lógica simula uma linha de tempo, na qual listas e funções se sucedem. Sua versão 3.0 foi lançada em dezembro de 2016 com a possibilidade de desenhar experimentos para *tablets*. Devido ao lançamento recente, os comentários sobre o E-Prime neste artigo se referem à versão 2.

Outro software semelhante é o *Paradigm*.[9] Com lógica semelhante e baseado em Python. Ele tem a vantagem de possibilitar a criação de experimentos que podem ser salvos em *DropBox* para serem apresentados em iPads. Tanto o E-Prime quanto o Paradigm contam com suporte das fabricantes. Os preços, porém, são uma grande desvantagem.

---

[8] A maior razão da incompatibilidade de diversos softwares entre plataformas Mac e PC era a utilização de processadores diferentes. Hoje todas as máquinas da Apple utilizam processadores Intel, o que permite, por exemplo, que o Windows seja instalado em um Mac, a existência dos diversos emuladores de Windows no Mac e, também, dos chamados Hackintoshs, que consistem na instalação do MacOS em PCs. Por esta razão, hoje o Psyscope poderia ser portado para o Windows, o que ainda não foi feito pelos desenvolvedores.

[9] No início de 2016 o *Paradigm* teve suas vendas interrompidas por conta do falecimento de seu único desenvolvedor, Bruno Tagliaferri. A empresa foi comprada por Josh Pritchard no final do mesmo ano, retornando as vendas e o suporte.

Além destes, existem softwares que são desenvolvidos pelas próprias fabricantes dos equipamentos, de forma a certificar a comunicação eficiente entre software e o seu hardware. Este é o caso dos rastreadores oculares. Para citar apenas as maiores fabricantes, o equipamento da Tobii conta com o software Tobii Studio, uma ferramenta que segue a lógica da linha do tempo para organizar os estímulos visuais. O equipamento da SMI conta com toda uma suíte de aplicativos para desenhar, aplicar e analisar os dados. Já os rastreadores da EyeLink contam com o software *Experiment Builder*, além de toolboxes adicionais para o Psychtoolbox, do Matlab e para Python.

A grande vantagem dos softwares com interface gráfica está na curva de aprendizagem. Geralmente um iniciante consegue aprender a usar e montar seu experimento em poucos dias. Porém, uma desvantagem é o fato de serem muito focados no seu objetivo principal: realizar experimentação. Desta forma, embora criem diversos tipos de algoritmos bastante poderosos e avançados, eles são apenas softwares de experimentação, não nos permitindo ir muito além das funções já previstas pelos seus idealizadores.[10]

Alguns softwares que driblam esta questão vêm surgindo no mercado, oferecendo uma interface gráfica que facilita a visualização da sequência de algoritmos, mas, ao mesmo tempo, por serem baseados em linguagens de alto nível, conseguem mesclar sua interface gráfica com o potencial da programação. Felizmente, as duas opções que conheço são gratuitas e multiplataforma: o *PsychoPy* e o *Open Sesame*, ambos baseados em Python.

---

[10] Excluindo o Psyscope, tecnicamente E-Prime e Paradigm podem ser expandidos através das ferramentas chamadas "*InLine*". Esta ferramenta permite inserir pedaços de programação em outra linguagem dentro do código gerado pelo software GUI. Os comandos *InLine* são a forma como podemos acessar algumas funções escondidas dos softwares, tendo apenas o objetivo de extensão das possibilidades oferecidas na interface gráfica. Desta forma, a linguagem dos *InLine* não é utilizada para criar um código completamente novo com funções que já não foram, de alguma forma, inseridas pelos desenvolvedores do software. Quando uma linguagem é inserida dentro de outros softwares com esta finalidade, elas são conhecidas como *Linguagens de Script* e criam *scrips*, diferente do código que o software cria ao final do processo de desenvolvimento e que contém estes scripts.

O PsychoPy[11] (PEIRCE, 2007, 2009) possui uma interface gráfica que permite elaborar e visualizar a organização de uma grande parte do seu experimento. Grosso modo, ele possui duas janelas de linhas do tempo, uma do experimento como um todo e outra de cada estímulo a ser apresentado. Este software roda em uma base (*backend*) que faz interface entre o Python e o OpenGL (pyglet).

A facilidade de uso e o fato de rodar em Python trouxe aos usuários de PsychoPy o sonho de vê-lo rodando em um RaspberryPi, uma espécie de minicomputador desenvolvido pela Fundação RaspberryPi no Reino Unido (Figura 3). Estes computadores são extremamente baratos, custando menos de 40 dólares sua versão mais potente hoje (versão 3 Model B) e menos de 20 dólares em sua versão mais simples (versão Zero). Devido ao seu preço, estes computadores vêm se tornando popular em todo tipo de projeto que envolva recursos computacionais. Porém, devido a incompatibilidades entre software (pyglet) e hardware do RaspberryPi, o PsychoPy era incompatível com o RaspberryPi. Este panorama pode mudar em breve. Felizmente, foram lançados em no ano passado os primeiros drivers experimentais do OpenGL para a plataforma, possibilitando o uso do PsychoPy nestes pequenos computadores. Segundo testes realizados por Mark Scase e publicados no fórum do PsychoPy[12] em fevereiro de 2016, ainda é inviável aplicar experimentos. Mas ainda assim é possível criar códigos no RaspberryPi e aplicá-lo em máquinas com drivers mais funcionais.

[11] É comum que categorizem o PsychoPy como uma *toolbox* devido a algumas características. Não discordo, porém, o fato de ele possuir uma GUI faz com que ele tenha mais interessados entre os leigos do que as *toolboxes* tradicionais e, por isso, preferi categorizá-lo entre os softwares GUI.

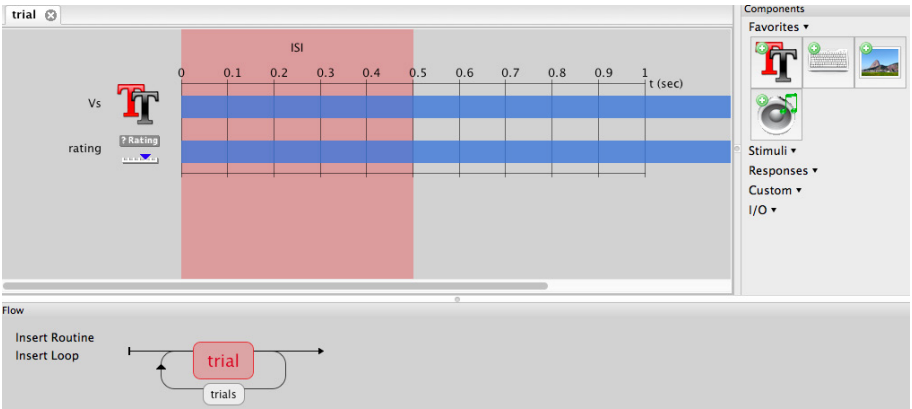
[12] "PsychoPy on RaspberryPi": https://groups.google.com/forum/ - !topic/psychopy-users/1mPwJqDVy1c

FIGURA 3 – Um RaspberryPi 3 Model B, em uma case de proteção (foto autoral).

Apesar da facilidade e de uma rápida curva de aprendizagem,[13] ainda me parece mais simples configurar as variáveis de algumas funções diretamente no código do PsychoPy. Outras, podem realmente não estar disponíveis na interface gráfica visto que o software privilegia uma interface simples com as funções mais comuns em experimentação psicofísica. Por exemplo, ele não possui um editor de tabelas em sua interface como o E-Prime e o Psyscope, por mais que estes editores sejam bastante limitados. Desta forma, é necessário organizar nossas tabelas em um software externo como o Excel. Geralmente isso é um procedimento padrão para alguns programadores e simples para quem está iniciando na área, não trazendo nenhuma dificuldade extra. Ainda assim, entre os utilizadores da interface gráfica, esta ausência é normalmente apontada como um de seus pontos fracos.

[13] Excelente tutorial do PsychoPy em Português gravado pela Prof. Mahayana Godoy (UFRN): <www.youtube.com/watch?v=W8cpnARvtNw>.

FIGURA 4 – Captura de tela da interface gráfica do PsychoPy

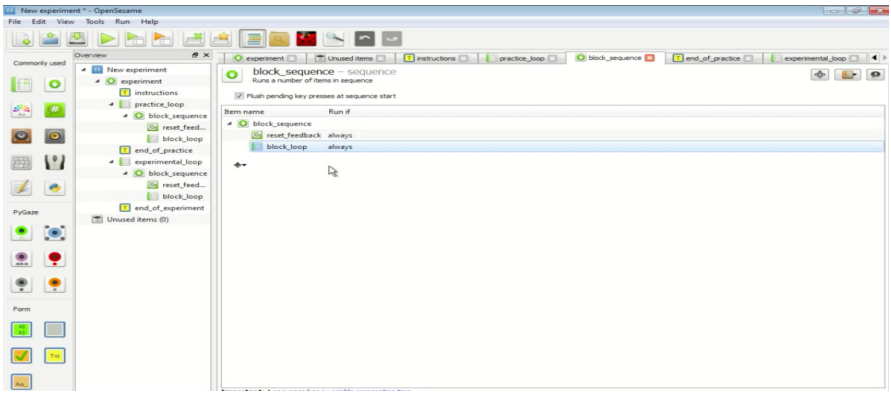


Nota: Além da Interface GUI ele conta com um console para programação em Python.

O Open Sesame[14] (MATHÔT *et al.*, 2012), embora tenha uma interface mais completa que a do PsychoPy, ainda requer algum conhecimento da relação hardware-software para que seja corretamente utilizado. Um exemplo é a decisão sobre o tempo dos estímulos visuais em comparação com a taxa de atualização de telas do monitor utilizado, como discutiremos com mais detalhes na próxima seção. Outra questão é a escolha da melhor base (*backend*) para ser utilizada pelo OpenSesame para o seu experimento, de acordo com o tipo de teste e de hardware [pyglet, pygame, xpyriment ou droid[15]]. Caso o pyglet não seja necessário, o Open Sesame pode se tornar uma boa ferramenta para ser utilizada nos RaspberryPi. Este software pode ser considerado também uma opção gratuita ao E-Prime visto que sua interface possui alguma semelhança com a do software proprietário.

[14] Na prática, o Python funciona como uma Linguagem de Script no Open Sesame, aparentemente de forma distinta do que acontece no PsychoPy. Ainda assim, o Python é mais fundamental no Open Sesame do que as linguagens de script no E-Prime ou no Paradigm e, por este motivo, o categorizamos entre as *toolboxes* e os softwares GUI.
[15] Para experimentos em tablets Android.

FIGURA 5 – Captura de tela da Interface gráfica do Open Sesame



5. Cuidados na relação software-hardware que influenciam a percepção

A seção anterior traz uma gama de alternativas de software para elaborar a estimulação. Por outro lado, nossos cuidados não podem se resumir a uma boa escolha de software. Assim como Chomsky propõe a diferença entre a competência e o desempenho, separando o que sabemos do que de fato fazemos em linguagem, podemos transpor a dicotomia para a relação software-hardware. O software nos permite enviar um comando para que a máquina faça uma determinada tarefa, mas será que o hardware é capaz de realizá-la?

5.1 Cronometria e estimulação visual: o caso do movimento aparente

Após os esforços de Helmholtz na Psicologia Fisiológica e de sua recuperação por Donders e Cattell na Psicologia Experimental, nossos métodos de coleta e de análise de tempos de resposta (Cronometria Mental) são reconhecidamente uma ferramenta de análise e de medida dos processos cognitivos. A psicolinguística se utiliza frequentemente de protocolos cronométricos em modalidades visual e auditiva como nos experimentos de decisão lexical, de priming, de leitura automonitorada, em testes de percepção entre outros. Muitos testes, porém, dependem de exatidão temporal na escala dos milissegundos e, para isso, é necessário ter uma noção do funcionamento de nossos equipamentos, como o monitor.

Antes de entrar nos detalhes do funcionamento dos monitores, precisamos entender duas ilusões visuais que foram de extrema importância na história do seu desenvolvimento. A primeira delas é o *Phi Phenomenon* (WERTHEIMER, 1912), que ocorre quando dispomos diversas lâmpadas uma ao lado da outra e as ligamos e desligamos de forma sucessiva. Esta ação impede que nossa mente perceba o desligar e ligar das lâmpadas, criando uma ilusão de que a luz se move de uma lâmpada para a outra.

A segunda ilusão é a *Beta Movement*, descrita por (KENKEL, 1913). Se apresentamos uma sequência de imagens ligeiramente semelhantes – como um boneco em diferentes posições – em uma determinada velocidade, nossa mente não consegue concebê-las como imagens estáticas, mas como uma mesma imagem cuja cena está em movimento. Estes dois fenômenos nos causam a ilusão conhecida como Movimento Aparente.

Estas ilusões são as responsáveis pela nossa capacidade de nos entreter com vídeo games, animações e filmes. Duas questões foram postas para as técnicas de apresentação de imagens com movimento aparente: (i) criar um material com maior número de imagens para resultar em uma melhor experiência ou (ii) criar um material que mantenha a experiência aceitável da forma mais barata possível?

Nos primeiros filmes mudos, as imagens eram apresentadas em uma sequência de frames registrados em películas de celuloide numa taxa que variava entre 14 e 26 quadros por segundo (fps, *frames per second*), que foi o suficiente para dar a ilusão de movimento. Por outro lado, este movimento era normalmente considerado irregular, dando a sensação de que as imagens tinham pequenos saltos no tempo (*skipping*). Desta forma, podemos dizer que o *threshould*[16] para o *beta movement* é de aproximadamente 15fps. Para resolver este problema, os filmes passaram a ser gravados e apresentados em uma taxa mais alta, variando entre 18 e 23fps, melhorando consideravelmente a experiência dos vídeos. Mais tarde, essa taxa passou para 24fps fixos, visto que esta é a taxa mínima

[16] Podemos traduzir *threshould* pelo termo "limiar". O limiar seria um 'nível' a partir do qual podemos observar uma mudança na percepção. Neste caso, até 14fps o sistema visual humano consegue distinguir as imagens de um filme. A partir de 15fps aproximadamente, esta percepção passará a ser de movimento, embora uma melhora na fluidez do movimento possa ainda ser observada com o aumento da frequência.

para que os vídeos pudessem ser corretamente sincronizados com o som (READ; MEYER, 2000).

5.2 Por que usar monitores CRT?

A televisão foi inventada na década de 50 e levou as imagens de filmes e programas televisivos para dentro das casas. Estes aparelhos eram enormes e pesados devido a sua tecnologia. Existem elementos que emitem radiações através da absorção de uma fonte de energia. Este é o caso do fósforo, que é utilizado tanto em objetos fluorescentes, que emite radiação visível enquanto absorve radiações de outras fontes, quanto nos fosforescentes, que continuam a emitir radiação visível algum tempo após a absorção. As telas das TVs são fosforescentes e absorvem os elétrons emitidos por um grande tubo de raios catódicos (*Cathode Ray Tube*, ou CRT), responsáveis pelo tamanho e peso destes aparelhos.

Os monitores mais antigos seguem a mesma tecnologia. Nos monitores CRT, cada quadro (imagem estática) é construído pixel por pixel de forma sequencial, partindo do primeiro ponto no canto superior esquerdo até o último no canto inferior direito da tela, tudo isso em poucos milissegundos. Neste momento, o computador recebe um sinal do monitor indicando que o quadro atual foi finalizado e começa a construção do próximo quadro. Este sinal é chamado de *retrace signal* (sinal de retorno; COHEN; PROVOST, 1994). Para não percebermos a mudança entre quadros, a tela pisca durante 1,5ms enquanto os raios que iluminam cada pixel da tela retornam ao canto superior esquerdo para iniciar a montagem do próximo quadro (PEIRCE, 2009). A frequência na qual um monitor consegue trocar de um quadro para o outro ficou conhecido como *refreshrate* (taxa de atualização). Este termo, em parte, substituiu o *fps* nas descrições de manuais.

Uma curiosidade dos aparelhos de TV é que o *refreshrate* era definido de acordo com a alternância de corrente elétrica local. Nos EUA, essa frequência é de 60Hz enquanto o fornecimento de energia na Europa funcionava a 50Hz. O *refreshrate* indica o quão rápido um aparelho consegue atualizar a imagem a cada segundo. Assim, os aparelhos de televisão na Europa poderiam apresentar uma imagem diferente a cada 20ms (1/50) enquanto, nos EUA, as televisões eram mais rápidas, podendo apresentar uma imagem a cada 16.7ms (1/60). Os monitores, seguindo a tecnologia das televisões, geralmente funcionam a 60Hz.

Embora consumam muita energia, os Monitores CRT possuem um excelente tempo de resposta aos comandos do computador (na casa dos microssegundos, μs), e excelente ângulo de visão, o que permite que pessoas em diferentes posições tenham uma experiência psicofísica da imagem muito semelhante. Por esta razão, diversos centros importantes de ciências cognitivas resistem às tecnologias recentes e insistem em apresentar estímulos visuais exclusivamente em monitores CRT.

5.3 Os monitores modernos são uma boa opção?

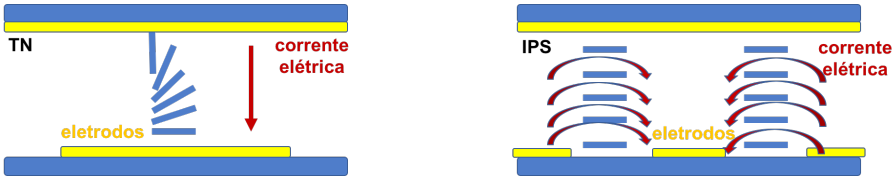
A tecnologia dos monitores mais modernos evoluiu a partir das telas monocromáticas utilizadas, por exemplo, em relógios e em alguns laptops antigos. Chamamos normalmente esta tecnologia de LCD (*Liquid Crystal Display*; Tela de Cristal Líquido) mas, o nome correto seria TFT (*Thin Film Transistor*, Transistor de Filme Fino). O LCD é apenas a forma como os primeiros monitores de tela fina funcionam.

O cristal líquido é uma substância transparente mas, ao receber a corrente elétrica, desmonta sua estrutura e se torna opaca, bloqueando passagem de luz. Nos monitores TFT-LCD, o cristal líquido é espalhado entre duas lâminas transparentes e polarizadas em sentidos opostos (HOOGBOOM *et al*., 2007). Para formar a imagem, o transistor emite uma corrente elétrica capaz de alterar a configuração do LCD, fazendo as moléculas girarem até 90º na vertical. Por este motivo os monitores que utilizam esta tecnologia são chamados *Twisted Nematic* (LCD-TN), devido ao arranjo torcido das moléculas de cristal líquido que se posicionam de forma perpendicular à tela (Figura 6). O movimento das moléculas de cristal guia os raios de luz na formação da luz e das cores, de acordo com a imagem a ser exibida.

Algumas das vantagens do LCD-TN foi ter diminuído o tamanho dos monitores, ter um tempo de resposta ainda razoável (poucos milissegundos) e ser extremamente barato hoje em dia. Por outro lado, seu ângulo de visão é bastante restrito devido à angulação das moléculas de cristal líquido. Isso resulta numa baixa fidelidade de cores, de brilho e de contraste da imagem exibida. Estas características fazem com que monitores LCD-TN não sejam recomendados para estimulação visual visto que é difícil que dois participantes tenham a mesma experiência psicofísica da imagem. Nos LCD-TN, brilho, cores e contraste se alteram drasticamente bastando um sutil movimento para o lado.

Na busca por uma solução para este ponto fraco, foi elaborada a tecnologia LCD-IPS (*In-Plane Switching*) que, através de um novo método, conseguiu com que as moléculas de cristal líquido girassem no sentido horizontal ao invés do vertical das telas TN, se posicionando de forma paralela à tela. Esta mudança diminui a distorção da imagem e aumenta seu ângulo de visão. Monitores IPS possuem excelentes ângulos de visão e fidelidade de cores. Seu ponto fraco, porém, está em seu tempo de resposta que é bem mais lento que o dos monitores TN. Inicialmente este se tornou um dos grandes pontos fracos da tecnologia, criando o efeito *Ghosting*, quando percebemos resquícios das imagens anteriores nas imagens correntes devido a baixa taxa de atualização. A tecnologia IPS ainda não resolveu este problema mas, hoje, por falta de opções, ainda é o monitor LCD, de preço acessível, mais recomendado para experimentação (Figura 6).

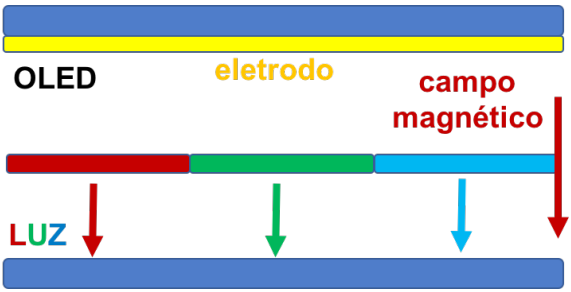
FIGURA 6 – Modelos de funcionamento de monitores LCD-TN e LCD-IPS



Nota: Adaptado dos manuais da Sharp.

Com a escassa oferta de monitores CRT no mercado, alguns trabalhos discutem a possibilidade de monitores OLED (*Organic Light-Emitting Diode*) e os displays de painel de plasma o substituírem (ITO *et al*., 2013; RICHLAN *et al*., 2013). No que se refere ao OLED, estes monitores são construídos com duas ou três camadas de materiais de carbono que emitem luz quando expostos a um campo eletromagnético (Figura 7). A primeira camada é responsável pela condução da energia elétrica enquanto a última é responsável pela emissão de luz. Esta luz é produzida por três lâminas, responsáveis pelas cores do sistema RGB (vermelho, verde e azul) e o brilho da luz é proporcional à força do campo magnético.

FIGURA 7 – Modelo de funcionamento de um monitor OLED



Nota: Baseado no modelo da Visionox.

Monitores OLED possuem gama e fidelidade de cores superiores, além de possuírem melhor brilho e um excelente campo de visão, girando em torno de 170-180 graus, o que evita as distorções comuns nos monitores LCD. Além disso, possuem um tempo de resposta também superior, o colocando à frente da concorrência em termos de usabilidade para a experimentação visual. Sua grande desvantagem é que se trata de uma tecnologia recente e, por isso, ainda possuem um alto custo no mercado. Cooper *et al*. (2013), em especial, já apontam os monitores OLED como ótimos substitutos aos CRT.

5.4 Questões sobre a tecnologia dos monitores, desenho e aplicação de testes psicolinguísticos

Atualmente, a grande maioria dos monitores funcionam a 60Hz, embora seja possível encontrar monitores de tecnologia mais recente com *refreshrates* de até 200Hz (5ms/quadro), especialmente para jogos. Ainda assim, a experiência de vídeo depende também da capacidade de o hardware processar cada imagem dentro desta taxa de atualização, através de um processador razoável, uma boa placa de vídeo, preferencialmente dedicada, e também uma quantidade razoável de memória RAM livre. Também é importante que, na hora da aplicação dos testes, sejam desabilitadas tarefas de fundo como antivírus, atualizações de software e notificações, a fim de evitar o consumo desnecessário de memória e de recursos do processador que podem deixar as tarefas mais lentas, alterando tanto a percepção quanto a medição dos dados.

Além disso, como indicado anteriormente, se precisamos de exatidão na casa dos poucos milissegundos, é de extrema importância termos noção sobre a competência e o desempenho da máquina. Para dar um exemplo do porquê, me basearei em um protocolo experimental de priming encoberto, no qual o experimentador apresenta a palavra prime por poucas dezenas de milissegundos a fim de que a palavra seja lida, mas dificilmente percebida pelo participante.

Considere um experimento de *priming* encoberto de Garcia (2013), em que a experimentadora quis apresentar a palavra *prime* por 38ms encoberta por uma máscara (uma sequência de *) que é apresentada por 50ms antes e após a palavra prime. Agora considere que este teste seja aplicado em um monitor de 60Hz. Ao utilizar um software com interface gráfica como o Psyscope (utilizado pela autora) ou o Open Sesame, um experimentador inexperiente irá indicar o tempo de apresentação desejado para o estímulo no campo correspondente: 38ms. Porém, isso quer dizer apenas que, aos 38ms, o computador enviará o comando para o monitor atualizar a imagem. Na prática, o estímulo só será de fato trocado no tempo de atualização da tela seguinte, ou seja, em um múltiplo de 16.7ms (1s / 60Hz). Neste cenário, podemos perceber que a última atualização do monitor teria sido por volta dos 33.4ms, o tempo de 2 quadros. Assim, a próxima atualização será aos 33.4ms + 16.7 = 50.1ms, o que indica que tanto a palavra prime (38ms) quanto a sua máscara (50ms) seriam, na verdade, apresentados com a mesma duração, de 50.1ms.

A princípio, isso não é necessariamente um problema para grande parte dos experimentos, quando trabalhamos com períodos relativamente longos como 300 ou 400ms. Ainda assim é importante estar atento a esta questão uma vez que, no caso do *priming* encoberto citado e de outros testes cujos estímulos sejam apresentados por poucos quadros, o tempo de uma atualização da tela pode ser a diferença entre o participante ter consciência do estímulo ou não. Além disso, o experimentador irá descrever o teste indicando que a palavra prime foi apresentada por um determinado período quando, por limitações de hardware, ela teria sido apresentada por um tempo consideravelmente maior. Caso as configurações de hardware e software não estejam explicitamente citadas na seção de métodos, não é possível ter certeza de que a descrição do teste corresponde, de fato, a sua realidade.

O problema pode ser ainda mais grave. Enquanto muitos guias de uso especificam que seus monitores funcionam a 60Hz, não é raro

encontrar em suas especificações técnicas detalhadas a informação de que eles funcionam em um range entre 60-75Hz. E estas especificações não deixam claro em que situações o monitor funciona em determinada faixa, se a atualização possui um timing variável a depender do tipo de imagem ou se ele permite configurações que nos possibilite controlar o *refreshrate* dentro desta faixa. Diversos trabalhos sobre métodos em estimulação visual se debruçam sobre este tema. A maioria concorda que, (i) ao se trabalhar com monitores LCD, é necessário realizar testes de precisão e que (ii) confiar apenas na taxa de atualização e na contagem de quadros não é um método confiável (PLANT; TURNER, 2009; ELZE, 2010a, b; BAUER, 2015).

Quem realmente programa está atento a estas questões pois saber a taxa de atualização do monitor é essencial para que seu código funcione. Isso pode ser observado no código matlab/psychtoolbox desenvolvido por Sampaio & van Wassenhove (2013), utilizado e publicado por Sampaio (2015) e ilustrado na figura 8. Neste código, existe um cálculo em cima do número de frames apresentado pelo hardware (variável "*dur_f*") para, enfim, adaptar o tempo indicado (variável "*dur*") e reportar o número de frames apresentados. Este tipo de cálculo é comumente chamado de sincronizador adaptativo (*adaptive synchronizer*). Para usuários de softwares GUI, porém, estes detalhes podem passar uma vida sem serem percebidos.

FIGURA 8 – Captura de tela com parte do código escrito em Matlab por Sampaio & van Wassenhove (2013), utilizando funções do 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
```

O PsychoPy também possui um sincronizador adaptativo utilizando uma função para testar a taxa de atualização do monitor e calcular a duração de cada frame, como ilustrado na figura 9. Ainda assim, é importante estar atento pois, em alguns casos, ele não consegue recuperar a taxa de atualização e usará 60Hz como padrão. O E-Prime 2, possui

uma ferramenta de diagnóstico que também realiza esta sincronização e nos dá informações sobre a capacidade do hardware (SCHNEIDER *et al*., 2002). Os desenvolvedores do Open Sesame, em seus tutoriais em vídeo,[17] chamam atenção para este ponto ao recomendar a utilização de múltiplos da taxa *fps* decrescidos de 5ms, para evitar possíveis atrasos no processamento. Já o Paradigm e outros softwares conseguem reportar o número de quadros apresentados e, por vezes, a sua duração. Porém não fica claro se eles possuem algum tipo de sincronizador.

FIGURA 9 – Sincronizador adaptativo nativo do PsychoPy

```
# 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
```

O Psyscope também não deixa claro se possui este sincronizador, mas Cohen & Provost (1994, p. 446) indicam a existência de outro método de controle, o *retrace synching*. Normalmente o computador envia a nova imagem para o monitor de acordo com a taxa de atualização. O monitor, então, aguarda até que o frame seja finalizado para atualizar a imagem, como vimos anteriormente. Com o *retrace synching*, o PsyScope aguarda o sinal de retorno do monitor (*retrace signal*) para enviar a imagem, o que garante que o tempo indicado nos resultados é o tempo exato do *onset* do estímulo. A duração exata dos frames e, por consequência, do estímulo, pode ser calculada a partir dos dados do *log* e do *retrace synching*.

Todas estas questões nos mostram que, para desenvolver testes psicofísicos e psicolinguísticos, não basta ter domínio sobre um determinado software. É necessário também ter alguma noção sobre que passos o hardware deverá seguir e o quão bem ele é capaz de executar estes passos, de forma que possamos pensar em como contornar eventuais problemas ou desvios de precisão e de exatidão os dados. Softwares com interface gráfica, são bastante úteis por simplificar a tarefa de elaborar um experimento. Por outo lado, eles nos permitem rodar experimentos sem a

[17] www.youtube.com/ceebassmusic

necessidade de entender o que, de fato, está sendo feito das variáveis que definimos para o teste. Isso pode nos fazer acreditar que uma determinada variável visual ou temporal está devidamente controlada quando, na verdade, não está.

Após esta discussão sobre testes rodando numa máquina controlada, me pergunto sobre a precisão e exatidão dos dados retornados em ferramentas web que não possuem máquinas, monitores e nem ambientes controlados.

6. Considerações Finais

Ao final deste trabalho, acredito ter atingido dois objetivos principais. O primeiro deles é a discussão e apresentação de diversos tipos e opções de software que podem ser utilizados para experimentação em ciências cognitivas em geral, o que inclui a psicolinguística. Existe uma enorme gama de softwares em diferentes plataformas e com diferentes níveis de curva de aprendizagem que poderiam ser muito mais difundidos no Brasil, aumentando o contato dos alunos de Linguística com a experimentação. O segundo objetivo é a discussão sobre eventuais problemas de método que podem ser facilmente contornáveis caso tenhamos conhecimento do que acontece na máquina quando estamos rodando nosso teste.

6.1 Mas qual software eu devo utilizar?

Uma das principais perguntas que poderá ser feita após esta discussão é: "qual software devo utilizar"? Acredito que minha contribuição neste artigo foi a de apresentar diversas opções e suas principais características, de forma que você tenha alguma base antes de escolher um deles. De uma forma mais prática, apesar de antigamente o DMDX, o Presentation e o Psyscope serem alguns dos mais utilizados, hoje percebo que os softwares mais populares são o C e suas toolboxes, o PyGame (Python) e o Psychtoolbox 3 (Matlab) entre aqueles que programam. Entre os que não programam, o E-Prime e o PsychoPy me parecem ser os mais populares nos laboratórios americanos, europeus. No Brasil, o E-Prime se tornou bastante popular nos últimos anos entre os não-programadores, seguido pelo Paradigm, devido ao seu preço mais acessível. Dentre as opções em software livre, vejo raros artigos utilizando o PsychoPy e o DMDX.

Para quem inicia sua vida acadêmica, acredito ser bastante razoável recomendar o PsychoPy. Esta recomendação se deve a 5 fatores: (i) se trata de um software extremamente simples com uma interface gráfica limpa, que contem apenas o necessário; (ii) te dá a possibilidade de continuar usando o mesmo software após uma iniciação à programação em Python, (iii), possui um sincronizador adaptativo, te dando maior confiança quanto aos dados obtidos; (iv) é bastante popular e você poderá trocar experiências e códigos com diversos pesquisadores do mundo que o utilizam, além de (v) se tratar de software livre. Vale ressaltar que esta indicação não passa de uma opinião pessoal de um software que considero ser extremamente prático e confiável para grande a maioria dos casos.

Particularmente, tenho uma ótima experiência ao utilizar o PsychoPy em aulas de psicolinguística para a graduação. Esta experiência faz com que os alunos percam o medo de elaborar experimentos por não saberem programar, consigam aplicar e analisar seus próprios testes em poucas aulas e, por consequência, tenham uma experiência mais real sobre o que é a experimentação psicolinguística, aumentando o interesse pela área. Além disso, os laboratórios de Psicolinguística no Brasil costumam pagar mais de mil dólares em cada licença que podem ser facilmente substituíveis por soluções em software livre, bastando um pouco mais de informação. Apesar de livres, todos estes softwares possuem grupos de discussão que funcionam como um suporte coletivo entre os usuários e os desenvolvedores responsáveis.

Além do PsychoPy, as opções em JavaScript parecem ser excelentes opções para coleta de dados via web. Embora eu ainda não me sinta a vontade para realizar coleta de tempos de resposta nestas plataformas, as comparações de De Leeuw & Moritz (2015) me pareceram consistentes. Me pergunto apenas se a precisão cronométrica se mantém independente da diferença de processamento das máquinas utilizadas e dos seus dispositivos. Por esta razão, apesar de recomendar, sugiro um certo cuidado com estas plataformas caso você trabalhe com diferenças sutis na estimulação física, como diferenças em imagens, em intensidade de luz e de cores ou com tempos de apresentação como nos testes de priming encoberto.

6.2 Atenção às configurações de software e de hardware

Meu segundo objetivo foi mostrar que é necessário ter uma compreensão mínima da interface software-hardware para que possamos ter certeza de que controlamos corretamente a estimulação psicolinguística. Apenas dizer ao computador o que queremos não significa que ele é capaz de realizar. Sem um conhecimento das capacidades do hardware ou sem a utilização de medidores externos acurados, é impossível perceber que a máquina não está controlando os tempos da forma que indicamos.

Ainda neste escopo, estas questões mostram a importância de descrevermos detalhadamente o software, código e hardware utilizados no desenho e aplicação dos testes. Muitos softwares podem não ter sido testados em uma determinada versão de um sistema operacional, especialmente os recém lançados. Por esta razão, não podemos atualizar os sistemas operacionais das plataformas de aplicação antes de termos certeza de sua total compatibilidade com os softwares, o que é diferente de o software simplesmente funcionar.

Alguns softwares podem apresentar problemas com determinadas peças de hardware como, por exemplo, uma placa de vídeo, mas raramente estamos atentos aos avisos dos desenvolvedores sobre estas questões. Além disso, muitas vezes indicamos em nossos testes que a apresentação dos estímulos foi realizada em um tempo que é notadamente impossível de ser apresentado em um monitor comum. Isso não é errado visto que não temos a obrigação de conhecer todos os detalhes, configurações e incompatibilidades de hardware existentes. E por isso é importante sinalizar estes detalhes em nossos métodos de forma que possíveis problemas possam ser facilmente identificados por aqueles que possuem um maior conhecimento no assunto. Esses cuidados evitam alguns dos fatores que levam ao problema da replicação dos resultados, tema que vem sendo bastante debatido como, por exemplo, no levantamento da Open Science Collaboration (2015) na Science que levou, posteriormente, à publicação de "*A manifesto for reproducible science*" (MUNAFÒ *et al*., 2017), na Nature.

Reforço que a divulgação detalhada e cuidadosa das principais informações sobre o hardware, o software, a elaboração, os métodos de elaboração, de aplicação e de análise de nossos experimentos, assim como a divulgação dos estímulos e dos códigos fonte, são fatores fundamentais para a viabilidade do método experimental, que tem sua eficácia e validade fundadas exatamente na reprodução sistemática destes

métodos e de seus resultados por diferentes pesquisadores em diversas partes do mundo.

Acredito que este artigo seja de alguma forma inspirador para que sejamos mais atentos e tenhamos um maior cuidado descritivo no momento de reportar nossos testes.

Agradecimentos

Agradeço a Virginie van Wassenhove, Douglas Bemis, Jansen Oliveira, Daniela Cid de Garcia e Julia Cataldo Lopes pelas discussões sobre linguagem de programação, experimentação e métodos. Renata Passetti que, há poucos dias do envio deste artigo, me inspirou a revisar as opções de experimentação via web, tornando este trabalho um pouco mais abrangente. Agradeço a Leticia Kolberg e aos alunos do curso Tópicos em Psicolinguística em 2016/2 que, nas aulas sobre métodos, me ajudaram a levantar alguns dos exemplos utilizados ao longo do texto. Agradeço aos revisores deste artigo que, com suas sugestões de reelaboração, tornaram este trabalho, na medida do possível, um pouco mais acessível. O presente artigo foi realizado sob a vigência do projeto FAEPEX 519.292 e no âmbito do auxílio FAPESP 2016/13.920-9.

Referências

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. https://doi.org/10.1037/cep0000043.

BEZANSON, J.; EDELMAN, A.; KARPINSKI, S.; SHAH, V.B. Julia: a fresh approach to numerical computing. *ArXiv*, 2014. Disponível em: <arxiv.org/abs/1411.1607>. Acesso em: 28 nov. 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. Disponível em: <psy.cns.sissa.it/psy_cmu_edu/PsyMan.pdf>. Acesso em: 28 nov. 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. 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. 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. Dissertação (Mestrado em Linguística) – 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. 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. 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, n. 256, p. 5-7, 2015. https://doi.org/10.3389/fnhum.2015.00256.

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. 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, n. 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. 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. Disponível em: <http://www.thiagomotta.net/uploads/7/0/5/2/7052840/spr_tests_-_octave-matlab_13.zip>. Acesso em: 30 mar. 2017.

SAMPAIO, T. O. M. *Coerção aspectual*: uma abordagem linguística da percepção do tempo. 2015. 398f. Tese (Doutorado em Linguística) – 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. Disponível em: <step.psy.cmu.edu/materials/manuals/users.pdf>. Acesso em: 28 nov. 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. 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. https://doi.org/10.3389/neuro.11.004.2008.

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.

ANEXO: Endereço das ferramentas computacionais citadas no texto

a) Multiplataforma Livre:

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

b) Multiplataforma Proprietário:

Matlab: www.mathworks.com
SuperLab: www.cedrus.com/superlab

c) MacOs X, Livre:

Psyscope: psy.ck.sissa.it

d) Windows, Proprietários:

E-Prime: www.pstnet.com/eprime.cfm
Paradigm: paradigmexperiments.com
Presentation: www.neurobs.com

e) RaspberryPi: www.raspberrypi.org

* O site do PsychJava está fora do ar há algum tempo. Não consegui informações sobre os motivos da queda do site. Como ele já estava incorporado em outros softwares, acredito que o projeto esteja parado.