Behav Res (2017) 49:1652–1667
DOI 10.3758/s13428-017-0900-z

Mousetrap: An integrated, open-source mouse-tracking package

Pascal J. Kieslich[1,2] · Felix Henninger[2,3,4]

Published online: 23 June 2017

Abstract Mouse-tracking – the analysis of mouse movements in computerized experiments – is becoming increasingly popular in the cognitive sciences. Mouse movements are taken as an indicator of commitment to or conflict between choice options during the decision process. Using mouse-tracking, researchers have gained insight into the temporal development of cognitive processes across a growing number of psychological domains. In the current article, we present software that offers easy and convenient means of recording and analyzing mouse movements in computerized laboratory experiments. In particular, we introduce and demonstrate the mousetrap plugin that adds mouse-tracking to OpenSesame, a popular general-purpose graphical experiment builder. By integrating with this existing experimental software, mousetrap allows for the creation of mouse-tracking studies through a graphical interface, without requiring programming skills. Thus, researchers can benefit from the core features of a validated software package and the many extensions available for it (e.g., the integration with auxiliary hardware such as eye-tracking, or the support of interactive experiments). In addition, the recorded data can be imported directly into the statistical programming language R using the mousetrap package, which greatly facilitates analysis. Mousetrap is cross-platform, open-source and available free of charge from https://github.com/pascalkieslich/mousetrap-os.

Keywords Mouse-tracking · Experimental design · Software · Response dynamics · Process tracing · OpenSesame · Python

Introduction

Mouse-tracking – the recording and analysis of mouse movements in computerized experiments – is becoming an increasingly popular method of studying the development of cognitive processes over time. In mouse-tracking experiments, participants typically choose between different response options represented by buttons on a screen, and the position of the mouse cursor is continuously recorded while participants move towards and finally settle on one of the alternatives (Freeman & Ambady, 2010). Based on the theoretical assumption that cognitive processing is continuously revealed in motor responses (Spivey & Dale, 2006), mouse movements are taken as indicators of commitment to or conflict between choice options during the decision process (Freeman, Dale, & Farmer, 2011).

Mouse-tracking was first introduced as a paradigm in the cognitive sciences by Spivey, Grosjean, and Knoblich (2005). In their study on language processing, participants received auditory instructions to click on one of two objects (e.g., "click the candle"). A picture of the target object was presented together with a picture of a distractor that was either phonologically similar (e.g., "candy") or dissimilar (e.g., "dice"). Participants' mouse movements were more curved towards the distractor if it was phonologically similar than if it was dissimilar, suggesting a parallel processing of auditory input that activated competing representations.

✉ Pascal J. Kieslich
kieslich@psychologie.uni-mannheim.de

1 Experimental Psychology, School of Social Sciences, University of Mannheim, Schloss Ehrenhof Ost, D-68131 Mannheim, Germany

2 Center for Doctoral Studies in Social and Behavioral Sciences, University of Mannheim, Mannheim, Germany

3 Cognitive Psychology Lab, University of Koblenz-Landau, Landau, Germany

4 Max Planck Institute for Research on Collective Goods, Bonn, Germany

Following Spivey et al. (2005), mouse-tracking has been used to gain insight into the temporal development of cognitive processes in a growing number of psychological domains, such as social cognition, decision making, and learning (for a review, see Freeman et al., 2011). More recently, researchers have extended the initial paradigm, combining mouse-tracking with more advanced methods. For example, mouse-tracking has been used in conjunction with eye-tracking to study the dynamic interplay of information acquisition and preference development in decision making under risk (Koop & Johnson, 2013). In an experiment with real-time interactions between participants, mouse-tracking uncovered different degrees of cognitive conflict associated with cooperating versus defecting in social dilemmas (Kieslich & Hilbig, 2014). As these examples show, an increasing number of researchers with different backgrounds and demands are using mouse-tracking to study cognitive processes. As a tool, mouse-tracking is increasingly combined with other methods to build complex paradigms and to integrate data across sources, leading to a richer understanding of cognition.

So far, many researchers conducting mouse-tracking studies have built their own experiments manually in code (e.g., Koop & Johnson, 2013; Scherbaum, Dshemuchadse, Fischer, & Goschke, 2010). These custom implementations were often one-off solutions tailored to a specific paradigm, and accompanied by custom analysis code to handle the resulting data specifically and exclusively. Researchers have spent considerable effort and technical expertise building these codebases.

As an alternative, other researchers have used *MouseTracker* (Freeman & Ambady, 2010), a stand-alone program for mouse-tracking data collection and analysis. Its ability to build simple experiments relatively quickly and design the mouse-tracking screen via a graphical user interface, as well as its integrated analysis tools have made mouse-tracking studies accessible to a broader range of researchers. However, researchers choosing MouseTracker lose the flexibility that general-purpose experimental software provides, in particular the ability to implement complex experimental designs within a single tool (involving, e.g., individually generated stimulus material, real-time communication between participants, and/or the inclusion of additional devices for data collection). In addition, many experimental software packages provide a graphical user interface not only for the design of single trials but of the entirety of the experimental procedure. Finally, most experimental software offers a scripting language so that its built-in features can be customized and extended. Although MouseTracker is free of charge (as citation-ware), the source code is not openly available and thereby not open to extensions and customization, limiting its features to those provided by the original authors. Moreover, MouseTracker is only available for the Windows operating system.

Going beyond custom implementations and stand-alone software solutions, there is a third option, namely providing modular components that extend existing experimental software. By building on the user-friendliness and flexibility of these existing tools, complex and highly customized experiments can be created easily, often without resorting to code. By using established open data formats for storage of mouse trajectories alongside all other data, preprocessing and statistical analyses are possible in common analysis frameworks such as R (R Core Team, 2016).

In this article, we present the free and open-source software *mousetrap* that offers users an easy and convenient way of recording mouse movements. Specifically, we introduce a plugin that adds mouse-tracking to *OpenSesame* (Mathôt, Schreij, & Theeuwes, 2012), a general-purpose graphical experiment builder. Together, these offer an intuitive, graphical user interface for creating mouse-tracking experiments that requires little to no further programming. Users can thus not only draw upon the extensive built-in functionality of OpenSesame for designing stimuli and controlling the experimental procedure, but also on additional plugins that extend it further, adding for example eye-tracking functionality (using PyGaze; Dalmaijer, Mathôt, & Van der Stigchel, 2014) and real-time interaction between participants (using Psynteract; Henninger, Kieslich, & Hilbig, in press). Yet further customization is possible through Python inline scripts. Like OpenSesame, mousetrap is available across all major platforms (Windows, Linux, and Mac).

In summary, mousetrap provides a flexible, extensible, open mouse-tracking implementation that integrates seamlessly with the graphical experiment builder OpenSesame and can be included by drag-and-drop in any kind of experiment. Its open data format allows users to analyze the data with a software of their choice. In particular, the recorded data can be imported directly into the statistical programming language R using the mousetrap package (Kieslich, Wulff, Henninger, Haslbeck, & Schulte-Mecklenbeck, 2016), which allows users to process, analyze, and visualize the collected mouse-tracking data.

In the following, we provide a brief introduction to mousetrap in combination with OpenSesame, and demonstrate how a mouse-tracking experiment can be created, what the resulting data look like, and how they can be processed and analyzed. In doing so, we create an experiment based on a classic mouse-tracking study by Dale, Kehoe, and Spivey (2007). In this study, participants' mouse movements are recorded while they classify exemplars (specifically: animals) into one of two categories; for example, a participant might be asked to classify a cat as mammal or reptile. The central independent variable in this paradigm is the typicality of the exemplar for its category: Exemplars are either typical members of their category, as above, or they are atypical (e.g., a whale), in that that they share both features with the correct (mammal) and a competing category (fish). The central hypothesis tested in this paradigm is that there should be more conflict between response options when classifying an atypical exemplar, and that mouse movements should therefore deviate more towards the competing category for atypical as compared to typical exemplars.

Building a mouse-tracking experiment

In the following, we provide a brief tutorial for building a mouse-tracking experiment with mousetrap, demonstrating the plugin's major features as we do so. Our final result will be a simplified version of Experiment 1 by Dale et al. (2007). This study incorporates many features of a typical mouse-tracking study: participants are presented with simple stimuli (here only a single word) in a forced-choice design with two response alternatives (one of which represents the correct response). Besides, a within-participants factor (typicality) is manipulated with a directed hypothesis regarding its influence on mouse movements.

Plugin installation and overview

Mousetrap depends on OpenSesame (version ≥ 3.1.0), which is available free of charge for all major operating systems from http://osdoc.cogsci.nl/, where it is also documented in depth. Mousetrap itself is available from GitHub (https://github.com/pascalkieslich/mousetrap-os), and is added to OpenSesame as a plugin.[1] The plugin includes built-in offline help and documentation for all features. Additional online resources are available from the GitHub repository, which offers extensive documentation and several example experiments, including the one built in the following (https://github.com/pascalkieslich/mousetrap-os#examples).

OpenSesame provides a graphical user interface through which users can create a wide range of experiments without programming. The building blocks of OpenSesame experiments are different *items*, from which an entire experiment can be assembled by drag-and-drop. For example, one might use a sketchpad item to present a visual stimulus, a keyboard_response or mouse_response item to record key presses or mouse clicks in response to the stimulus, and a logger item to write the collected data into a log file. Where desired, Python code can be included in an experiment using inline_script items to add further functionality. All of these items can be organized into sequences to run multiple items in direct succession and loops to repeat the same items multiple times (with variations). In a typical mouse-tracking experiment, a loop may contain the list of different stimuli that are presented in different trials, while a sequence contains all the items that are needed for each trial.

The items provided by the mousetrap plugin allow users to include mouse-tracking in any experiment using the same drag-and-drop operations and with the same ease. As OpenSesame provides two different ways of building displays, the mousetrap plugin contains two corresponding items: the mousetrap_response and the mousetrap_form item. Both provide comparable mouse-tracking functionality, but differ in the way the stimulus display is designed.

The *mousetrap_response* item tracks mouse movements while the stimulus display is provided by another item – typically by a sketchpad item that offers a graphical user interface for stimulus design. The mousetrap_response item then monitors the cursor position and registers button clicks.

In comparison, the *mousetrap_form* item extends the built-in OpenSesame form_base item to provide both a visual display as well as mouse-tracking. The visual content (e.g. text, images, and buttons) can be specified directly from within the item using a simple syntax and positioned on a user-defined grid.

Both the mousetrap_response and the mousetrap_form can be used without writing Python code. For even more flexibility, both items provide corresponding Python classes which can be accessed directly from code. Examples as well as documentation for these are provided online.

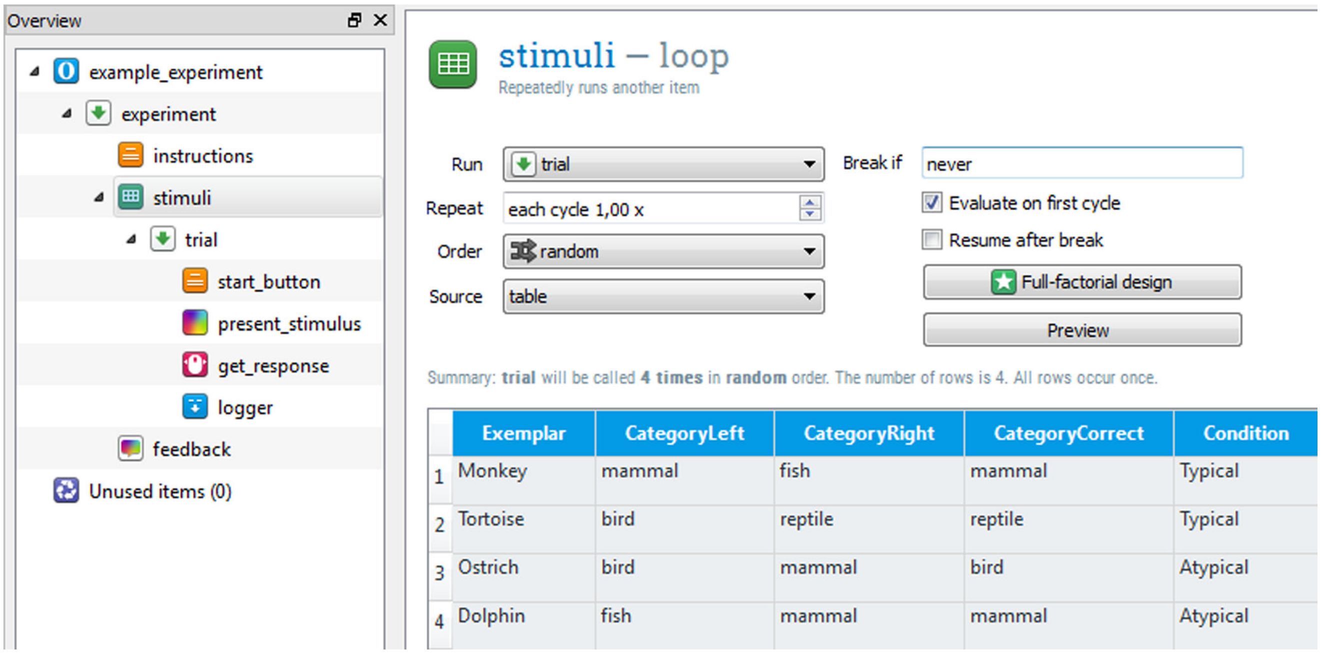
Creating a mouse-tracking trial

Figure 1 shows the structure of our example experiment. In the beginning of the experiment, a *form_text_display* item labelled "instructions" is included to explain the task to participants. Next, a *loop* item called "stimuli" is added, which repeats the same sequence of items in each trial while varying the exemplars and response categories in random order (this data, along with additional metadata, is entered in the loop in tabular format – see bottom right of Fig. 1, where each row corresponds to one stimulus and the associated response options).

A simple way to create a mouse-tracking trial via the graphical user interface is to use a sketchpad item to create the visual stimulus display and a subsequent mousetrap_response item to track the mouse movements while the sketchpad is presented. Before creating the individual items, the overall experiment resolution should be set to match the resolution that will be used during data collection, because sketchpad items run at a fixed resolution and do not scale with the display size. As mouse-tracking experiments are normally run in full-screen mode, the experiment resolution will typically correspond to the display resolution of the computers on which the experiment will be conducted.

The trial *sequence* itself begins with a *form_text_display* item that contains a start button in the lower part of the screen, as is typical for mouse-tracking experiments (Freeman & Ambady, 2010). Participants start the stimulus presentation by clicking on this button, which also ensures that the start position of the cursor is comparable across trials. Using a form_text_display item is the most basic way of implementing

[1] Information on installing the plugin is provided at https://github.com/pascalkieslich/mousetrap-os#installation



Run: trial
Repeat: each cycle 1,00 x
Order: random
Source: table
Break if: never
☑ Evaluate on first cycle
☐ Resume after break

Summary: **trial** will be called **4 times** in **random** order. The number of rows is 4. All rows occur once.

	Exemplar	CategoryLeft	CategoryRight	CategoryCorrect	Condition
1	Monkey	mammal	fish	mammal	Typical
2	Tortoise	bird	reptile	reptile	Typical
3	Ostrich	bird	mammal	bird	Atypical
4	Dolphin	fish	mammal	mammal	Atypical

Fig. 1 Structure of the example OpenSesame experiment (left) and settings for the stimuli loop (right). The panel on the left provides an overview of all items in the experiment, organized in a sequential (from top to bottom) and hierarchical (from left to right) display. On the highest (i.e., leftmost) level, the experiment sequence contains the instructions, the stimuli loop that generates the individual trials, and a final feedback screen. The loop contains a trial sequence, which is subdivided into the start button screen, a sketchpad that presents the stimulus, a mousetrap_ response item that collects the participant's response and tracks cursor movements, and a logger item to save the data into the logfile. On the right, the details of the loop are visible. The design options at the top configure the loop such that each stimulus is presented once in random order, and the table at the bottom contains the actual stimulus data for four trials, namely the exemplar and response categories to be shown on screen, the correct response, and the experimental condition for inclusion in the dataset

a start screen because it provides a ready-made layout including some adaptable instruction text and a centered button which can be used to start the trial. Further customization of the start screen is possible, for example, by instead using an additional sketchpad – mousetrap_response combination (as was done in the experiment reported below; see also the online example experiment without forms).

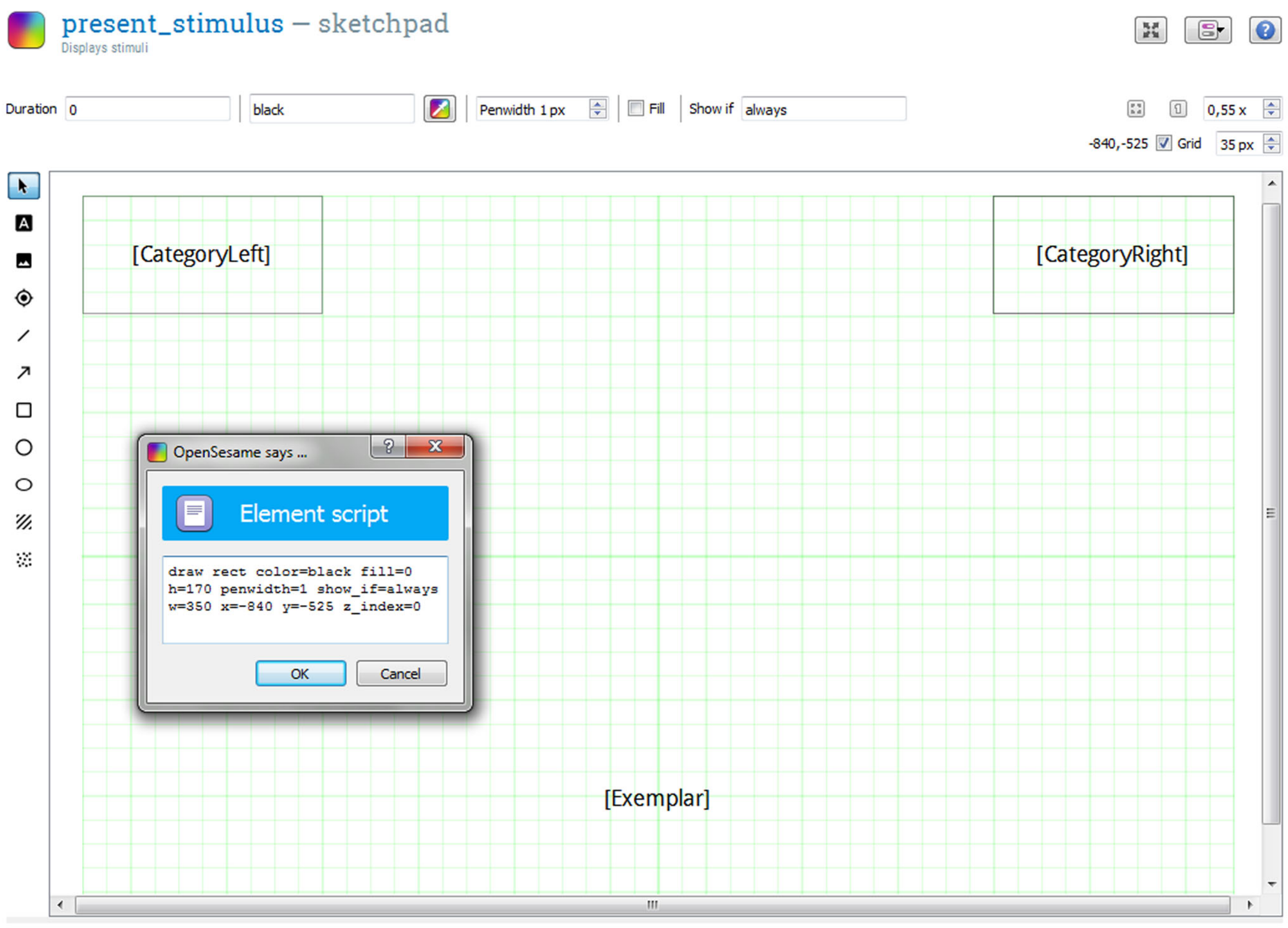
The start item is followed by a *sketchpad* that defines the actual stimulus (Fig. 2). In the most general terms, a typical mouse-tracking task involves the presentation of a stimulus (e.g., a name or picture of an object), and several buttons. In the current study, the buttons correspond to different categories, and the participant's task is to indicate which category the presented exemplar (i.e., the name of the animal as text) belongs to by clicking on the corresponding button.

The most important part of the mouse-tracking screen is the exemplar that is to be categorized. It is added to the sketchpad using a *textline element* which allows for creating formatted text. To vary the presented text in each trial and insert the data from the loop (cf. Fig. 1), the corresponding variable name can be added in square brackets.

Creating button-like elements on a sketchpad item consists of two steps. First, the borders of the buttons are drawn using *rect elements*. Next, the button labels are inserted using *textline elements* (again using the variable names from the loop in square brackets). When designing the buttons, a symmetrical layout is desirable in most cases. Importantly, all buttons should have the same distance from the starting position of the mouse. Typically, the buttons are placed in the corners of the screen so that participants can easily reach them without risking overshooting the button, yet the distance between buttons is maximal.

As the tracking of mouse movements should start immediately when the sketchpad is presented, the duration of the sketchpad is set to 0 and a *mousetrap_response* item is inserted directly after the sketchpad in the trial sequence (see Fig. 1, where the mousetrap_response item is labelled "get_response"). Because the mousetrap_response item is separated from the stimulus display, the number of buttons[2] as well as their location and internal name need to be provided (see Fig. 3). In our case, and indeed for the majority of experiments, the buttons correspond to the rectangles added to the sketchpad earlier. Thus, the appropriate values for x and y coordinates as well as width and height can be copied from the element script, which can be accessed by double-clicking on its border (Fig. 2). In addition to the coordinates, each

[2] The mousetrap_response item supports up to four buttons. More can be added by using the mousetrap_form item or by defining buttons in Python code.



Duration 0 | black | Penwidth 1 px | Fill | Show if always | 0,55 x
-840,-525 | Grid | 35 px

```
draw rect color=black fill=0
h=170 penwidth=1 show_if=always
w=350 x=-840 y=-525 z_index=0
```

Fig. 2 Exemplary sketchpad item containing two buttons and a stimulus. The drawing tools used to create the stimulus are shown on the left: The button labels and the stimulus are created using textline elements. As they vary for each trial, the experimental variables defined in the stimuli loop (cf. Fig. 1) are used by enclosing the variable name in square brackets, so that their values will be substituted when the experiment runs. The button borders are drawn using rect elements. The underlying element script for each button can be accessed by double-clicking on the respective rectangle: the script corresponding to the left button is shown in the pop-up window. The *x*, *y*, *w*, and *h* arguments define the left and top coordinates of the rectangle and its width and height. They can be copied and pasted into the mousetrap_response item (cf. Fig. 3) to define the buttons

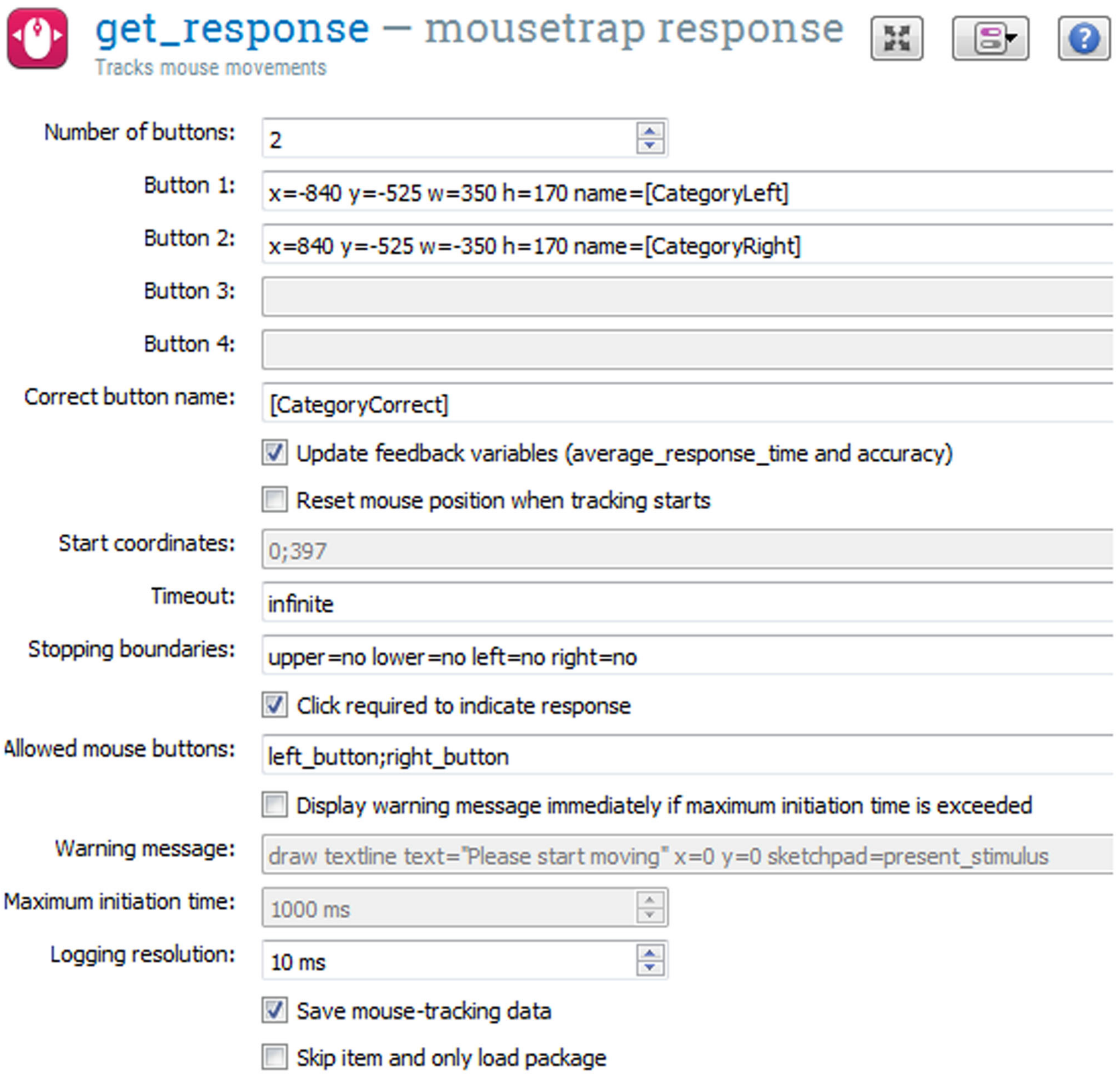
button receives a name argument that will be saved as response when a participant clicks within the area of the corresponding button. We recommend using the text content of the button for this purpose (e.g., `name =[CategoryLeft]` for the left button in Fig. 2).

When the response options are named, a correct response can be defined by adding the corresponding button's name in the respective field. OpenSesame will then automatically code the correctness of the response (as 1 or 0) in the variable labelled *correct*, which is included in the data for later analysis and can also be used to provide feedback during the study. As with the labels, the correct response in each trial is determined based on the variables specified in the loop (variable *CategoryCorrect*, cf. Fig. 1).

In addition to logging the correctness of a single response, OpenSesame's global feedback variables (e.g., the overall accuracy) can be updated automatically by selecting the corresponding option, which makes it easy to, for example, pay participants contingent on their performance. In the current experiment, participants are provided with feedback on their performance on the last screen of the experiment through this mechanism.

The cursor position is recorded as long as the mousetrap_response item is active. The interval in which the positions are recorded is specified under *logging resolution* in the item settings (see Fig. 3). By default, recording takes place every 10 ms (corresponding to a 100-Hz sampling rate). The actual resolution may differ depending on the performance of the hardware (but has proven to be very robust in our studies, see example experiment below and software validation in the Appendix).

Finally, a *logger* item is inserted at the end of the trial sequence (see Fig. 1). This item writes the current state of all variables to the participant's log file, which will later be used



Number of buttons:	2
Button 1:	x=-840 y=-525 w=350 h=170 name=[CategoryLeft]
Button 2:	x=840 y=-525 w=-350 h=170 name=[CategoryRight]
Button 3:	
Button 4:	
Correct button name:	[CategoryCorrect]
	[✓] Update feedback variables (average_response_time and accuracy)
	[ ] Reset mouse position when tracking starts
Start coordinates:	0;397
Timeout:	infinite
Stopping boundaries:	upper=no lower=no left=no right=no
	[✓] Click required to indicate response
Allowed mouse buttons:	left_button;right_button
	[ ] Display warning message immediately if maximum initiation time is exceeded
Warning message:	draw textline text="Please start moving" x=0 y=0 sketchpad=present_stimulus
Maximum initiation time:	1000 ms
Logging resolution:	10 ms
	[✓] Save mouse-tracking data
	[ ] Skip item and only load package

Fig. 3 Settings of the mousetrap_response item: The topmost settings define the number of buttons used, as well as their position (using the arguments from the rect element script, cf. Fig. 2) and internal name (the button label that was defined in the stimuli loop, cf. Fig. 1). The correct answer can be specified in the Correct button name option to make use of OpenSesame's feedback capabilities. If desired, the mouse cursor can be reset to exact start coordinates at tracking onset. Optionally, a timeout (in ms) can be specified to restrict the time participants have to give their answer. The boundary setting can be used to terminate data collection if the cursor crosses a specified vertical or horizontal boundary on the screen. Additional options concern the possibility to restrict the mouse buttons available for responding, the immediate display of a warning if cursor movement is not initiated within a given interval, and the adjustment of the logging resolution, that is, the interval between subsequent recordings of the cursor position

in the analysis. The variable inspector can be used to monitor the current state of the variables in the experiment if it is run from within OpenSesame. The central mouse-tracking data recorded through mousetrap items is stored in variables starting with *timestamps*, *xpos*, and *ypos*.

Alternative implementation using forms

As mentioned above, mousetrap also provides an alternative way of implementing mouse-tracking via the *mousetrap_form* item. In contrast to the mousetrap_response item, the display is defined directly within the item by using a form. Forms are a general item type which is used throughout OpenSesame. They place content (which is referred to as "widgets" and can include labels, images, buttons and image buttons) on a grid, which allows forms to scale with the display resolution. Forms do not provide a graphical interface, but instead use a simple syntax to define and arrange the content.

In the current example, a mousetrap_form could replace both the "present_stimulus" sketchpad and the "get_response" mousetrap_response item. Assuming a grid with 16 columns and 10 rows, a visual stimulus display similar to Fig. 2 can be created as follows:

```
widget 6 7 4 2 label text="[Exemplar]"
widget 0 0 4 2 button text="[CategoryLeft]"
widget 12 0 4 2 button text="[CategoryRight]"
```

The numbers in the example define the position and extent of each widget on the grid, followed by the type of element and its specific settings. The additional mouse-tracking settings are largely identical to the settings of the mousetrap_response item (see Fig. 3 and the online example experiment demonstrating a mousetrap_form).

Methodological considerations

With the basic structure of the experiment in place, the stimulus display designed and the mouse-tracking added, the experiment would now be ready to run. However, some additional methodological details should be given consideration. Mouse-tracking studies in the literature differ in many methodological aspects, depending on the implementation and researchers' preferences. We can provide no definitive recommendations, but we aim to cover most common design choices and their implementation using the mousetrap plugin in the following (see also Fischer & Hartmann, 2014; Hehman, Stolier, & Freeman, 2015, for recommendations regarding the setup of mouse-tracking experiments).

General display organization

One general challenge is the design of the information shown during the mouse-tracking task. Because mouse movements should reflect the developing commitment to the choice options rather than information search, the amount of new information that participants need to acquire during tracking should be minimized. At the same time, some information must be withheld until tracking begins, so that participants develop their preferences only during the mouse-tracking task and not before.

To some degree, this also represents a challenge for the current example experiment, where in addition to the name of the exemplar, the information about the two response categories needs to be acquired. Dale et al. (2007) solved this by presenting the response categories for 2,000 ms at the beginning of each trial, even before the start button appeared. The experiment sketched above can be adapted to implement this procedure by including an additional sketchpad in the beginning of the trial sequence that presents only the buttons and their labels for a specified duration. This procedure was also used in the experiment reported below.

Note that other mouse-tracking studies have used an alternative approach by presenting the critical information acoustically (e.g., Spivey et al., 2005). One advantage of this approach is that it prevents any artifacts that might be caused from reading visually presented information. This approach can be implemented easily in OpenSesame, for example, by inserting a sampler item between the sketchpad and the mousetrap_response item.

Starting position

As previously discussed, it is often desirable to have a comparable starting position of the cursor across trials, as is achieved through the start button in our experiment. However, this method only leads to generally comparable, but not identical starting positions across trials. Though the start coordinates can be aligned during the later analysis, the cursor position can also be reset to exact coordinates by the experimental software before the tracking starts. This can be achieved by checking the corresponding option, and the start coordinates can be specified as two integers (indicating pixel values in the sketchpad metric where "0;0" represents the screen center). These values are usually chosen to correspond to the center of the start button, so that the jump in position is minimized (the mousetrap_response item by default uses start coordinates that correspond to the center of the button on a form_text_display item).

Movement initialization

In many mouse-tracking studies, participants are explicitly instructed to initiate their mouse movement within a certain time limit (as described by Hehman et al., 2015) while other studies refrain from giving participants any instructions regarding mouse movement (e.g., Kieslich & Hilbig, 2014; Koop & Johnson, 2013). If such an instruction is given, compliance will typically be monitored and participants may be given feedback. The mousetrap items provide several ways of implementing this. The items automatically compute the *initiation_time* variable that contains the time it took the participant to initialize any mouse movement in the trial. This variable can be used to give feedback to the participant after the task, for example, by conditionally displaying a warning message if the initiation time is above a predefined threshold.

Alternatively, it is also possible to display a warning message while the mouse-tracking task is running. In this case, the time limits and the customized warning message can be specified in the item settings (see Fig. 3). We recommend not using this second option during the actual mouse-tracking task to avoid distracting participants. However, it might be useful in initial practice trials.

Going beyond a mere a priori instruction to initiate movement quickly, some studies have also used a more advanced procedure implementing a dynamic start condition (e.g., Dshemuchadse, Scherbaum, & Goschke, 2013; Frisch, Dshemuchadse, Görner, Goschke, & Scherbaum, 2015). In these studies, the critical stimulus information was presented only after participants crossed an invisible horizontal boundary above the start position, ensuring that movement had already been initiated. A dynamic start condition can be implemented by including an additional sketchpad and mousetrap_response item specifying an upper boundary for tracking in the item settings (see corresponding online example experiment).

In a first attempt to assess the influence of the starting procedure on mouse-tracking measures, Scherbaum and Kieslich (2017) compared data from an experiment using such a dynamic start condition to a condition in which the stimulus was presented after a fixed delay. While results showed that theoretically expected effects on trial-level mouse-tracking measures (i.e., trajectory curvature) were reliably found in both conditions, effects on within-trial continuous measures were stronger and more temporally distinguishable in the dynamic start condition. This was in line with generally more consistent and homogeneous movements in the dynamic start condition.

Another alternative to ensure a quick initialization of mouse movements is to restrict the time participants have for giving their answer. This time limit can be specified (in ms) in the corresponding option in the item settings (Fig. 3).[3]

Response indication

An additional methodological factor that varies across mouse-tracking studies is the way participants indicate their response. While many studies require participants to click on the button representing their choice (e.g., Dale et al., 2007; Koop, 2013), in other studies merely entering the area corresponding to the button with the cursor is sufficient (e.g., Dshemuchadse et al., 2013; Scherbaum et al., 2010). Both options are available in mousetrap (see Fig. 3). If a click is required, the (physical) mouse buttons that are accepted as a response indication can be specified. By default, mouse clicks are required and both left and right mouse clicks are accepted.

[3] However, introducing a time limit might also induce time pressure which might lead to other (undesired) effects.

Counterbalancing presentation order

A final consideration should be given to potential position effects: So as not to introduce confounds between response alternatives and the position of the corresponding button, the mapping should be varied across trials and / or across participants. This is especially important if the response alternatives stay constant across trials (which is often the case in decision making studies, e.g., Kieslich & Hilbig, 2014; Koop, 2013). In the current study, the position of the correct response and the foil (left vs. right) should be varied. This can be done statically by varying their order across trials (see Fig. 1). To go further, the position of response options can be randomized at run time using OpenSesame's advanced loop operations (as was done in the experiment reported below, see also *shuffle_horiz* online example experiment).

Data collection

After creating the mouse-tracking experiment, it should be tested on the computers that will later be used to collect the data. We also recommend importing and analyzing self-created test data to check that all relevant independent and dependent variables have been recorded, and to check the logging resolution (see below). When preparing the study for running in the lab, a number of methodological factors need to be considered.

As noted in the previous section, mouse-tracking experiments should be run in full screen mode at the maximum possible screen resolution. The *OpenSesame Run* program, which is included with OpenSesame, can run the experiment without having to open it in the editor, making the starting process more efficient, and hiding the internal structure, conditions, and item names from participants.

In addition, the mouse sensitivity settings of the operating system should be checked and matched across laboratory computers, in particular the speed and acceleration of the cursor relative to the physical mouse (these settings cannot be influenced directly from within OpenSesame). There is currently no single setting applied consistently across studies in the literature, and the settings used in the field are often not reported. Presumably, the settings will often have been left to the operating system defaults (under Windows 7 and 10, medium speed with acceleration) or speed will have been reduced deliberately and acceleration turned off (as recommended by Fischer & Hartmann, 2014).

When preparing the laboratory, it should be ensured that participants have enough desk space to move the mouse. In this regard, we have found it useful to move the keyboard out of the way and design the experiment so that participants can complete the entire experiment by using only the mouse. Additionally, heretofore largely unexplored factors concern the handedness of participants, the hand used for moving the mouse, and their interplay. Some authors go as far as to recommend including only right-handed participants (Hehman et al., 2015). We would recommend assessing the handedness of participants, as well as the hand actually used for moving the mouse in the experiment.

In general, we would like to stress the importance of documenting mouse-tracking studies in sufficient detail, both so that fellow researchers can replicate the experiment and so that potentially differing findings between individual mouse-tracking studies can be traced back to differences in their methodological setup. Ideally, each of the degrees of freedom sketched above should be documented, as well as the specifics of the lab computers (especially screen resolution and mouse sensitivity settings). It is also very useful to provide a screenshot of the actual mouse-tracking task. Finally, to give interested colleagues the opportunity to explore the specific details of the task setup, it is also useful to provide them directly with the experiment files. This is particularly easy if mouse-tracking experiments are created in OpenSesame with the mousetrap plugin, as OpenSesame is freely available for many platforms. OpenSesame also provides the option to automatically save experiments on the Open Science Framework and share them with other researchers.

Example experiment

Having built and tested the experiment, enterprising colleagues could begin with the data collection immediately. We have done exactly this, and have performed a replication of Experiment 1 by Dale et al. (2007). In doing so, we aimed to assess the technical performance of the plugins (especially with regard to the logging resolution), to demonstrate the structure of the resulting data and how they can be processed and analyzed, and to replicate the original result that atypical exemplars lead to more curved trajectories than typical exemplars. The exact experiment that was used in the study (with German material and instructions) and a simplified but with regard to the task identical version (with English example material and instructions) can be found online at https://github.com/pascalkieslich/mousetrap-resources, as can the raw data and analysis scripts.

Methods

We used the 13 typical and 6 atypical stimuli from Dale et al.'s Experiment 1 (see Table 1 in Dale et al., 2007) translated to German. Participants first received instructions about their task and completed three practice trials. Thereafter, the 19 stimuli of interest were presented in random order. Participants were not told that their mouse movements were recorded, nor did they receive any specific instructions about moving the mouse.

Each trial began with a blank screen that was presented for 1,000 ms. After that, the two categories were displayed for 2,000 ms in the top left and right screen corners (the order of the categories was randomized at run time), following the procedure of the original study. Next, the start button appeared in the bottom center of the screen, and participants started the trial by clicking on it. Directly thereafter (the cursor position was not reset in this study), the to-be-categorized stimulus word was displayed above the start button and participants could indicate their response by clicking on one of the two categories (see Fig. 2).

The experiment was conducted full screen with a resolution of 1,680 × 1,050 pixels. Laboratory computers were running Windows 7, and mouse settings were left at their default values (acceleration turned on, medium speed). Cursor coordinates were recorded every 10 ms.

The experiment was conducted as the second part in a series of unrelated studies. Before the experiment, we assessed participants' handedness using the Edinburgh Handedness Inventory (EHI; Oldfield, 1971). We used a modified version of the EHI with a five-point rating scale on which participants indicated which hand they preferred to use for ten activities (-100 = exclusively left, −50 = preferably left, 0 = no preference, 50 = preferably right, 100 = exclusively right) and included an additional item for computer mouse usage.

Participants were recruited from a local student participant pool at the University of Mannheim, Germany, and paid for their participation (the payment was variable and depended on other studies in the same session). Participants were randomly assigned to either an implementation of the study using the mousetrap plugin in OpenSesame ($N = 60$, 39 female, mean age = 22.2 years, $SD = 3.5$ years) or another implementation (a development version of an online mouse-tracking data collection tool) not included in the current article. Participants' mean handedness scores based on the original EHI items indicated a preference for the right hand for the majority of participants (50 of 60 participants had scores greater than 60), no strong preference for eight participants (scores between −60 and 60) and preference for the left hand for two participants (below −60). Interestingly, all participants reported using a computer mouse preferably or exclusively with the right hand, as indicated by the newly added item.

Data preprocessing

In the following section, we focus on a simple but frequently applied comparison of (aggregate) mouse trajectory curvature.[4] In doing so, we will go through all analysis steps from loading the raw data to the statistical tests in the statistical programming language R (R Core Team, 2016). The complete analysis script is shown in Fig. 4.

The libraries required for the following analyses can be installed from CRAN using the following command: `install.packages(c("readbulk","mousetrap"))`. Thereafter, both libraries are loaded using `library(readbulk)` and `library(mousetrap)` respectively. We will only touch upon the most basic features of both; additional library-level documentation can be accessed with the command `package?mousetrap` (or online at http://pascalkieslich.github.io/mousetrap/), and help for specific functions is available by prepending a question mark to any given command, as in `?mt_import_mousetrap`.

OpenSesame produces an individual comma-separated (CSV) data file for each participant. Because there is a single logger item in the experiment that is repeated with each trial, every line corresponds to a trial. Different variables are spread across different columns. For our purposes, the most important columns are those containing the mouse-tracking data, namely the columns beginning with *timestamps*, *xpos*, and *ypos*. These columns contain the interval since the start of the experiment in milliseconds, and the x and y coordinates of the cursor at each of these time points. The position coordinates are given in pixels, whereby the value 0 for both x and y coordinates corresponds to the center of the screen and values increase as the mouse moves toward the bottom right.

As a first step after opening R (or RStudio), the current working directory should be changed to the location where the raw data is stored (either using *setwd* or via the user interface in RStudio). To read the data of all participants into R, we suggest the *readbulk* R package (Kieslich & Henninger, 2016), which can read and combine data from multiple CSV files into a single dataset. Readbulk provides a specialized function for OpenSesame data (*read_opensesame*). Assuming that the raw data is stored in the subfolder "raw_data" of the working directory, we can combine all individual files into a single data.frame using `read_opensesame("raw_data")`.

Next, the raw data are filtered so that only the trials of interest are retained. Specifically, all trials from the practice phase are excluded. Besides, we determined which trials were solved correctly using the *correct* variable, which was automatically set by the mousetrap_response item. The accuracy in the current study was 88.9% for atypical and 95.4% for typical trials – results comparable to those in the original study. Following Dale et al., only the correctly completed trials were kept for the analyses.

For preprocessing and analyzing mouse-tracking data, we have developed the *mousetrap* R package (Kieslich et al., 2016). A detailed description of the package and its functions is provided elsewhere (Kieslich, Wulff, Henninger, Haslbeck, & Schulte-Mecklenbeck, 2017). In the following, we will focus on the most basic functions needed for the present analyses.

As a precondition for further analysis, the raw data must be represented as a mousetrap data object using the *mt_import_mousetrap* function. This function will automatically select the mouse-tracking data columns from the raw data[5] and transform their contents into a data structure amenable to analysis.

Next, several preprocessing steps ensure that the data can be aggregated within and compared meaningfully between conditions. Trajectories are remapped using *mt_remap_symmetric* which ensures that every trajectory starts at the bottom of the coordinate system and ends in the top left corner (regardless of whether the left or the right response option was chosen). Because the mouse cursor was not reset to a common coordinate at the start of tracking, *mt_align_start* is needed to align all trajectories to the same initial coordinates (0, 0). Trajectories are then typically time-normalized so that each trajectory contains the same number of recorded coordinates regardless of its response time (Spivey et al., 2005). To this end, *mt_time_normalize* computes time-normalized trajectories using a constant (but adjustable) number of time steps of equal length (101 by default, following Spivey et al.).

Several different measures for the curvature of mouse trajectories have been proposed in the literature (Freeman & Ambady, 2010; Koop & Johnson, 2011). One frequently used measure is the maximum absolute deviation (MAD). The MAD represents the maximum perpendicular deviation of the actual trajectory from the idealized trajectory, which is the straight line connecting the trajectories' start and end points.[6] The MAD and many additional trial-level measures can be calculated using the *mt_measures* function.[7] These measures are then typically aggregated per participant for each level of the within-participants factor. For this, *mt_aggregate_per_subject* can be used (see Fig. 4).

[4] These analyses differ from the more elaborate analyses in the original article by Dale et al. (2007), which we have omitted for reasons of brevity. We provide an R script for replication of the original analyses online.

[5] In case that more than one mousetrap item is included in the experiment, the names of the columns need to be provided explicitly using the corresponding arguments.

[6] If this maximum deviation occurs in the direction of the non-chosen option (i.e., "above" the idealized trajectory), it receives a positive sign, otherwise a negative sign.

[7] This function uses the raw trajectories by default to avoid the (unlikely) possibility that relevant spatial information gets lost during time normalization. In the current sample, the MAD values based on the raw trajectories and on the time-normalized trajectories correlate to .9999.

```r
library(readbulk)
raw_data <- read_opensesame("raw_data", verbose = FALSE)
raw_data <- subset(raw_data, Condition!="Example" & correct==1)

library(mousetrap)
mt_data <- mt_import_mousetrap(raw_data)
mt_data <- mt_remap_symmetric(mt_data)
mt_data <- mt_align_start(mt_data)
mt_data <- mt_measures(mt_data)

agg_measures <- mt_aggregate_per_subject(mt_data, subject_id = "subject_nr",
  use_variables = "MAD", use2_variables = "Condition")
t.test(MAD~Condition, data = agg_measures, paired = TRUE)

mt_data <- mt_time_normalize(mt_data)
mt_plot_aggregate(mt_data, use = "tn_trajectories", points = TRUE,
  x = "xpos", y = "ypos", color = "Condition", subject_id = "subject_nr")
```

Fig. 4 R script for replicating the main data preparation and analysis steps. First, the individual raw data files are merged and read into R. They are then filtered, retaining only correctly solved trials from the actual task. Next, the mouse-tracking data are imported and preprocessed by remapping all trajectories to one side, aligning their start coordinates and computing trial-level summary statistics (such as the maximum absolute deviation, MAD). The MAD values are aggregated per participant and condition, and compared using a paired *t*-test. Finally, the trajectories are time-normalized, aggregated per condition, and visualized

Data quality check

To check whether the intended logging resolution was actually met, *mt_check_resolution* can be used to compute the achieved interval between logs. Across all recorded mouse positions in all trials that entered the following analyses, 99.4% of the logging intervals were exactly 10 ms, corresponding to the desired logging resolution. An additional 0.5% of intervals were shorter than 10 ms, due to the fact that every click in the experiment leads to an immediate recording of the current cursor position, even outside of the defined logging interval. Finally, 0.1% of logging intervals were greater than 10 ms, of which 76.2% lagged by 1 additional ms only. Overall, the mean timestamp difference was 9.98 ms (*SD* = 0.43 ms).

A more comprehensive technical validation of the mousetrap plugin is reported in the appendix. Extending a procedure by Freeman and Ambady (2010), we used external hardware (Henninger, 2017) to generate known movement patterns from the start button to one of the response buttons. An analysis of the recorded cursor positions revealed that almost every change in position was captured on the raw coordinate level, and that the recorded positions and derived trial-level measures almost perfectly corresponded to their expected values.

Results

A quick first visual impression of the effect of the typicality manipulation on mouse movements can be obtained by inspecting the aggregate mouse trajectories. Specifically, *mt_plot_aggregate* can be used to average the time-normalized trajectories per condition (first within and then across participants) and to plot the resulting aggregate trajectories (Fig. 5). In line with the hypothesis by Dale et al., the aggregate response trajectory in the atypical condition showed a greater attraction to the non-chosen option than the trajectory in the typical condition.

To statistically test for differences in curvature, the average MAD values per participant and condition can be compared. In line with the hypothesis and the visual inspection of the aggregate trajectories, the MAD was larger in the atypical ($M = 343.8$, $SD = 218.6$) than in the typical condition ($M =$

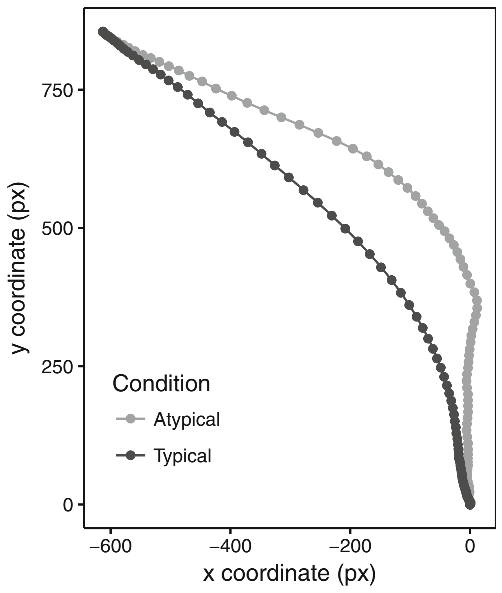


Fig. 5 Average time-normalized trajectories per experimental condition

172.2, $SD = 110.8$). This difference was significant in a paired t-test, $t(59) = 6.73$, $p < .001$, and the standardized difference of $d_z = 0.87$ represented a large effect.

The analyses just described give an initial impression of what mouse-tracking data look like. While we have provided a first simple test of our basic hypothesis, the analysis has barely scratched the surface of what is possible with this data (and what can be realized using the mousetrap package). Specifically, we have skipped a number of important preprocessing and analyses steps that are standard procedure in mouse-tracking studies, such as the inspection of individual trials to detect anomalous or extreme mouse movements (Freeman & Ambady, 2010) and analyses to detect the presence of bimodality (Freeman & Dale, 2013).[8] The original article our study was based upon also contains many more analyses (see online supplementary material for a replication of the analyses by Dale et al., 2007, based on the current dataset).

Several more advanced analyses methods and measures have also been proposed, such as velocity and acceleration profiles, spatial disorder analyses via sample entropy, or the investigation of smooth versus abrupt response competition via distributional analyses (see, e.g., Hehman et al., 2015, for an overview). Many of these methods and measures are implemented in the mousetrap R package, and are described and explained in the package documentation. We discuss elsewhere in detail the methodological possibilities and considerations when processing and analyzing mouse-tracking data, as well as their implementation in mousetrap (Haslbeck, Wulff, Kieslich, Henninger, & Schulte-Mecklenbeck, 2017; Kieslich et al., 2017).

Discussion

In this article, we presented the free and open-source software *mousetrap* that offers users easy and convenient means of recording mouse movements, and demonstrated how a simple experiment can be built and analyzed. Specifically, we introduced mousetrap as a plugin that adds mouse-tracking to the popular, open-source experiment builder OpenSesame, allowing users to create mouse-tracking experiments via a graphical user interface. To demonstrate the usage of mousetrap, we created and replicated a mouse-tracking experiment by Dale et al. (2007), and analyzed the resulting data using the mousetrap R package. In line with the original hypothesis and results, we found that mouse trajectories displayed greater curvature towards the competing response option for atypical compared to typical exemplars. Naturally, we have only been able to discuss the most salient decisions in the construction of mouse-tracking experiments. However, where possible, we have noted the additional degrees of freedom and design choices, and sketched their implementation.

Mousetrap offers an alternative to the two major ways mouse-tracking studies are currently implemented. First, researchers have built custom code-based implementations of mouse-tracking for specific paradigms. These custom-built experiments can be flexibly tailored to the individual researchers' needs, but their implementation requires extensive programming skills, and paradigms are often cumbersome to adapt to new tasks. Secondly, researchers have relied on MouseTracker (Freeman & Ambady, 2010), a specialized experimental software for building mouse-tracking experiments and analyzing the resulting data. While this software has made mouse-tracking studies accessible to more researchers by providing a visual interface for designing the mouse-tracking screen and recording the mouse movements, it forgoes the flexibility and many useful features of general-purpose experimental software (such as the option to define the structure of the experiment itself via a graphical user interface, or to directly include a scripting language for customization and run time adaptation).

Aiming to combine the advantages while avoiding the disadvantages of both approaches, mousetrap extends the general purpose graphical experiment builder OpenSesame (Mathôt et al., 2012). Thereby, it allows users to easily create mouse-tracking experiments via a graphical interface without requiring programming skills. In addition, it makes available the many useful features of OpenSesame, such as a user-friendly interface for designing the structure of the experiment and implementing advanced randomizations, the support for diverse audiovisual stimuli, an open data format, extensibility via Python scripts, and cross-platform availability.

While mouse-tracking is a frequently used method for assessing response dynamics (Koop & Johnson, 2011), it should be noted that other methods are also available, such as the use of remote controllers (e.g., a Nintendo Wii Remote, cf. Dale, Roche, Snyder, & McCall, 2008) or the direct recording of hand movements (via a handle, e.g., Resulaj, Kiani, Wolpert, & Shadlen, 2009, or using a motion capture system, e.g., Awasthi, Friedman, & Williams, 2011). Another approach that might become more important in future research is the tracking of finger (or pen) movements via touchscreens (e.g., Buc Calderon, Verguts, & Gevers, 2015; Wirth, Pfister, & Kunde, 2016) due to the increasing availability of tablets

[8] A simple bimodality analysis can be conducted by computing bimodality coefficients (BC). Following Freeman and Ambady (2010), we z-standardized MAD values per participant and computed the BC separately for the atypical and the typical condition. In both conditions, the BC was higher than the recommended cutoff (.555), $BC_{Typical} = .608$, $BC_{Atypical} = .593$, indicating a bimodal distribution. To analyze whether the difference in MAD between typicality conditions remained significant after excluding outliers, we excluded all trials with $|z_{MAD}| > 1.50$ and repeated the main analyses (for details, see online supplementary material). As in the complete dataset, aggregate MAD was significantly higher in the atypical than in the typical condition, $p < .001$. Note, however, that more advanced and comprehensive alternative analyses are available (Kieslich et al., 2017).

and smartphones. The mousetrap plugin could be extended to implement the latter approach in OpenSesame.

With mousetrap, we hope to make mouse-tracking accessible to researchers from many different fields, and thereby to enable them to gain insights into the dynamics of cognitive processes. Given the fast-paced development of the mouse-tracking method, we hope that our modular and open approach will help users to implement the increasingly complex designs, to combine mouse-tracking with other process tracing methods such as eye-tracking, and to apply the method in fields where only few mouse-tracking studies have been conducted so far, such as behavioral economics with real-time interactive experiments. Similarly, we hope that the open data format and the close link to open analysis tools such as those demonstrated herein will make the manifold methods of analyzing mouse-tracking data widely available.

Acknowledgments We thank Anja Humbs for testing a development version of the mousetrap plugin for OpenSesame, Monika Wiegelmann and Mila Rüdiger for collecting the data for the example experiment, and Arndt Bröder and Johanna Hepp for helpful comments on an earlier version of this manuscript. This work was supported by the University of Mannheim's Graduate School of Economic and Social Sciences funded by the German Research Foundation

Appendix

Software validation

To validate the data collection procedure, we extended the procedure employed by Freeman and Ambady (2010), who simulated and processed artificial mouse trajectories. We used external hardware (Henninger, 2017) to generate two known movement patterns that connected the start and the top left response button: either a diagonal line, or a triangular path leading only upward at first, and then left towards the response button (Fig. 6). The validation experiment was built in OpenSesame (version 3.1.6, using the *legacy* backend[9]) using the mousetrap_response item (version 1.2.1). The screen layout and mouse-tracking settings were identical to the example experiment reported in the main article (cf. Figs. 2 and 3). The study was run on a laboratory terminal with modest hardware (Windows 7 Professional, on an Intel Pentium Dual-Core running at 3 GHz with 4 GB RAM).

In the following simulations, we ventured to perform a strict test of the software: First, to test the performance of the data collection procedure under heavy load, we simulated rapidly changing cursor coordinates. Specifically, in all simulations, the cursor position was updated at the logging resolution (10 ms) to assess whether data is recorded correctly when the cursor position changes as fast as data are collected. On each update, the cursor moved to the next integer pixel location on its path, that is, both one pixel up and one left for 800 px for the diagonal trajectory, or first one pixel upwards for 800 px and then one left for 800 px for the triangular path. The trial was started by a (simulated) click on a start button, which initiated the display of the response buttons, and ended with a mouse click on the left response button, with pauses of 110 ms before movement initiation and 100 ms between the end of movement and the simulated response. This means that the time between the start and end click was 8,210 ms for the diagonal path and 16,210 ms for the triangular path. Second, we validate the resulting data at the lowest possible level, that is, using the raw trajectory coordinates of each individual (simulated) trial. In scientific practice, standard mouse-tracking analyses will compensate for imperfect measurement to some degree because mouse trajectories are typically time-normalized and analyses are based on aggregate statistics.

For both the diagonal and the triangular path, we simulated 1,000 trials. The resulting data files were read into R and processed and analyzed using the mousetrap R package (Kieslich et al., 2016). All data and analyses scripts can be found at https://github.com/pascalkieslich/mousetrap-os#validation.

To determine the temporal alignment between the external hardware and the data recorded by the mousetrap_response item, we performed several analyses (based on the absolute timestamps recorded in OpenSesame): After the click on the start button, the screen with the response buttons was displayed with an average delay of 6.9 ms ($SD = 0.7$ ms) in both simulations. Mouse-tracking started after an additional delay of 0.7 ms ($SD = 0.5$ ms). This means that, on average, 7.6 ms passed between a click on the start screen and tracking onset on the next screen. Taking this delay into account, the observed tracking durations[10] in both simulations matched the expected value very closely, with an average duration of 8202.9 ms ($SD = 0.9$ ms) for the diagonal simulation, and an average duration of 16203.1 ms ($SD = 0.9$ ms) for the triangular simulation.

Next, we assessed whether the specified logging resolution was met, using the *mt_check_resolution* function to compute the time interval between subsequent recorded cursor positions. In the diagonal simulation, the mean interval was 10.0 ms ($SD = 0.3$ ms) matching the intended logging interval. Specifically, 99.86% of the logging intervals were exactly

[9] OpenSesame provides other backends with superior temporal accuracy. However, we used legacy in our simulations and the example experiment, as it is generally more stable, especially when using forms, which are often used when designing mouse-tracking experiments. More information on general benchmark results for OpenSesame can be found at http://osdoc.cogsci.nl/manual/timing/

[10] Tracking durations can be obtained via the *response_time* variable stored in OpenSesame or by using the *RT* variable computed from the timestamps using the *mt_measures* function of the mousetrap R package. Both approaches lead to identical results.

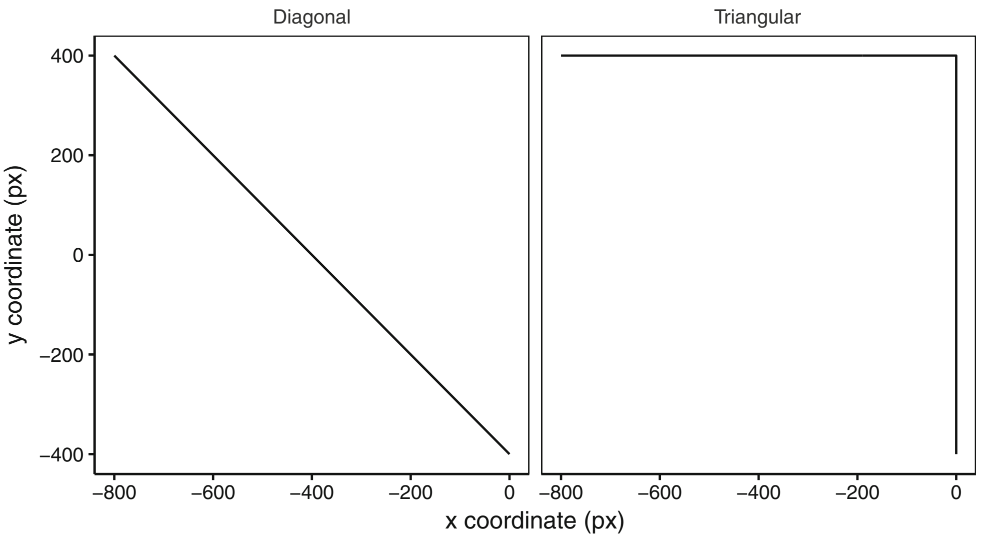


Fig. 6 Plot of all raw trajectories for each simulation. All trajectories started at the bottom center of the screen and ended at the top left

10 ms, corresponding precisely to the desired logging resolution. An additional 0.12% of intervals were shorter than 10 ms, due to the fact that each click led to an immediate recording of the current cursor position, even before the end of a logging interval (and because logging was not exactly synchronized with simulated cursor movements and clicks). Finally, 0.02% of logging intervals were greater than 10 ms, of which 99.3% lagged by 1 additional ms only. Similar results were obtained in the triangular simulation, in which the mean timestamp difference was 10.0 ms (*SD* = 0.2 ms) and where 99.92% of the logging intervals were exactly 10 ms, 0.06% were shorter, and 0.02% longer (of which 94.4% lagged by 1 ms only).

To gain a first visual impression of the data, all raw trajectories were plotted separately for the two simulations. As can be seen in Fig. 6, all trajectory shapes were perfectly aligned within each simulation and no anomalous positions were recorded.

Missed position changes due to lags in the logging interval can be identified simply by computing the distance between two adjacent cursor positions recorded in each trial. These are expected to be either 0 px for a period where the cursor did not move along the respective dimension or 1 px along one (for the triangular simulation) or both (for the diagonal simulation) dimensions for a period with movement. Any value greater than 1 px indicates a missed change in position. In the diagonal simulation, 99.9995% of the subsequently recorded positions were either 0 px or 1 px apart for both x and y coordinates – the remaining 0.0005% differed by 2 px along either dimension, indicating that a single movement was missed. In the triangular simulation that involved changes in x coordinate only for the first, and y coordinate only for the second half of the trial, for the x coordinates, 99.9949% of the distances were either 0 px or 1 px, and 0.0051% were 2 px indicating that a single movement was missed (in only a single additional case were two changes in position missed). For the y coordinates, 99.9953% of the distances were either 0 px or 1 px, and 0.0047% were 2 px.

To assess the accuracy of the recorded cursor position at each point during the trial, we computed its expected position for each set of recorded coordinates (based on the known path generated by the external hardware, and taking into account the average tracking onset). We then computed Pearson correlations between the observed and the expected position separately for the x and y coordinates. In the diagonal simulation, the correlation was .99999999996 for both x and y coordinates, and the expected and observed position were identical in 99.9995% of cases (and differed by 1 px for the remaining cases). In the triangular simulation, the correlation was .999999993 for the x coordinates, and .999999995 for the y coordinates. For the x coordinates, the observed and expected position were identical in 99.8994% of cases (and differed by 1 px for all remaining cases except one, where it differed by 2 px). For the y coordinates, the observed and expected position were identical in 99.9298% of cases (and differed by 1 px for the remainder).

Table 1 Expected values, observed mean and standard deviation for selected mouse-tracking measures per simulation

	Diagonal			Triangular		
	MAD	AUC	AD	MAD	AUC	AD
Expected	0.00	0.00	0.00	565.69	320000.00	279.01
M	0.00	0.00	0.00	565.69	320000.00	279.01
SD	0.00	0.00	0.00	0.00	0.00	0.02

MAD maximum absolute deviation, *AUC* area under curve, *AD* average deviation.

In the diagonal simulation, *M*s for MAD and AD were $< 9*10^{-14}$ and *SD* for AD was $< 3*10^{-20}$

Finally, we computed a number of mouse-tracking indices based on the raw trajectory data, using the *mt_measures* function. The descriptive statistics for a selection of the measures can be found in Table 1. In line with the expected measures based on the predetermined paths, the maximum absolute deviation (MAD), area under curve (AUC) and average deviation (AD) were 0 for the diagonal simulation and did not vary between trials. For the triangular simulation, the MAD always met the expected value of 565.69 px (which is the height of a right-angled triangle where both legs have a length of 800 px) and the AUC was always 320,000 px^2 (which corresponds exactly to the area of the previously described triangle). The AD values were on average also as expected ($M = 279.01$ px) with a minor variation between trials ($SD = 0.02$ px) because the AD takes every logged coordinate value into account and is therefore most sensitive to variations therein.

In sum, with regard to both logging resolution and measured coordinates, the mousetrap plugin for OpenSesame captures the raw mouse trajectory extremely well. It should be noted that the current validation was performed under even stricter conditions than those used in the validation of another software package (Freeman & Ambady, 2010): in the current simulation, the cursor was updated at a higher rate (every 10 ms instead of 30 ms) and more fine-grained analyses were used, focusing on exact raw trajectories instead of averaged data. When applied to actual data, even the remaining minute discrepancies will most often be negligible given that mouse-tracking analyses usually interpolate the raw trajectories to some extent (e.g., through time-normalization) and analyze trial summary statistics such as the measures reported above. Thus, we are confident that our software will perform reliably under most conditions.

References

Awasthi, B., Friedman, J., & Williams, M. A. (2011). Faster, stronger, lateralized: Low spatial frequency information supports face processing. *Neuropsychologia, 49*(13), 3583–3590. doi:10.1016/j.neuropsychologia.2011.08.027

Buc Calderon, C., Verguts, T., & Gevers, W. (2015). Losing the boundary: Cognition biases action well after action selection. *Journal of Experimental Psychology: General, 144*(4), 737–743. doi:10.1037/xge0000087

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15–28. doi:10.3758/BF03195938

Dale, R., Roche, J., Snyder, K., & McCall, R. (2008). Exploring action dynamics as an index of paired-associate learning. *PLOS ONE, 3*(3), e1728. doi:10.1371/journal.pone.0001728

Dalmaijer, E. S., Mathôt, S., & Van der Stigchel, S. (2014). PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior Research Methods, 46*(4), 913–921. doi:10.3758/s13428-013-0422-2

Dshemuchadse, M., Scherbaum, S., & Goschke, T. (2013). How decisions emerge: Action dynamics in intertemporal decision making. *Journal of Experimental Psychology: General, 142*(1), 93–100. doi:10.1037/a0028499

Fischer, M. H., & Hartmann, M. (2014). Pushing forward in embodied cognition: May we mouse the mathematical mind? *Frontiers in Psychology, 5,* 1315. doi:10.3389/fpsyg.2014.01315

Freeman, J. B., & Ambady, N. (2010). MouseTracker: Software for studying real-time mental processing using a computer mouse-tracking method. *Behavior Research Methods, 42*(1), 226–241. doi:10.3758/BRM.42.1.226

Freeman, J. B., & Dale, R. (2013). Assessing bimodality to detect the presence of a dual cognitive process. *Behavior Research Methods, 45*(1), 83–97. doi:10.3758/s13428-012-0225-x

Freeman, J. B., Dale, R., & Farmer, T. A. (2011). Hand in motion reveals mind in motion. *Frontiers in Psychology, 2,* 59. doi:10.3389/fpsyg.2011.00059

Frisch, S., Dshemuchadse, M., Görner, M., Goschke, T., & Scherbaum, S. (2015). Unraveling the sub-processes of selective attention: Insights from dynamic modeling and continuous behavior. *Cognitive Processing, 16*(4), 377–388. doi:10.1007/s10339-015-0666-0

Haslbeck, J. M. B., Wulff, D. U., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2017). *Advanced mouse- and hand-tracking analysis: Detecting and visualizing clusters in movement trajectories*. Manuscript in preparation.

Hehman, E., Stolier, R. M., & Freeman, J. B. (2015). Advanced mouse-tracking analytic techniques for enhancing psychological science. *Group Processes & Intergroup Relations, 18*(3), 384–401. doi:10.1177/1368430214538325

Henninger, F. (2017). *The participant on a chip: Flexible, low-cost, high-precision validation of experimental software*. Manuscript in preparation.

Henninger, F., Kieslich, P. J., & Hilbig, B. E. (in press). Psynteract: A flexible, cross-platform, open framework for interactive experiments. *Behavior Research Methods*. doi:10.3758/s13428-016-0801-6

Kieslich, P. J., & Henninger, F. (2016). Readbulk: An R package for reading and combining multiple data files. doi:10.5281/zenodo.596649

Kieslich, P. J., & Hilbig, B. E. (2014). Cognitive conflict in social dilemmas: An analysis of response dynamics. *Judgment and Decision Making, 9*(6), 510–522.

Kieslich, P. J., Wulff, D. U., Henninger, F., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2016). Mousetrap: An R package for processing and analyzing mouse-tracking data. doi:10.5281/zenodo.596640

Kieslich, P. J., Wulff, D. U., Henninger, F., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2017). *Mouse- and hand-tracking as a window to cognition: A tutorial on implementation, analysis, and visualization*. Manuscript in preparation.

Koop, G. J. (2013). An assessment of the temporal dynamics of moral decisions. *Judgment and Decision Making, 8*(5), 527–539.

Koop, G. J., & Johnson, J. G. (2011). Response dynamics: A new window on the decision process. *Judgment and Decision Making, 6*(8), 750–758.

Koop, G. J., & Johnson, J. G. (2013). The response dynamics of preferential choice. *Cognitive Psychology, 67*(4), 151–185. doi:10.1016/j.cogpsych.2013.09.001

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods, 44*(2), 314–324. doi:10.3758/s13428-011-0168-7

Oldfield, R. C. (1971). The assessment and analysis of handedness: The Edinburgh inventory. *Neuropsychologia, 9*(1), 97–113. doi:10.1016/0028-3932(71)90067-4

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/

Resulaj, A., Kiani, R., Wolpert, D. M., & Shadlen, M. N. (2009). Changes of mind in decision-making. *Nature, 461*(7261), 263–266. doi:10.1038/nature08275

Scherbaum, S., Dshemuchadse, M., Fischer, R., & Goschke, T. (2010). How decisions evolve: The temporal dynamics of action selection. *Cognition, 115*(3), 407–416. doi:10.1016/j.cognition.2010.02.004

Scherbaum, S., & Kieslich, P. J. (2017). *Stuck at the starting line: How the starting procedure influences mouse-tracking data*. Manuscript submitted for publication.

Spivey, M. J., & Dale, R. (2006). Continuous dynamics in real-time cognition. *Current Directions in Psychological Science, 15*(5), 207–211. doi:10.1111/j.1467-8721.2006.00437.x

Spivey, M. J., Grosjean, M., & Knoblich, G. (2005). Continuous attraction toward phonological competitors. *Proceedings of the National Academy of Sciences of the United States of America, 102*(29), 10393–10398. doi:10.1073/pnas.0503903102

Wirth, R., Pfister, R., & Kunde, W. (2016). Asymmetric transfer effects between cognitive and affective task disturbances. *Cognition and Emotion, 30*(3), 399–416. doi:10.1080/02699931.2015.1009002