Visuomotor Serial Targeting Task

SSC

Apr 22, 2024

GETTING STARTED

Visuomotor Serial Targetting Task (VSTT) is an open source Python GUI tool for designing, running and analyzing motor skill acquisition experiments.

CHAPTER

ONE

INSTALLATION

1.1 Windows with standalone Psychopy (recommended)

For windows users the recommended way to install VSTT is to first install StandalonePsychoPy, then use the VSTT windows installer:

1. Install StandalonePsychoPy
2. Install Visuomotor Serial Targeting Task

1.2 Pip install

VSTT can also be installed from PyPI using pip:

```
pip install vstt
```

Note: On linux the optional psychtoolbox dependency needs permission to set its priority. To allow this:

- `sudo setcap cap_sys_nice+ep `python -c "import os; import sys; print(os.path.realpath(sys.executable))"``

Alternatively you can simply remove psychtoolbox:

- `pip uninstall psychtoolbox`

CHAPTER

TWO

CREATING A NEW EXPERIMENT

To create a new experiment, go to File -> New, or press `Ctrl+N`, or click on the white new document toolbar button.

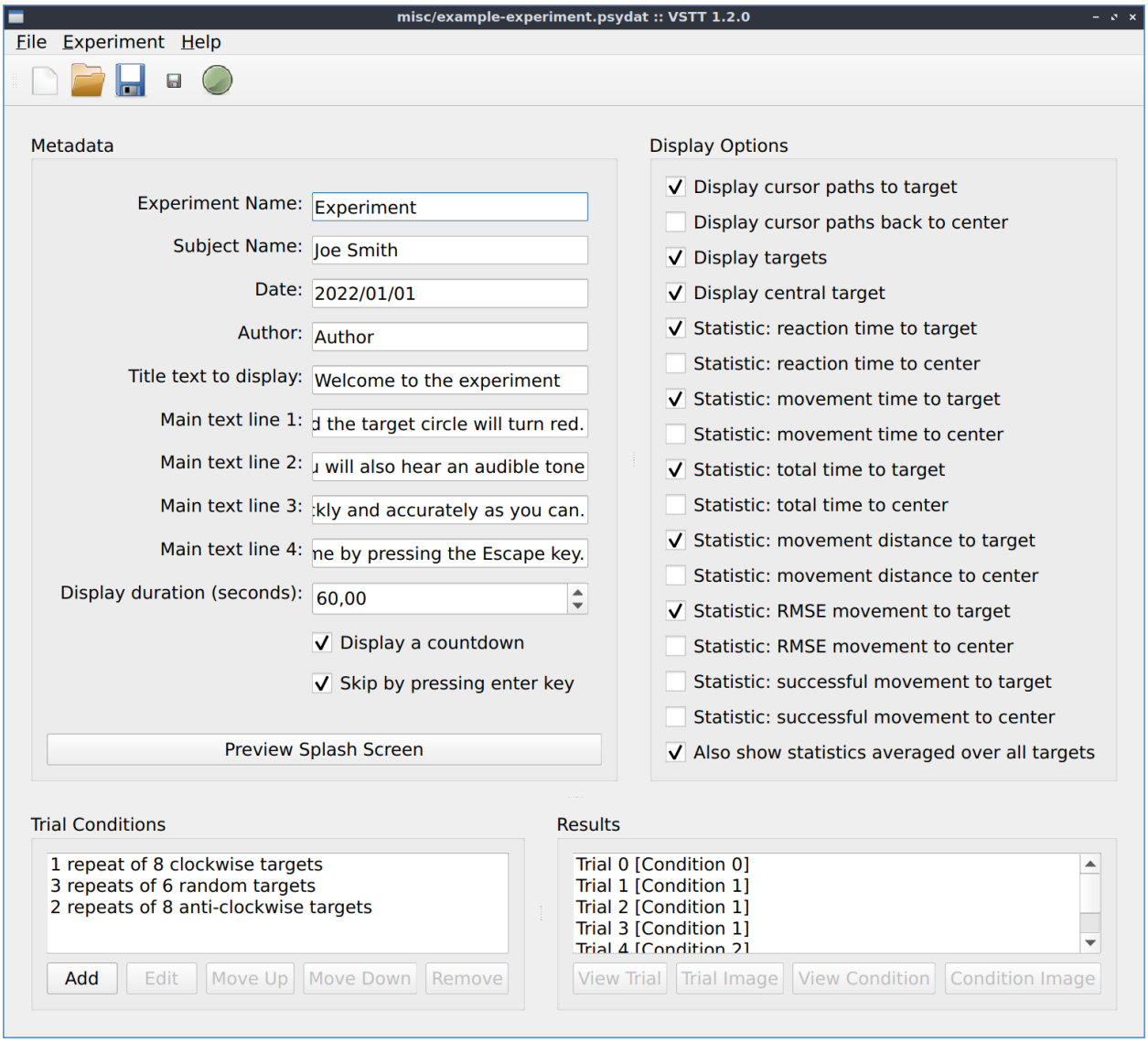


Fig. 1: The main graphical user interface

2.1 Metadata

In the Metadata section you can edit the metadata for the experiment, as well as the text that should be displayed before the experiment begins. To see a preview of what the splash screen will look like, click on "Preview Splash Screen".

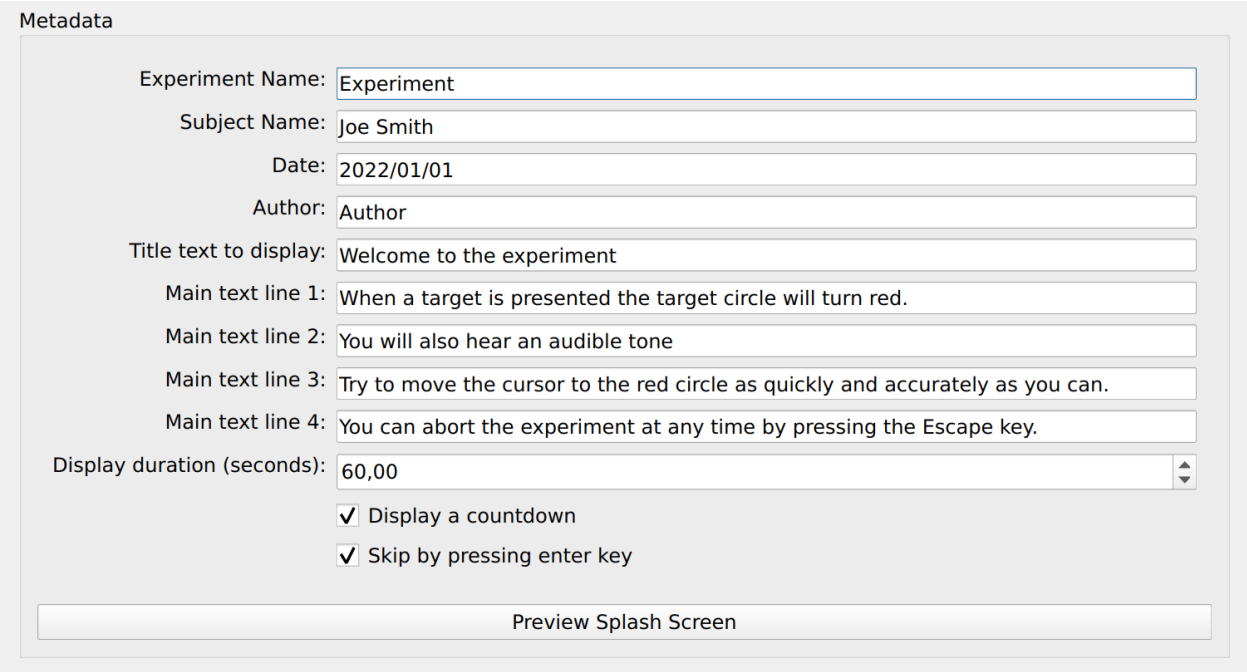


Fig. 2: Metadata: Information about the experiment and text to display before it starts.

2.2 Display Options

You can choose which results and statistics to display in the Display Options section.

2.3 Trial Conditions

The conditions used for the trials are listed here. Conditions can be added, re-ordered, edited and removed using the buttons below the list of trials.

To edit a trial, either click "Edit" or double click on the trial. This will then open a dialog box where the trial conditions can be adjusted. When you are finished, click "OK".

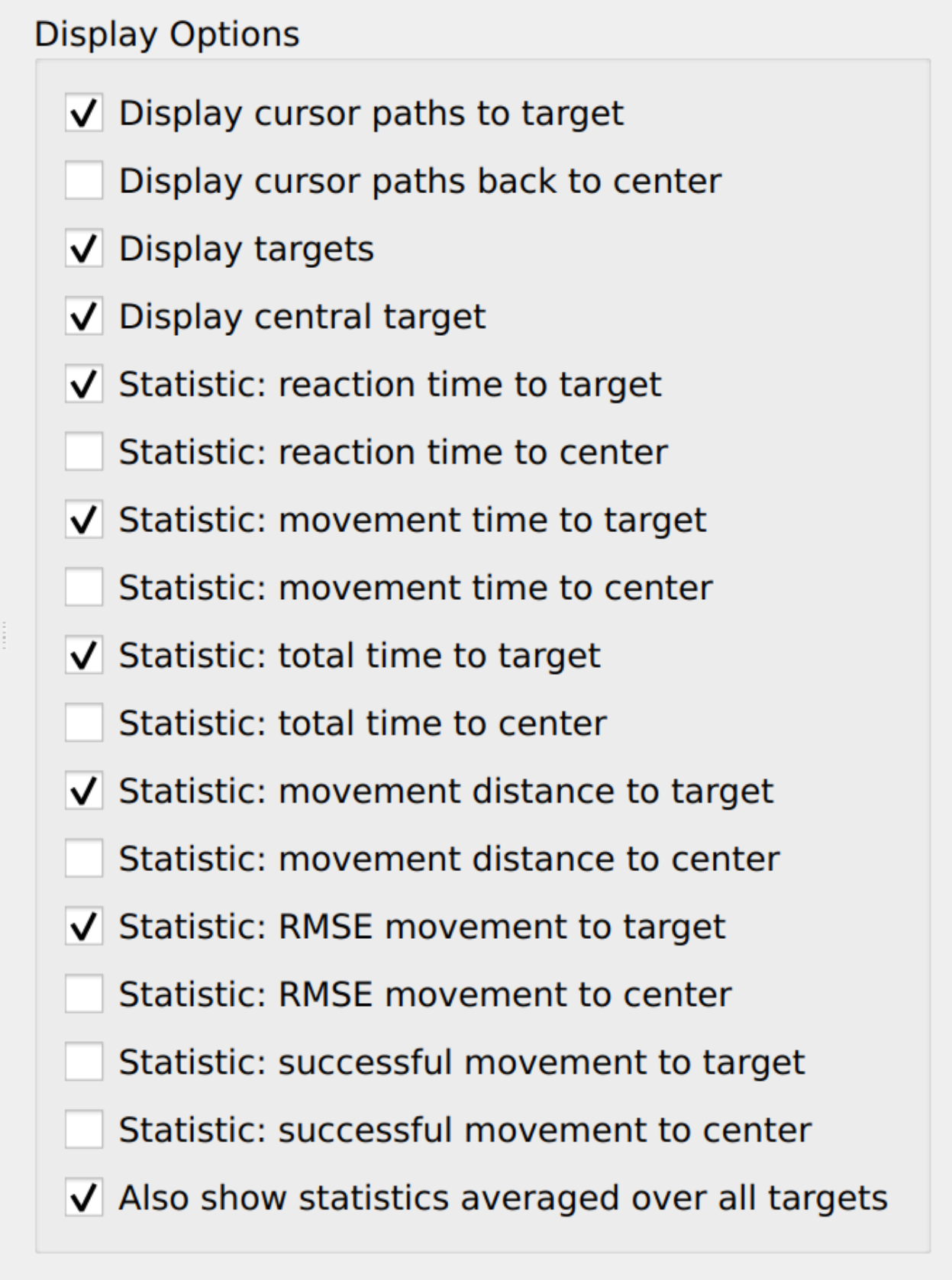


Fig. 3: Display Options: select which results and statistics to display.

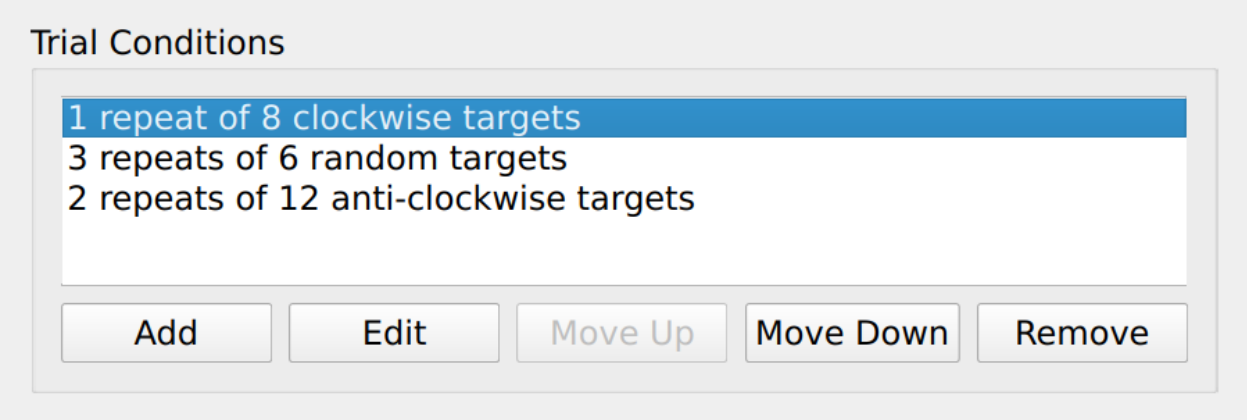


Fig. 4: The list of conditions used in the trial.

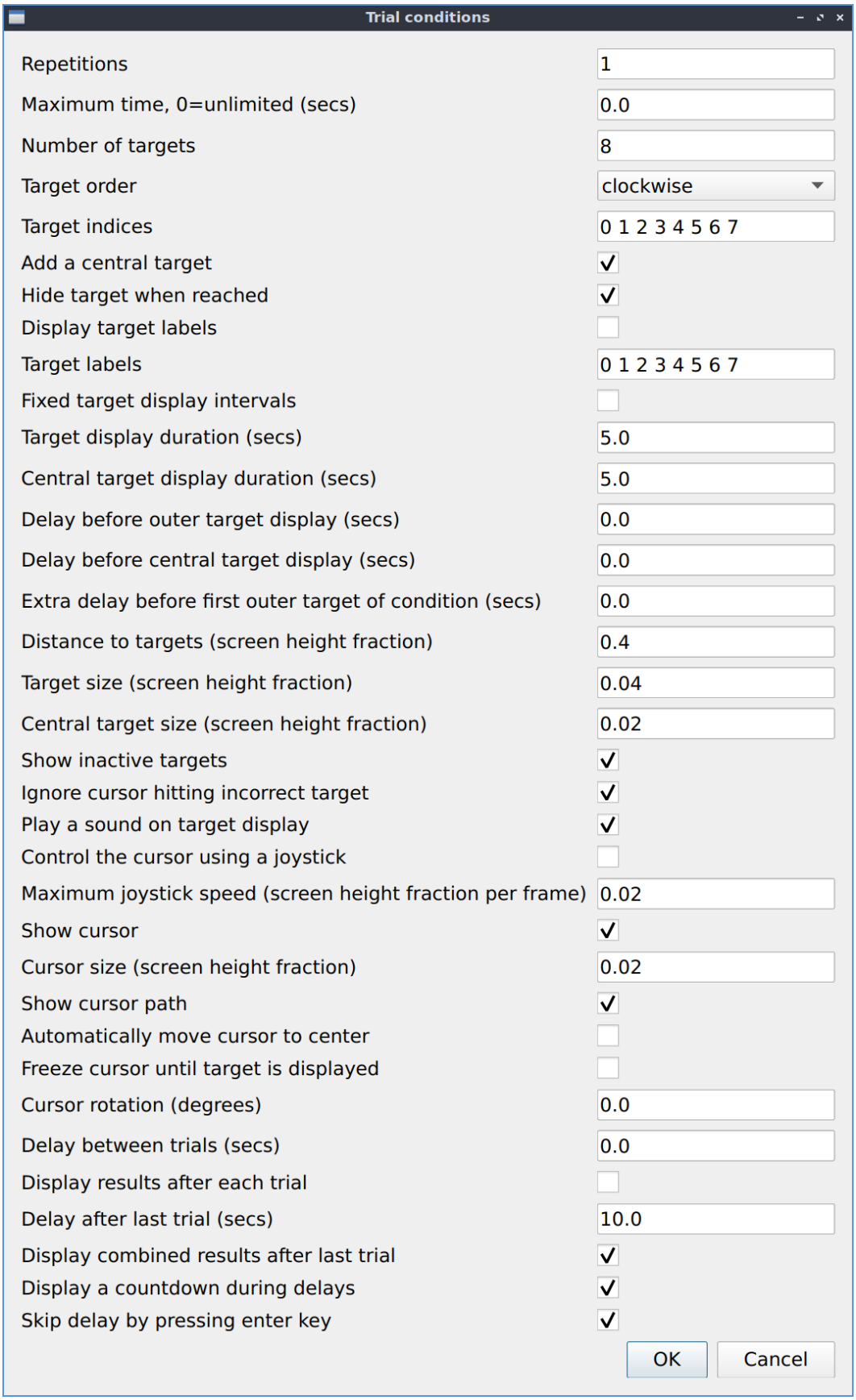


Fig. 5: Dialog to edit the conditions for a trial.

CHAPTER

THREE

RUNNING AN EXPERIMENT

To run an experiment, got to Experiment -> Run, or press `Ctrl+R`, or click on the circular green toolbar button.

3.1 Splash screen

The experiment will then start, first displaying the splash screen.

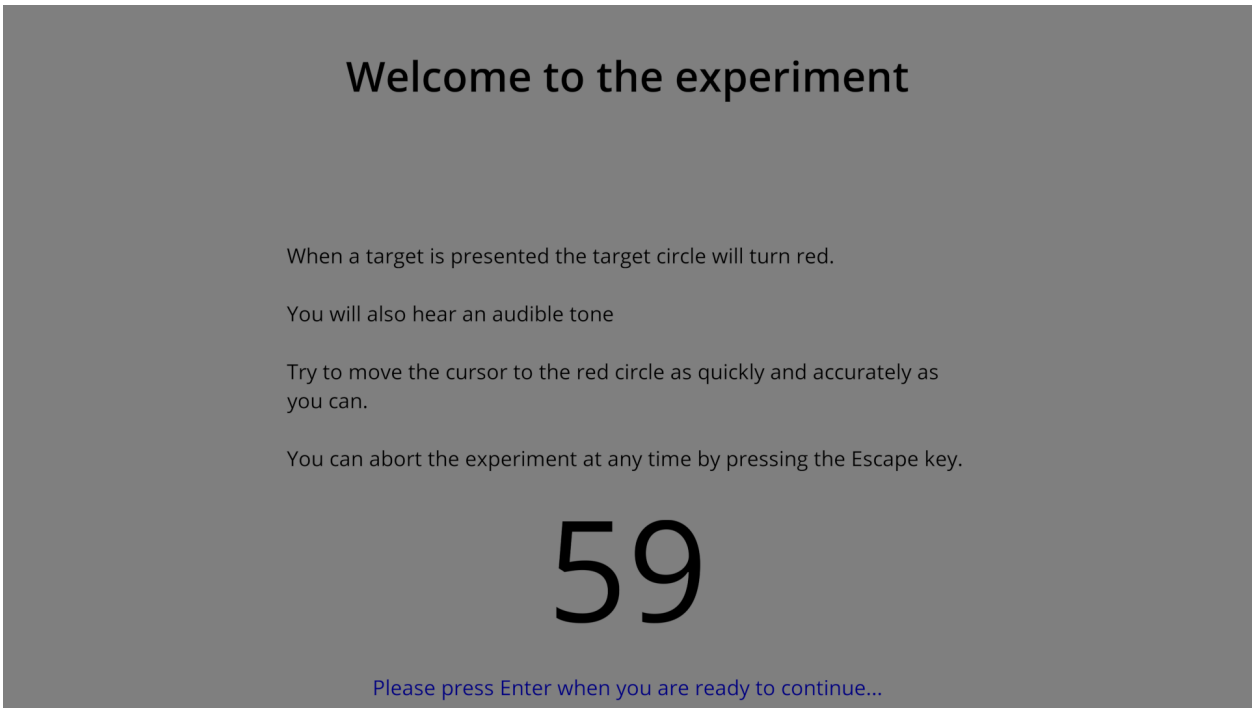


Fig. 1: An example of an experiment splash screen displayed before the experiment begins.

3.2 Trials

Next come the trials.

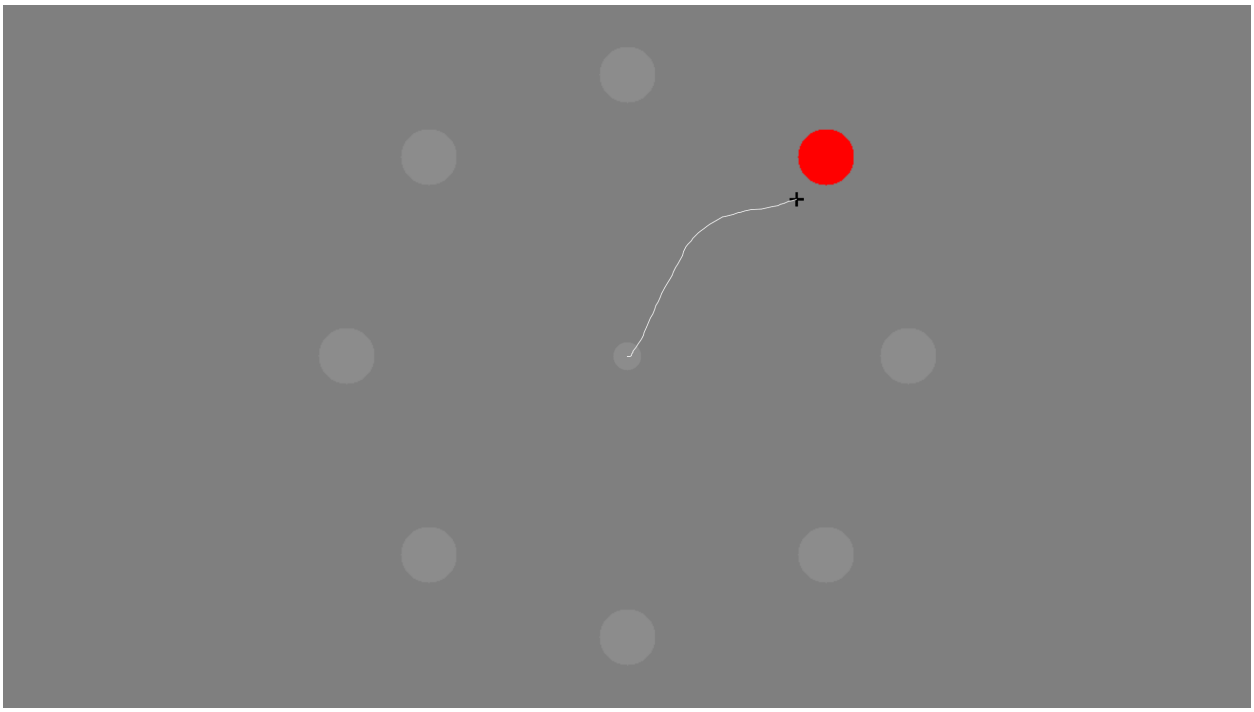


Fig. 2: An example of a trial in progress during an experiment.

3.3 Results

Depending on the trial settings, results may be displayed after a trial or block of trials.

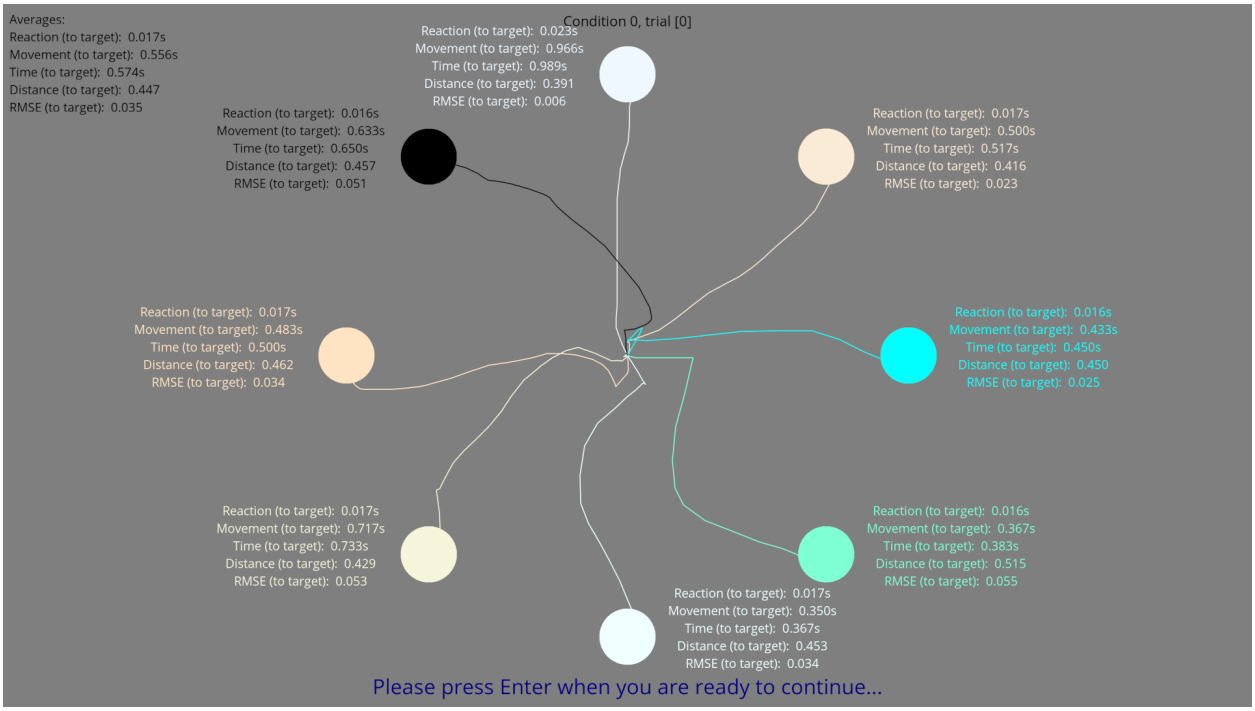


Fig. 3: An example of a results display after a block of trials during an experiment.

CHAPTER

FOUR

DISPLAYING RESULTS

To display the results from a trial, select it from the list and click “View Trial”. Alternatively click “Image Trial” to save a screenshot of these results as a png image file. For combined results from all the trials with the same trial conditions as the selected trial, click “View Condition” or “Image Condition”.

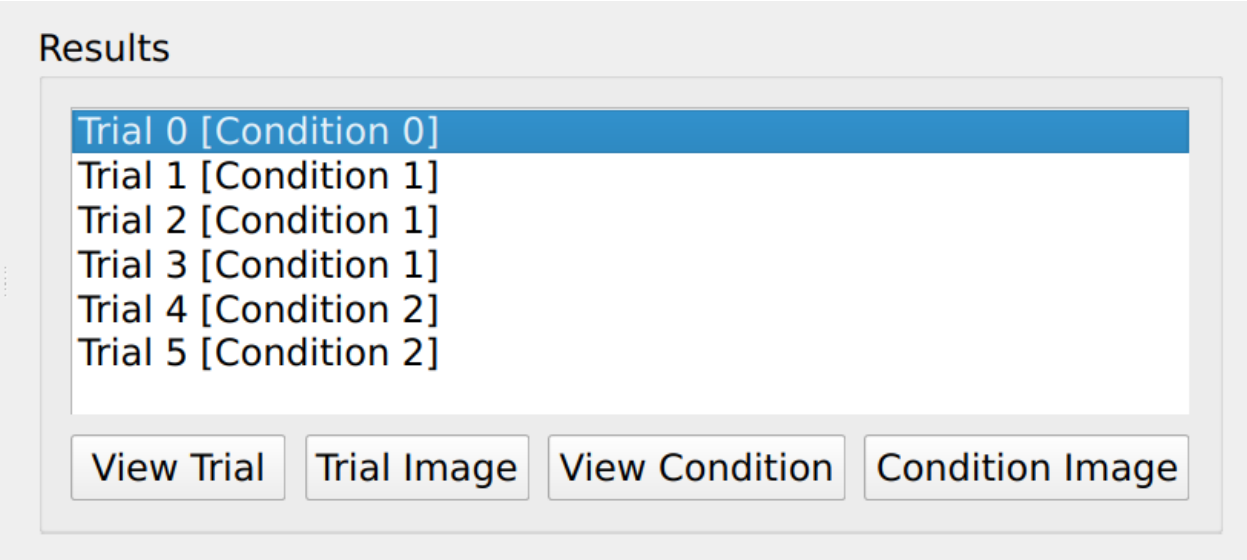


Fig. 1: Results: a list of trials which can be displayed.

4.1 Display options

You can choose which results and statistics to display in the Display Options section.

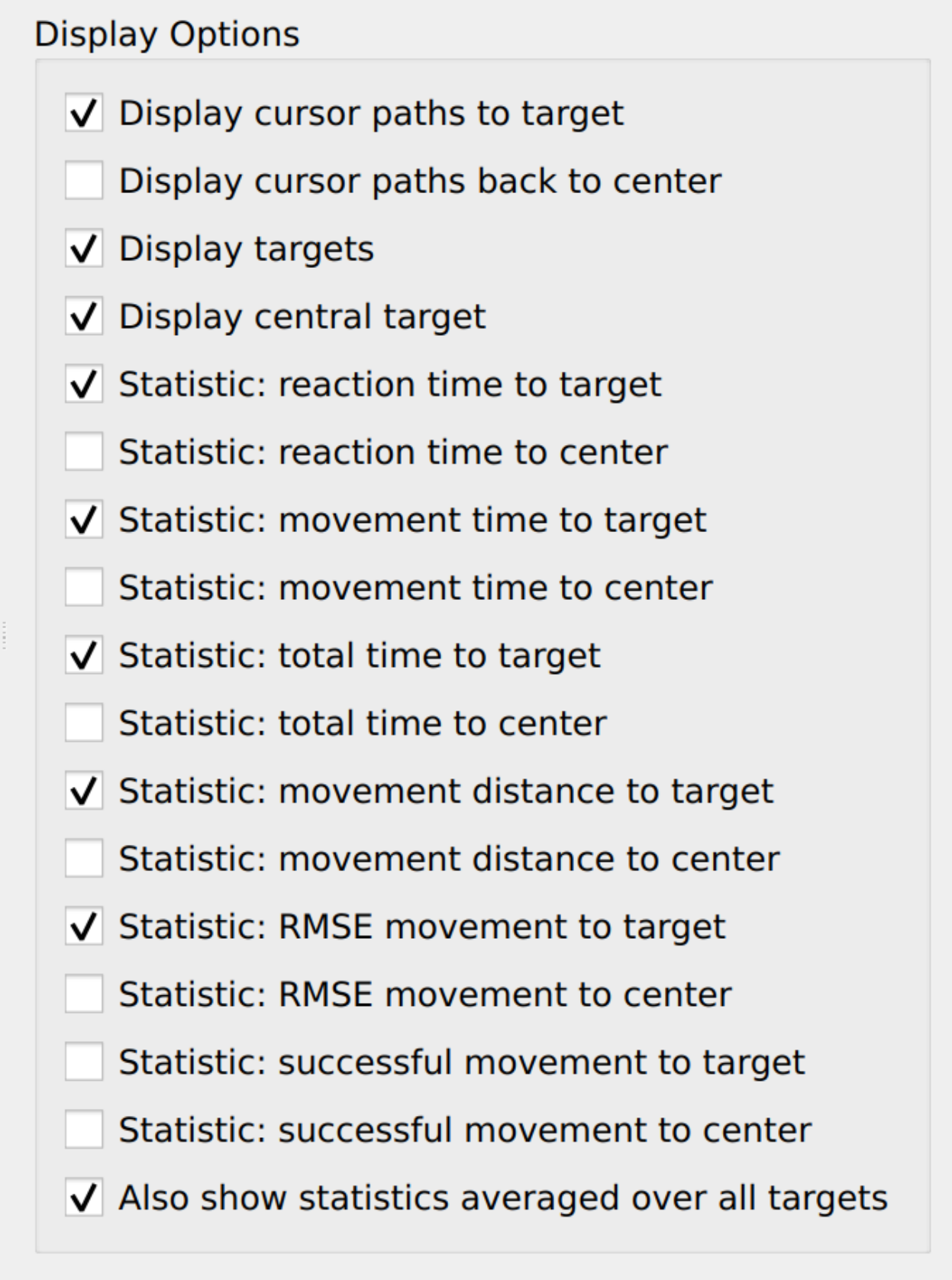


Fig. 2: Display Options: select which results and statistics to display.

4.2 Results

The selected results are then displayed.

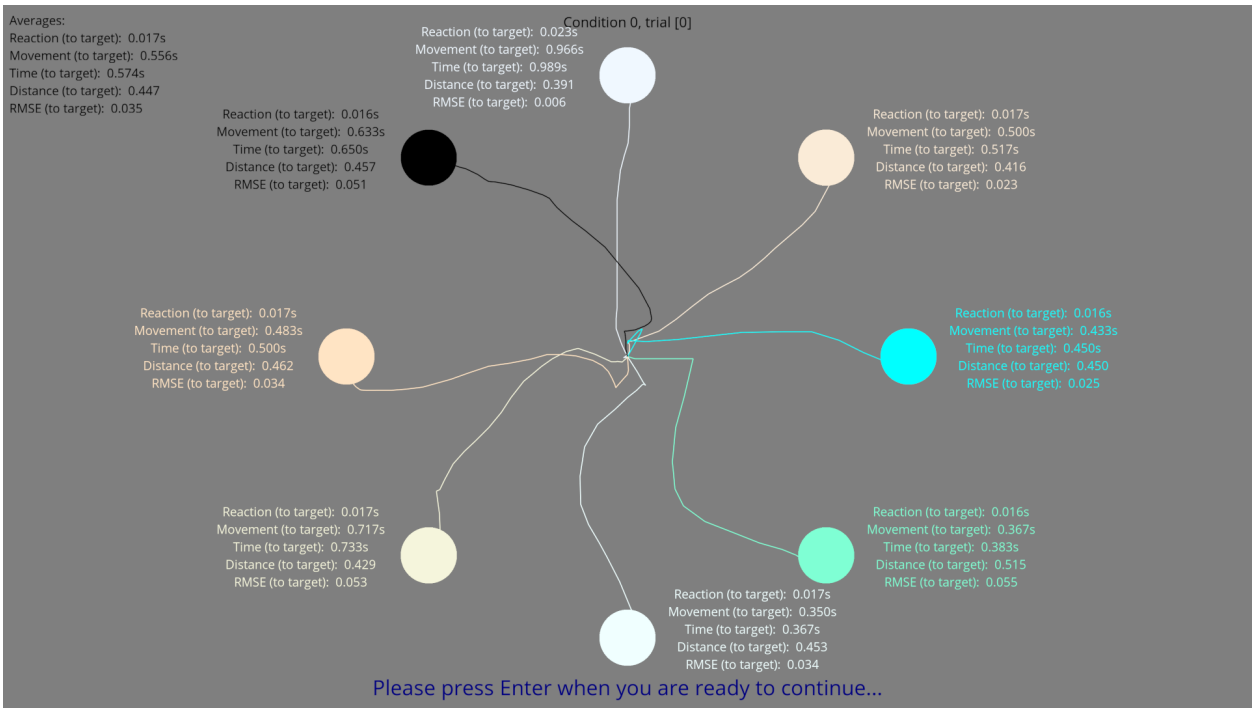


Fig. 3: An example of results from an experiment.

CHAPTER

FIVE

SHARING EXPERIMENTS

There are several ways to share experiment setups and results with others.

If they also install VSTT then you can simply share the .psydat file, which contains both the experimental setup and any results.

You can also export experiments in Excel or json format, which allows experiments and results to be shared, analysed and modified without requiring the VSTT software to do so.

See *File formats* for more information.

CHAPTER

SIX

TRIAL CONDITIONS

There are many conditions that define the behaviour of a set of trials. All time values are in seconds, and all distance values are fractions of the screen height (i.e. a line of length 1.0 in these units would extend from the bottom to the top of the screen).

- **Repetitions**
  - The number of times the trial should be repeated
  - The trial will be repeated this many times (unless the maximum time is exceeded)
  - Default: `1`
- **Maximum time, 0=unlimited (secs)**
  - The maximum time allowed to complete all repeats of these trial conditions
  - If zero, there is no time limit
  - If a trial is in progress when time runs out it is stopped early and no data is recorded for this trial
  - Default: `0`
- **Number of targets**
  - The number of outer targets to display
  - Default: `8`
- **Target order**
  - The order in which targets should be activated
  - Can be `clockwise`, `anti-clockwise`, `random`, or `fixed`
  - If `fixed`, the order is determined by the supplied "Target indices" (see next item)
  - Default: `clockwise`
- **Target indices**
  - An ordered list of target ids to display, separated by spaces
  - Target ids go clockwise starting from 0 at the top of the circle
  - This is only used if `fixed` is chosen for "Target order"
  - Default: `0 1 2 3 4 5 6 7`
- **Add a central target**
  - Enable to show a central target between each outer target
  - Default: `enabled`

- **Hide target when reached**
  - Enable to hide a target once the cursor has reached it
  - Default: `enabled`
- **Display target labels**
  - Enable to show a label inside each target
  - The labels are taken from "Target labels" (see below)
  - Default: `disabled`
- **Target labels**
  - The label to display in each target, separated by spaces
  - Default: `0 1 2 3 4 5 6 7`
- **Fixed target display intervals**
  - Enable to display a new target at fixed intervals
  - The interval is given by "Target display duration" below
  - The interval is not affected by the cursor reaching the target or not
  - If enabled, "Central target display duration" and "Delay between targets" are ignored
  - Default: `disabled`
- **Target display duration (secs)**
  - How long each outer target should be displayed for
  - Default: `5.0`
- **Central target display duration (secs)**
  - How long each central target should be displayed for
  - Default: `5.0`
- **Delay before outer target display (secs)**
  - The delay before each outer target is displayed
  - Default: `0.0`
- **Delay before central target display (secs)**
  - The delay before each central target is displayed
  - Default: `0.0`
- **Extra delay before first outer target of condition (secs)**
  - An additional delay before the first outer target is displayed for this set of conditions
  - Default: `0.0`
- **Distance to targets (screen height fraction)**
  - The distance from the centre of the screen to the outer targets
  - Default: `0.4`
- **Target size (screen height fraction)**
  - Diameter of outer targets

  - Default: `0.04`
- **Central target size (screen height fraction)**
  - Diameter of central targets
  - Default: `0.02`
- **Show inactive targets**
  - Enable to make inactive targets visible (but greyed out)
  - Default: `enabled`
- **Ignore cursor hitting incorrect target**
  - Enable to ignore when a cursor reaches an incorrect/inactive target
  - If disabled, when the cursor reaches any target the task will proceed to the next target
  - Default: `enabled`
- **Play a sound on target display**
  - Enable to play an audible sound when a target is displayed
  - Default: `enabled`
- **Control the cursor using a joystick**
  - Enable to use a joystick to control the cursor
  - If disabled, the mouse will be used instead
  - Default: `disabled`
- **Maximum joystick speed (screen height fraction per frame)**
  - The maximum distance the cursor can move in a single frame
  - Multiply by the monitor refresh rate (fps) to get the speed in screen height fraction per second
  - The refresh rate is often 60fps, but some gaming monitors can have much higher refresh rates.
  - Default: `0.02`
- **Show cursor**
  - Enable to display a cursor at the current cursor location
  - Default: `enabled`
- **Cursor size (screen height fraction)**
  - The size of the cursor
  - Default: `0.02`
- **Show cursor path**
  - Enable to display the path the cursor took
  - Default: `enabled`
- **Automatically move cursor to center**
  - Enable to automatically move the cursor to the center after reaching an outer target
  - Default: `disabled`
- **Freeze cursor until target is displayed**

  - Enable to freeze the cursor until a target is displayed
  - Default: `disabled`
- **Cursor rotation (degrees)**
  - Rotate the cursor direction anticlockwise by this number of degrees
  - Default: `0.0`
- **Delay between trials (secs)**
  - How long to wait after each trial
  - Default: `0.0`
- **Display results after each trial**
  - Enable to display results for the trial after each trial
  - Default: `disabled`
- **Delay after last trial (secs)**
  - How long to wait after the last trial with these trial conditions
  - Default: `10.0`
- **Display combined results after last trial**
  - Enable to display combined results for these trial conditions after the last trial
  - Default: `enabled`
- **Display a countdown during delays**
  - Enable to display a countdown in seconds while waiting between trials
  - Default: `enabled`
- **Skip delay by pressing enter key**
  - Enable to allow the user to skip a delay between trials by pressing the enter key
  - Default: `enabled`

CHAPTER

SEVEN

STATISTICS

VSTT calculates a number of statistics for each trial. These are displayed in the application, are provided in the statistics Pandas DataFrame, and are also contained in the excel data export. Here we provide the definitions used to calculate them.

7.1 Time

Fig. 1: The timepoints of events for a target.

The following timepoints are used to calculate timing statistics:

- t_{start} is when timestamps begin for this target
- $t_{display}$ is when the target is first displayed
- t_{move} is when the cursor is first moved
- t_{final} is when the cursor reaches the target or timeout occurs

The following timing statistics are calculated, all in units of seconds:

- **Time**
  - $t_{final} - t_{display}$
  - Time from target being displayed to the target being reached by the cursor or timeout
- **Reaction time**
  - $t_{move} - t_{display}$
  - Time from target being displayed to first cursor movement
- **Movement time**
  - $t_{final} - t_{move}$
  - Time from first cursor movement to the target being reached by the cursor or timeout

7.2 Distance

Fig. 2: The cursor positions for a target.

The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the height of the screen.

Given n pairs of (x, y) cursor locations, labelled in order from 1 to n, the following statistics are calculated, all in units of screen height:

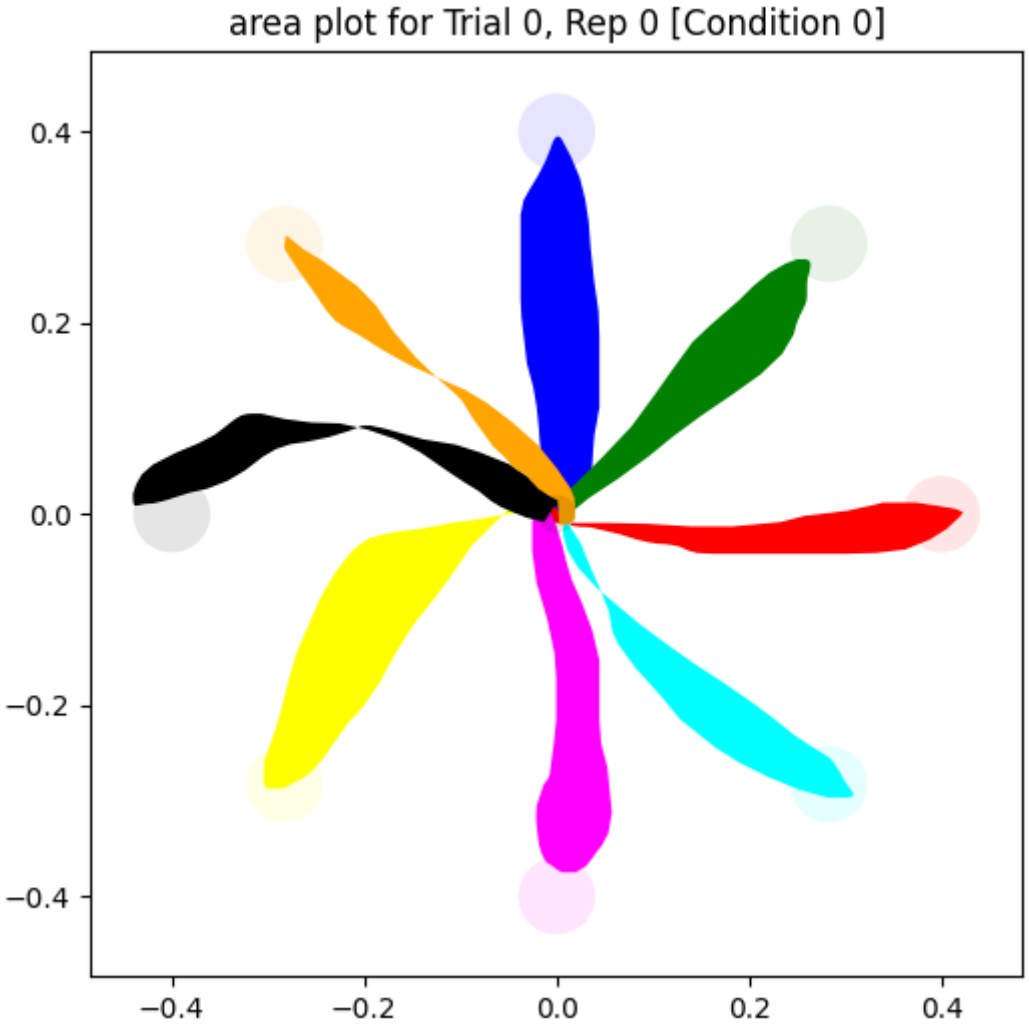
- **Distance**
  - $\sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$
  - Euclidean point-to-point distance travelled by the cursor
- **RMSE**
  - $\sqrt{\frac{\sum_{i=2}^{n}[(x_t - x_1)(y_1 - y_i) - (x_1 - x_i)(y_t - y_1)]^2}{(n-1)[(x_t - x_1)^2 + (y_t - y_1)^2]}}$
  - where (x_t, y_t) is the location of the target
  - Root Mean Square Error (RMSE) from the ideal path to the target
  - this is the RMS of the perpendicular distance of each cursor point from the ideal path
  - where the ideal path is the straight line from the first cursor point to the target
  - based on Distance from a point to a line: Line defined by two points

7.3 Area

The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the height of the screen.

Given pairs of (x, y) cursor locations, the following statistics are calculated, all in units of screen height:

- **Area**
  - get the cursor coordinates of the polygon which is closed by the to target and to center (x, y) cursor locations, use the build-in function to calculate the area of the polygon.
  - In cases where the cursor movement results in intersecting paths, multiple polygons are formed, and their areas are summed.
  - Moreover, when the movement not only intersects but also leads to overlapping regions, the overlapped area is counted twice.
  - e.g.
  -
- **Normalized Area**
  - (the area formed by paths) / (length of the paths)^2


area plot for Trial 0, Rep 0 [Condition 0]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

7.4 Velocity

The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the height of the screen per second.

Given pairs of (x, y) cursor locations, and pairs of t timestamp, the following statistics are calculated, all in units of screen height per second:

- **Velocity**
  - $\frac{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}}{t_{i+1}-t_i}$
  - the rate of change of position with respect to time
- **Peak Velocity**
  - maximum velocity

7.5 Acceleration

The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the screen height per second squared.

Given pairs of (x, y) cursor locations, and pairs of t timestamp, the following statistics are calculated, all in units of screen height per second squared:

- **Acceleration**
  - $\frac{\sqrt{(\frac{x_{i+2}-x_{i+1}}{t_{i+2}-t_{i+1}}-\frac{x_{i+1}-x_i}{t_{i+1}-t_i})^2+(\frac{y_{i+2}-y_{i+1}}{t_{i+2}-t_{i+1}}-\frac{y_{i+1}-y_i}{t_{i+1}-t_i})^2}}{t_{i+1}-t_i}$
  - the rate of change of the velocity of an object with respect to time
- **Peak Acceleration**
  - maximum Acceleration

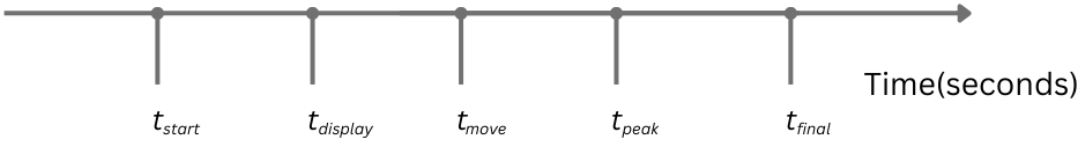
7.6 Spatial Error

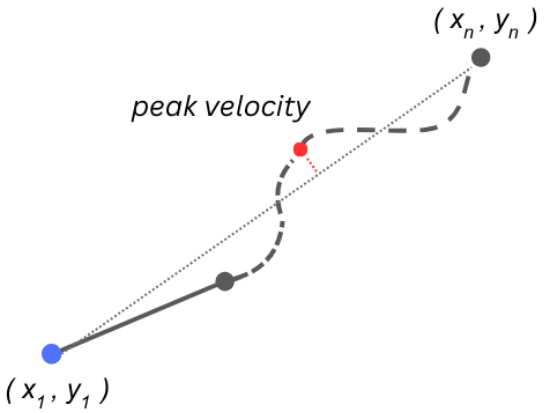
The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the screen height.

Given pairs of (x, y) cursor locations,the following statistics are calculated, all in units of screen height:

- **Spatial Error to target**
  - the distance between the end point of the movement to the center of the target - radius of target
- **Spatial Error to central target**
  - the distance between the end point of the movement to the center of the central target - radius of central target

7.7 Statistics at peak velocity





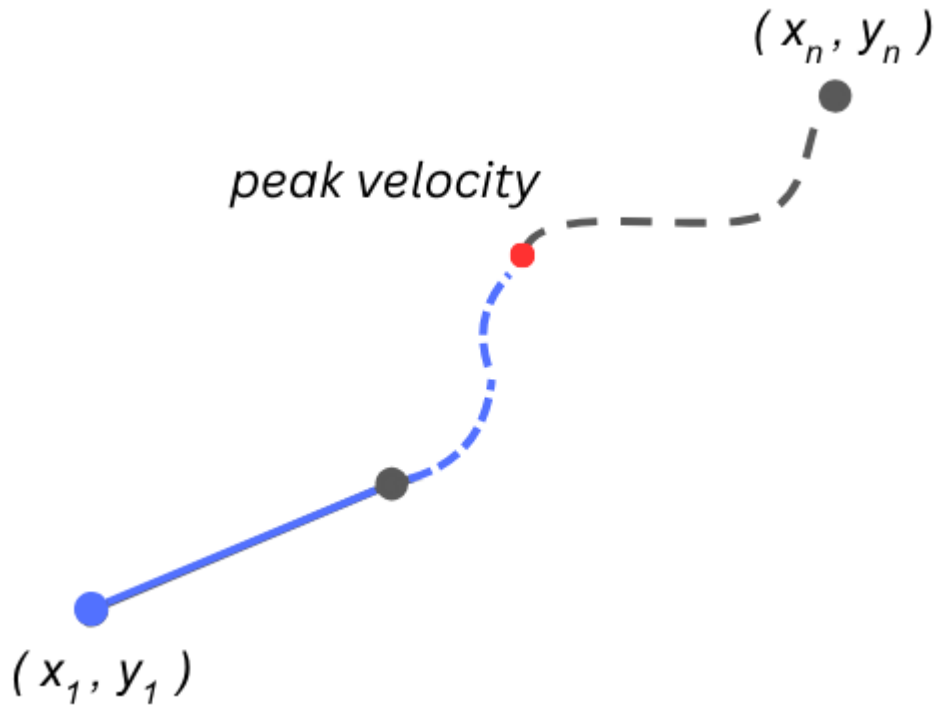
The cursor location at a timestamp is given by a pair of (x, y) coordinates, where $(0, 0)$ corresponds to the center of the screen, and 1 in these units is equal to the screen height.

Given pairs of (x, y) cursor locations,the following statistics are calculated, all in units of screen height:

- **Movement time**
  - $t_{peak} - t_{move}$
  - Time from first cursor movement to the movement at peak velocity
- **Total time**
  - $t_{peak} - t_{display}$

  - Time from target being displayed to the movement at peak velocity

- **Movement distance**



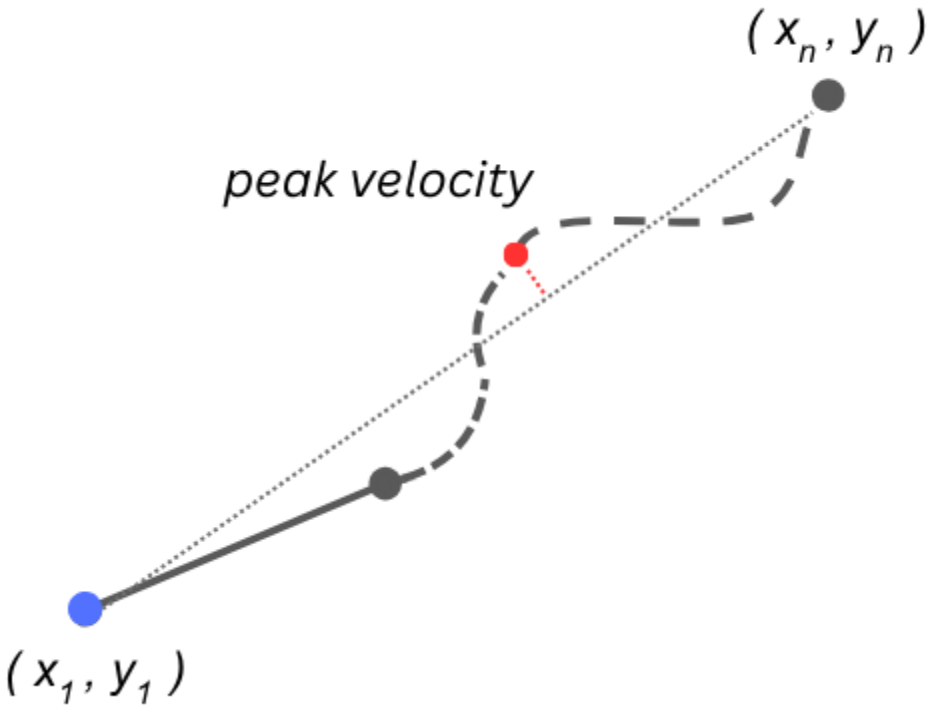
  -

  - Euclidean point-to-point distance travelled from first cursor movement to the peak velocity

- **RMSE movement**

  -

  - Root Mean Square Error (RMSE) of the perpendicular distance from the peak velocity mouse point to the straight line that intersects the first mouse location and the target.


(x_n , y_n)
peak velocity
(x_1 , y_1)

CHAPTER

EIGHT

FRAMERATES AND TIMESTAMPS

8.1 Introduction

During an experiment you see the cursor moving on the screen, but what your screen actually displays is a series of still images, or frames, which are updated quickly enough that we don't notice the individual frames.

Typically computer monitors do this at a rate of 60 frames per second (fps), although gaming monitors are available that offer much higher refresh rates.

The experiment saves information about the cursor position each time a new frame is displayed.

8.2 How an experiment works

This loop happens every time the monitor refreshes:

- store the timestamp and current cursor location
- prepare next frame to display (draw objects such as the cursor, targets, etc)
- wait until the screen is updated with the new frame

In the stored data, each timestamp is the time when the specified target was displayed on the screen, and the cursor location at that time. The display on the screen doesn't change until the next timestamp.

8.3 Dropped frames

The sampling frequency of the experimental data is limited by the refresh rate of the monitor.

In addition, it could sometimes be the case that the code preparing the next frame to be displayed takes too long, and the monitor refreshes the screen before the next frame is available. This is known as a "dropped frame", and will be visible in the recorded data as a larger than usual delay before the next timestamp.

If this happens, ensuring that nothing else is running on the computer when an experiment is running may help.

CHAPTER

NINE

FILE FORMATS

Experiments can be imported and exported in different formats: psydat, Excel, and json. Psydat is the default format used by VSTT, which can also be opened from Python scripts or Jupyter notebooks. The Excel and json formats are provided to allow experiments and results to be shared, analysed and modified without requiring the VSTT software to do so.

Table 1: File format comparison

	psydat	Excel	json
Export experiment	Yes	Yes	Yes
Import experiment	Yes	Yes	Yes
Export results	Yes	Yes	No
Import results	Yes	No	No

9.1 psydat

This is the default file format, which includes the experiment results. These files can be opened in the VSTT GUI or in Python (see the example notebook for more information).

9.2 Excel

Experiments can also be exported as Excel spreadsheets. The first 3 pages in the spreadsheet define the experiment:

- metadata
- display_options
- trial_list

These values can be edited in Excel and then the modified experiment can be opened in the VSTT GUI (File -> Open, choose Excel as filetype). Note that results are not imported from an Excel file.

The excel sheet also contains a statistics page with calculated statistics for each trial. Then there is a sheet of data for each trial in the experiment with the following columns:

- **timestamps**
  - start at 0 for each trial, increase throughout the trial
- **mouse_positions_x**
  - x coordinate of mouse at this timestamp

- **mouse_positions_y**
  - y coordinate of mouse at this timestamp
- **i_target**
  - `-99` if no target is visible
  - `-1` if central target is visible
  - otherwise index (e.g. `0`, `1`, etc) of current displayed target
- **target_x**
  - x coordinate of currently visible target
  - `-99` if no target is visible
- **target_y**
  - y coordinate of currently visible target
  - `-99` if no target is visible

There is also the option to export a separate page of data for each target, so a single trial with 8 targets would result in 8 separate pages of experiment data. In this case for each target the timestamps begin at a negative value, and reach zero when the target is displayed.

9.3 json

Experiments can also be exported as a json text file. This is a convenient format for sharing experimental setups, as it can be easily read and modified in a text editor. These files can also be opened in the VSTT GUI (File -> Open, choose JSON as filetype). The json file contains all the information required to define the experiment:

- metadata
- display_options
- trial_list

This format does not include any statistics or experimental results.

CHAPTER

TEN

CHANGELOG

10.1 [1.4.0] - 2023-12-08

10.1.1 Added

- peak velocity and acceleration statistics #253

10.1.2 Fixed

- additional delay between repetitions of a condition when using fixed target display intervals #256

10.2 [1.3.1] - 2023-11-06

10.2.1 Fixed

- missing dependency for vstt 1.3.0 #254

10.3 [1.3.0] - 2023-09-29

10.3.1 Added

- hand path area statistic and visualization #241
- normalised hand path area statistic #247
- support for psychopy 2023.2.0 #244

10.4 [1.2.0] - 2023-07-07

10.4.1 Added

- optional extra delay before first target of a condition #236
- successful movement to target / center statistics #221

10.4.2 Changed

- cursor position is not updated during the delay between trials #237

10.5 [1.1.0] - 2023-06-27

10.5.1 Added

- option to save screenshot of results #230
- optional delay before central target is displayed #228

10.5.2 Changed

- trial stops when maximum condition time exceeded instead of allowing trial to complete #229

10.6 [1.0.0] - 2023-06-26

First official release.

Changelog format based on Keep a Changelog.

CHAPTER

ELEVEN

ANALYSING YOUR RESULTS

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import vstt
```

11.1 Import

To import an experiment from a psydat file:

[2]:
```
experiment = vstt.Experiment("example.psydat")
```

```
WARNING:root:Key 'area' missing, using default 'False'
WARNING:root:Key 'normalized_area' missing, using default 'False'
WARNING:root:Key 'peak_velocity' missing, using default 'False'
WARNING:root:Key 'peak_acceleration' missing, using default 'False'
WARNING:root:Key 'to_target_spatial_error' missing, using default 'False'
WARNING:root:Key 'to_center_spatial_error' missing, using default 'False'
WARNING:root:Key 'movement_time_at_peak_velocity' missing, using default 'False'
WARNING:root:Key 'total_time_at_peak_velocity' missing, using default 'False'
WARNING:root:Key 'movement_distance_at_peak_velocity' missing, using default 'False'
WARNING:root:Key 'rmse_movement_at_peak_velocity' missing, using default 'False'
```

11.2 Conditions

The trial conditions are in `trial_list`, each element in this list is a dict of trial conditions that defines a trial:

[3]:
```
experiment.trial_list
```

[3]:
```
[{'weight': 2,
  'condition_timeout': 0.0,
  'num_targets': 8,
  'target_order': 'clockwise',
  'target_indices': '0 1 2 3 4 5 6 7',
```

(continues on next page)

(continued from previous page)

```
  'add_central_target': True,
  'hide_target_when_reached': True,
  'show_target_labels': False,
  'target_labels': '0 1 2 3 4 5 6 7',
  'fixed_target_intervals': False,
  'target_duration': 5.0,
  'central_target_duration': 5.0,
  'pre_target_delay': 0.0,
  'pre_central_target_delay': 0.0,
  'pre_first_target_extra_delay': 0.0,
  'target_distance': 0.4,
  'target_size': 0.04,
  'central_target_size': 0.02,
  'show_inactive_targets': True,
  'ignore_incorrect_targets': True,
  'play_sound': True,
  'use_joystick': False,
  'joystick_max_speed': 0.02,
  'show_cursor': True,
  'cursor_size': 0.02,
  'show_cursor_path': True,
  'automove_cursor_to_center': False,
  'freeze_cursor_between_targets': False,
  'cursor_rotation_degrees': 0.0,
  'post_trial_delay': 0.0,
  'post_trial_display_results': False,
  'post_block_delay': 0.0,
  'post_block_display_results': True,
  'show_delay_countdown': True,
  'enter_to_skip_delay': True},
 {'weight': 2,
  'condition_timeout': 0.0,
  'num_targets': 6,
  'target_order': 'random',
  'target_indices': '5 3 1 0 4 2',
  'add_central_target': True,
  'hide_target_when_reached': True,
  'show_target_labels': False,
  'target_labels': '0 1 2 3 4 5 6 7',
  'fixed_target_intervals': False,
  'target_duration': 5.0,
  'central_target_duration': 5.0,
  'pre_target_delay': 1.0,
  'pre_central_target_delay': 0.0,
  'pre_first_target_extra_delay': 0.0,
  'target_distance': 0.4,
  'target_size': 0.04,
  'central_target_size': 0.02,
  'show_inactive_targets': True,
  'ignore_incorrect_targets': True,
  'play_sound': True,
  'use_joystick': False,
```

(continues on next page)

(continued from previous page)

```
  'joystick_max_speed': 0.02,
  'show_cursor': True,
  'cursor_size': 0.02,
  'show_cursor_path': True,
  'automove_cursor_to_center': True,
  'freeze_cursor_between_targets': True,
  'cursor_rotation_degrees': 0.0,
  'post_trial_delay': 0.0,
  'post_trial_display_results': False,
  'post_block_delay': 0.0,
  'post_block_display_results': True,
  'show_delay_countdown': True,
  'enter_to_skip_delay': True}]
```

This can be more easily viewed if converted to a pandas DataFrame:

[4]:
```
pd.DataFrame(experiment.trial_list)
```

[4]:
```
   weight  condition_timeout  num_targets target_order   target_indices  \
0       2                0.0            8    clockwise  0 1 2 3 4 5 6 7
1       2                0.0            6       random      5 3 1 0 4 2

   add_central_target  hide_target_when_reached  show_target_labels  \
0                True                      True               False
1                True                      True               False

     target_labels  fixed_target_intervals  ...  show_cursor_path  \
0  0 1 2 3 4 5 6 7                   False  ...              True
1  0 1 2 3 4 5 6 7                   False  ...              True

   automove_cursor_to_center  freeze_cursor_between_targets  \
0                      False                          False
1                       True                           True

   cursor_rotation_degrees  post_trial_delay  post_trial_display_results  \
0                      0.0               0.0                       False
1                      0.0               0.0                       False

   post_block_delay  post_block_display_results  show_delay_countdown  \
0               0.0                       True                  True
1               0.0                       True                  True

   enter_to_skip_delay
0                 True
1                 True

[2 rows x 35 columns]
```

The `weight` of a trial is how many times it should be repeated.

11.3 Results

The results of the trials are in `stats` which provides both the raw data and calculated statistics as a pandas DataFrame:

[5]:
```
stats = experiment.stats
```

[6]:
```
stats.head()
```

[6]:
```
   i_trial  i_rep  i_target  condition_index  target_index  \
0        0      0         0                0             0
1        0      0         1                0             1
2        0      0         2                0             2
3        0      0         3                0             3
4        0      0         4                0             4

                                target_pos  target_radius  \
0                                [0.0, 0.4]           0.04
1   [0.282842712474619, 0.28284271247461906]           0.04
2              [0.4, 2.4492935982947065e-17]           0.04
3  [0.28284271247461906, -0.282842712474619]           0.04
4              [4.898587196589413e-17, -0.4]           0.04

                              to_target_timestamps  \
0  [0.016512155532836914, 0.03316307067871094, 0...
1  [1.269320011138916, 1.2859880924224854, 1.3026...
2  [2.1527011394500732, 2.16939115524292, 2.18605...
3  [3.1360599994659424, 3.1527769565582275, 3.169...
4  [4.052838087081909, 4.0694849491119385, 4.0861...

                         to_target_mouse_positions to_target_success  ...  \
0  [[0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0...              True  ...
1  [[0.013888888888888888, 0.0032407407407407406]...              True  ...
2  [[-0.0032407407407407406, 0.008796296296296297...              True  ...
3  [[0.014351851851851852, -0.012037037037037037]...              True  ...
4  [[-0.0125, 0.01712962962962963], [-0.017592592...              True  ...

   to_center_rmse to_center_spatial_error      area normalized_area  \
0        0.033841                       0  0.023840        0.034475
1        0.017958                       0  0.016878        0.026436
2        0.030604                       0  0.012411        0.014747
3        0.040075                       0  0.015122        0.018269
4        0.038653                       0  0.018386        0.025205

  peak_velocity peak_acceleration  movement_time_at_peak_velocity  \
0      2.223369         76.701244                        0.269374
1      2.546450        119.925339                        0.733372
2      3.017843         90.854791                        0.249999
3      3.348506         94.232559                        0.266654
4      2.846187         96.380859                        0.750033

   total_time_at_peak_velocity  movement_distance_at_peak_velocity  \
0                     0.602779                            0.257532
1                     0.750040                            0.626939
```

(continues on next page)

(continued from previous page)

```
2                     0.266689                                0.217018
3                     0.283371                                0.086299
4                     0.766680                                0.507352

   rmse_movement_at_peak_velocity
0                        0.036990
1                        0.021135
2                        0.015429
3                        0.026286
4                        0.061012

[5 rows x 37 columns]
```

- each row in this DataFrame contains the data from a single movement to a target (and optionally back to the center)
- they are in the order that they were shown in the experiment
- `condition_index` is the index of the corresponding trial conditions in `trial_index`

[7]:
```
stats.describe()
```

[7]:
```
         i_trial  i_rep   i_target  condition_index  target_index  \
count  28.000000   28.0  28.000000        28.000000     28.000000
mean    1.357143    0.0   3.071429         0.428571      3.071429
std     1.129218    0.0   2.159022         0.503953      2.159022
min     0.000000    0.0   0.000000         0.000000      0.000000
25%     0.000000    0.0   1.000000         0.000000      1.000000
50%     1.000000    0.0   3.000000         0.000000      3.000000
75%     2.000000    0.0   5.000000         1.000000      5.000000
max     3.000000    0.0   7.000000         1.000000      7.000000

       target_radius  to_target_num_timestamps_before_visible  center_radius  \
count          28.00                                28.000000          28.00
mean            0.04                                25.464286           0.02
std             0.00                                29.944989           0.00
min             0.04                                 0.000000           0.02
25%             0.04                                 0.000000           0.02
50%             0.04                                 0.000000           0.02
75%             0.04                                59.000000           0.02
max             0.04                                60.000000           0.02

       to_center_num_timestamps_before_visible  to_target_distance  ...  \
count                                     28.0           28.000000  ...
mean                                       0.0            0.414450  ...
std                                        0.0            0.050779  ...
min                                        0.0            0.364467  ...
25%                                        0.0            0.381896  ...
50%                                        0.0            0.397624  ...
75%                                        0.0            0.420149  ...
max                                        0.0            0.562351  ...

       to_center_rmse  to_center_spatial_error       area  normalized_area  \
count       16.000000                     28.0  28.000000        28.000000
```

(continues on next page)

(continued from previous page)

```
mean          0.030781                      0.0   0.014558          0.019241
std           0.017339                      0.0   0.007793          0.009220
min           0.007777                      0.0   0.003810          0.006553
25%           0.018277                      0.0   0.009981          0.013583
50%           0.031392                      0.0   0.012561          0.018035
75%           0.039008                      0.0   0.017399          0.022368
max           0.077367                      0.0   0.035034          0.042204

       peak_velocity  peak_acceleration  movement_time_at_peak_velocity  \
count      28.000000          28.000000                       28.000000
mean        2.746698          88.582911                        0.349513
std         0.651132          23.619614                        0.290741
min         1.761711          55.124790                        0.000000
25%         2.234549          73.140230                        0.195854
50%         2.671863          82.988582                        0.266655
75%         3.192919          96.768431                        0.362553
max         4.206990         136.721870                        1.166727

       total_time_at_peak_velocity  movement_distance_at_peak_velocity  \
count                    27.000000                           28.000000
mean                      0.407524                            0.236214
std                       0.276798                            0.213964
min                       0.099994                            0.000000
25%                       0.241679                            0.085500
50%                       0.283350                            0.156280
75%                       0.501398                            0.261430
max                       1.183390                            0.841735

       rmse_movement_at_peak_velocity
count                       28.000000
mean                         0.030708
std                          0.024542
min                          0.000000
25%                          0.015340
50%                          0.021695
75%                          0.040291
max                          0.091925

[8 rows x 29 columns]
```

11.4 Plot statistics

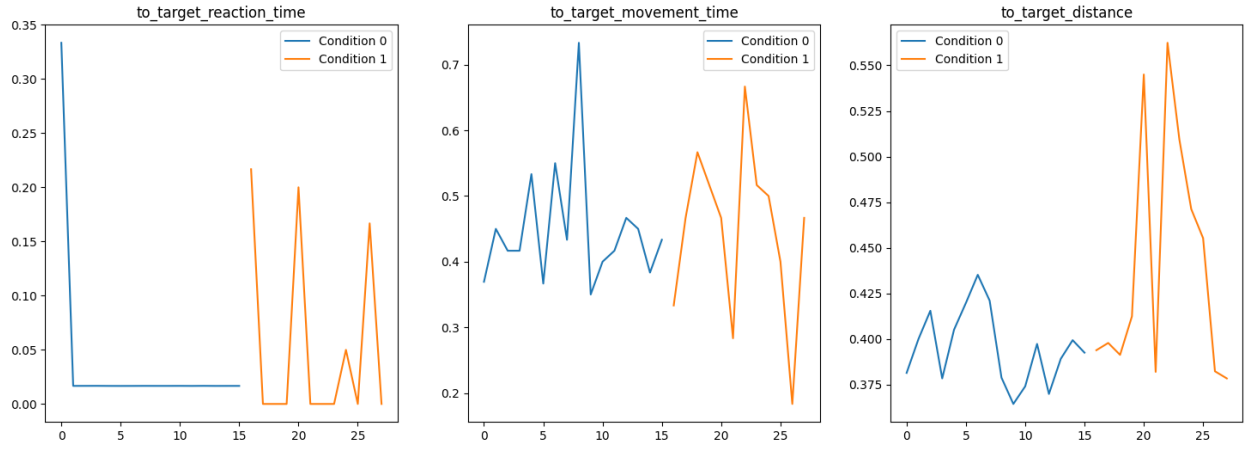
Here we plot a few of the calculated statics for each target, identifying each set of conditions with a different color:

[8]:
```
fig, axs = plt.subplots(ncols=3, figsize=(18, 6))
for ax, statistic in zip(
    axs, ["to_target_reaction_time", "to_target_movement_time", "to_target_distance"]
):
    ax.set_title(f"{statistic}")
    for condition_index, df in stats.groupby("condition_index"):
```

(continues on next page)

(continued from previous page)

```
        ax.plot(df.index, df[statistic], label=f"Condition {condition_index}")
    ax.legend()
plt.show()
```



[ ]:

CHAPTER

TWELVE

WORKING DIRECTLY WITH PSYDAT FILES

The recommended way to analyse your data in Python is to use the provided pandas DataFrame of data and statistics.

However, if you need to you can access the raw data from which these DataFrames are constructed directly as shown in the examples below.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from psychopy.misc import fromFile
from shapely.geometry import LineString
from shapely.geometry import Polygon
from shapely.ops import polygonize
from shapely.ops import unary_union
```

12.1 Import

A psydat file can be imported using the psychopy `fromFile` function:

[2]:
```
psydata = fromFile("example.psydat")
```

12.2 Contents

This returns a Python object that contains all of the trial conditions and results as attributes. These attributes can be listed using the Python `vars` function:

[3]:
```
for var in vars(psydata):
    print(var, end=" | ")
```

```
name | autoLog | trialList | nReps | trialWeights | nTotal | nRemaining | method |␣
↪thisRepN | thisTrialN | thisN | thisIndex | thisTrial | finished | extraInfo | seed |␣
↪data | sequenceIndices | originPath | origin | _exp |
```

12.3 Conditions

The trial conditions are in `trialList`, each element in this list is a dict of trial conditions that defines a trial:

[4]:
```
psydata.trialList
```

[4]:
```
[{'weight': 2,
  'condition_timeout': 0.0,
  'num_targets': 8,
  'target_order': 'clockwise',
  'target_indices': '0 1 2 3 4 5 6 7',
  'add_central_target': True,
  'hide_target_when_reached': True,
  'show_target_labels': False,
  'target_labels': '0 1 2 3 4 5 6 7',
  'fixed_target_intervals': False,
  'target_duration': 5.0,
  'central_target_duration': 5.0,
  'pre_target_delay': 0.0,
  'pre_central_target_delay': 0.0,
  'pre_first_target_extra_delay': 0.0,
  'target_distance': 0.4,
  'target_size': 0.04,
  'central_target_size': 0.02,
  'show_inactive_targets': True,
  'ignore_incorrect_targets': True,
  'play_sound': True,
  'use_joystick': False,
  'joystick_max_speed': 0.02,
  'show_cursor': True,
  'cursor_size': 0.02,
  'show_cursor_path': True,
  'automove_cursor_to_center': False,
  'freeze_cursor_between_targets': False,
  'cursor_rotation_degrees': 0.0,
  'post_trial_delay': 0.0,
  'post_trial_display_results': False,
  'post_block_delay': 0.0,
  'post_block_display_results': True,
  'show_delay_countdown': True,
  'enter_to_skip_delay': True},
 {'weight': 2,
  'condition_timeout': 0.0,
  'num_targets': 6,
  'target_order': 'random',
  'target_indices': '2 5 3 0 1 4',
  'add_central_target': True,
  'hide_target_when_reached': True,
  'show_target_labels': False,
  'target_labels': '0 1 2 3 4 5 6 7',
  'fixed_target_intervals': False,
  'target_duration': 5.0,
  'central_target_duration': 5.0,
```

(continues on next page)

(continued from previous page)

```
  'pre_target_delay': 1.0,
  'pre_central_target_delay': 0.0,
  'pre_first_target_extra_delay': 0.0,
  'target_distance': 0.4,
  'target_size': 0.04,
  'central_target_size': 0.02,
  'show_inactive_targets': True,
  'ignore_incorrect_targets': True,
  'play_sound': True,
  'use_joystick': False,
  'joystick_max_speed': 0.02,
  'show_cursor': True,
  'cursor_size': 0.02,
  'show_cursor_path': True,
  'automove_cursor_to_center': True,
  'freeze_cursor_between_targets': True,
  'cursor_rotation_degrees': 0.0,
  'post_trial_delay': 0.0,
  'post_trial_display_results': False,
  'post_block_delay': 0.0,
  'post_block_display_results': True,
  'show_delay_countdown': True,
  'enter_to_skip_delay': True}]
```

This can be more easily viewed if converted to a pandas DataFrame:

[5]:
```
pd.DataFrame(psydata.trialList)
```

[5]:
```
   weight  condition_timeout  num_targets target_order   target_indices  \
0       2                0.0            8    clockwise  0 1 2 3 4 5 6 7
1       2                0.0            6       random      2 5 3 0 1 4

   add_central_target  hide_target_when_reached  show_target_labels  \
0                True                      True               False
1                True                      True               False

     target_labels  fixed_target_intervals  ...  show_cursor_path  \
0  0 1 2 3 4 5 6 7                   False  ...              True
1  0 1 2 3 4 5 6 7                   False  ...              True

   automove_cursor_to_center  freeze_cursor_between_targets  \
0                      False                          False
1                       True                           True

   cursor_rotation_degrees  post_trial_delay  post_trial_display_results  \
0                      0.0               0.0                       False
1                      0.0               0.0                       False

   post_block_delay  post_block_display_results  show_delay_countdown  \
0               0.0                       True                  True
1               0.0                       True                  True

   enter_to_skip_delay
```

(continues on next page)

(continued from previous page)

```
0                True
1                True

[2 rows x 35 columns]
```

The `weight` of a trial is how many times it should be repeated. This information is also stored in the `trialWeights` list, so for example `trialList[i]` will be repeated `trialWeights[i]` times

[6]:
```
psydata.trialWeights
```

[6]:
```
[2, 2]
```

A block consists of doing a trial with each condition in `trialList` `weight` times, where `weight` can be a different number for each trial, so the total number of trials done in a block is then given by

$$nTrials = \sum_{i}^{nConditions} weight[i]$$

This block is then repeared `nReps` times:

[7]:
```
psydata.nReps
```

[7]:
```
1
```

The total number of trials `nTotal` is then given by

$$nTotal = nReps \times \left(\sum_{i}^{nConditions} weight[i] \right)$$

[8]:
```
psydata.nTotal
```

[8]:
```
4
```

The condition used for a given trial `iTrial` and repetition number `iRep` is given by `sequenceIndices[iTrial][iRep]`, which gives the index of the conditions used in the `trialList` list:

[9]:
```
psydata.sequenceIndices
```

[9]:
```
array([[0],
       [0],
       [1],
       [1]])
```

The `method` specifies the order in which the trials were done:

- sequential
  - the same order as `trialList`
- random
  - order of trials shuffled within each block
- fullRandom
  - order of trials fully shuffled

[10]:
```
psydata.method
```

[10]:
```
'sequential'
```

12.4 Results

The results of the trials are in `data` which contains a dict of numpy arrays of recorded data:

[11]:
```
for key, value in psydata.data.items():
    print(key, value.shape)
```

```
ran (4, 1)
order (4, 1)
target_indices (4, 1)
target_pos (4, 1)
to_target_timestamps (4, 1)
to_target_num_timestamps_before_visible (4, 1)
to_center_timestamps (4, 1)
to_center_num_timestamps_before_visible (4, 1)
to_target_mouse_positions (4, 1)
to_center_mouse_positions (4, 1)
to_target_success (4, 1)
to_center_success (4, 1)
```

[12]:
```
colors = ["blue", "green", "red", "cyan", "magenta", "yellow", "black", "orange"]
nTrials, nReps = psydata.sequenceIndices.shape


def get_axs_row():
    row = 0
    for condition in range(len(psydata.trialList)):
        if not psydata.trialList[condition]["automove_cursor_to_center"]:
            row += 2 * psydata.trialWeights[condition]
        else:
            row += psydata.trialWeights[condition]
    return row


def get_fig_height():
    add = 0
    for condition in range(len(psydata.trialList)):
        if not psydata.trialList[condition]["automove_cursor_to_center"]:
            add += psydata.trialWeights[condition]
    return 6 * (nTrials + add) * nReps


fig, axs = plt.subplots(get_axs_row(), nReps, figsize=(6, get_fig_height()))
axs = np.reshape(
    axs, (get_axs_row(), nReps)
)  # ensure axs is a 2d-array even if nTrials or nReps is 1
for trial in range(nTrials):
```

(continues on next page)

(continued from previous page)

```
    for rep in range(nReps):
        loc = (trial, rep)
        condition = psydata.sequenceIndices[loc]
        target_radius = psydata.trialList[condition]["target_size"]
        central_target_radius = psydata.trialList[condition]["central_target_size"]
        ax = axs[loc]
        ax.set_title(f"Trial {trial}, Rep {rep} [Condition {condition}]")
        for positions, target_pos, color in zip(
            psydata.data["to_target_mouse_positions"][loc],
            psydata.data["target_pos"][loc],
            colors,
        ):
            ax.plot(positions[:, 0], positions[:, 1], color=color)
            ax.add_patch(
                plt.Circle(
                    target_pos,
                    target_radius,
                    edgecolor="none",
                    facecolor=color,
                    alpha=0.1,
                )
            )

        # if condition "automove_cursor_to_center" is deselected, plot the line to␣
↪center, fill the enclosed area and output the area
        if not psydata.trialList[condition]["automove_cursor_to_center"]:
            loc_area = (trial + nTrials, rep)
            ax_area = axs[loc_area]
            ax_area.set_xbound(-0.5, 0.5)
            ax_area.set_ybound(-0.5, 0.5)
            ax_area.set_title(
                f"area plot for Trial {trial}, Rep {rep} [Condition {condition}]"
            )
            print("-----------------------------------------------")
            print(f"area of Trial {trial}, Rep {rep} [Condition {condition}]")
            for (
                to_target_mouse_positions,
                to_center_mouse_positions,
                target_pos,
                color,
            ) in zip(
                psydata.data["to_target_mouse_positions"][loc],
                psydata.data["to_center_mouse_positions"][loc],
                psydata.data["target_pos"][loc],
                colors,
            ):
                ax.plot(
                    to_center_mouse_positions[:, 0],
                    to_center_mouse_positions[:, 1],
                    color=color,
                )
                coords = np.concatenate(
```

(continues on next page)

(continued from previous page)

```
                    (to_target_mouse_positions, to_center_mouse_positions)
                )
                polygon = Polygon(coords)
                lr_coords = np.concatenate((coords[:], to_target_mouse_positions[0:1]))
                lr = LineString(lr_coords)
                validation = lr.is_valid
                multi_LineString = unary_union(lr)
                area = 0
                for pg in polygonize(multi_LineString):
                    area += pg.area
                    ax_area.plot(*pg.exterior.xy, color=color)
                    ax_area.fill(*pg.exterior.xy, facecolor=color)
                ax_area.add_patch(
                    plt.Circle(
                        target_pos,
                        target_radius,
                        edgecolor="none",
                        facecolor=color,
                        alpha=0.1,
                    )
                )
                print(f"{color}, area: {area}")

            ax.add_patch(
                plt.Circle(
                    [0, 0],
                    central_target_radius,
                    edgecolor="none",
                    facecolor="black",
                    alpha=0.1,
                )
            )

plt.show()
```

```
---------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
```

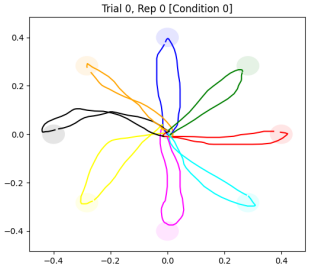
```
blue, area: 0.02384012774348422
green, area: 0.016878322187928677
red, area: 0.01241100823045268
cyan, area: 0.015122099828252057
magenta, area: 0.01838616683813443
yellow, area: 0.026584183527663462
black, area: 0.01402879923061912
orange, area: 0.009765195843465515
```

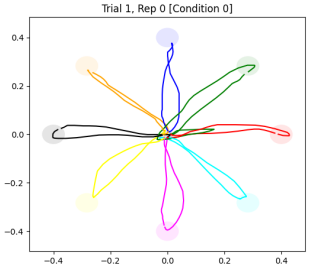
```
---------------------------------------------
area of Trial 1, Rep 0 [Condition 0]
blue, area: 0.012710585484282402
green, area: 0.022543002158882722
red, area: 0.008019782096372857
cyan, area: 0.008811416635490702
```

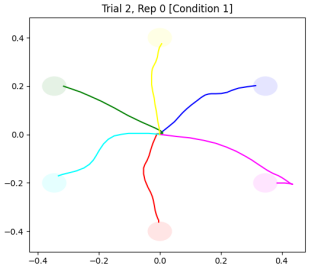
(continues on next page)

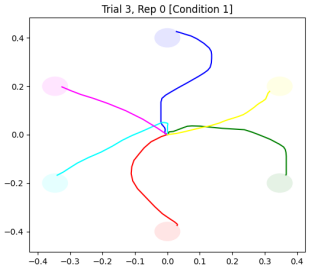
(continued from previous page)

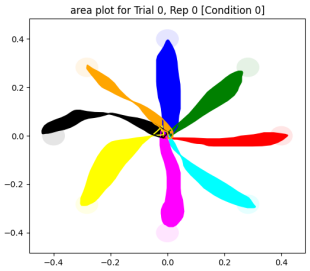
```
magenta, area: 0.014736532418141348
yellow, area: 0.014034243031060156
black, area: 0.011178626543209877
orange, area: 0.004766468942901244
```

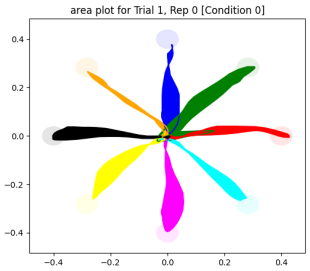

Trial 0, Rep 0 [Condition 0]


Trial 1, Rep 0 [Condition 0]


Trial 2, Rep 0 [Condition 1]


Trial 3, Rep 0 [Condition 1]


area plot for Trial 0, Rep 0 [Condition 0]


area plot for Trial 1, Rep 0 [Condition 0]

Each of these is a `nTrials` x `nReps` 2d array, where each element of this array contains the results from the corresponding trial for this variable (which might itself be a single value, e.g. `target_pos`, or an array of values, e.g. `timestamps`)

Which set of conditions was used is given by the `sequenceIndices` entry in the same location

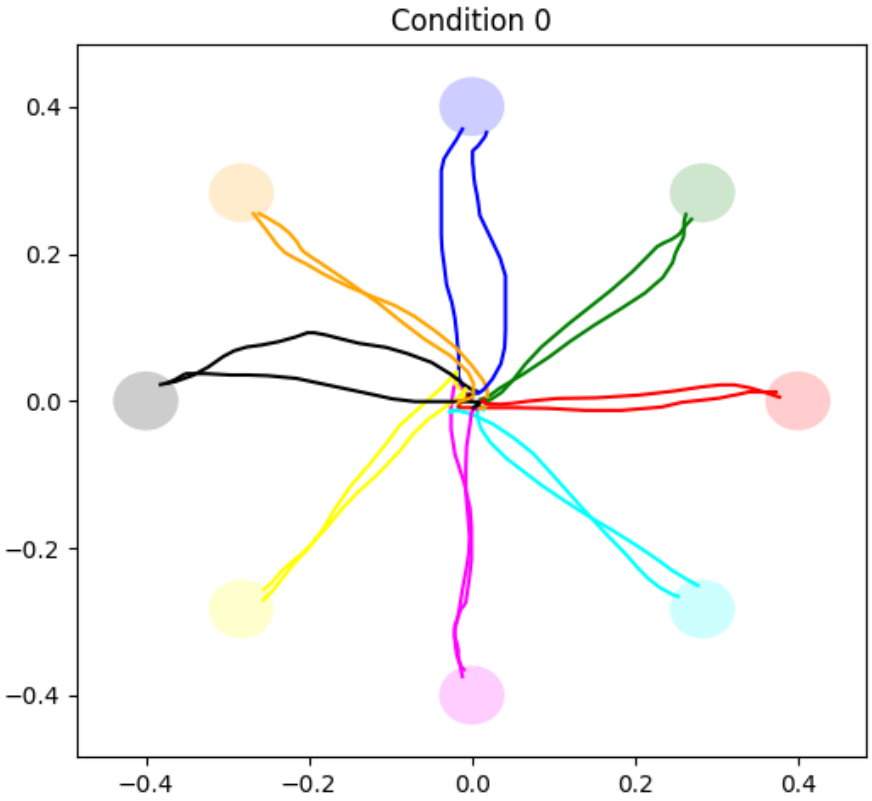
12.5 Plot of results for each trial

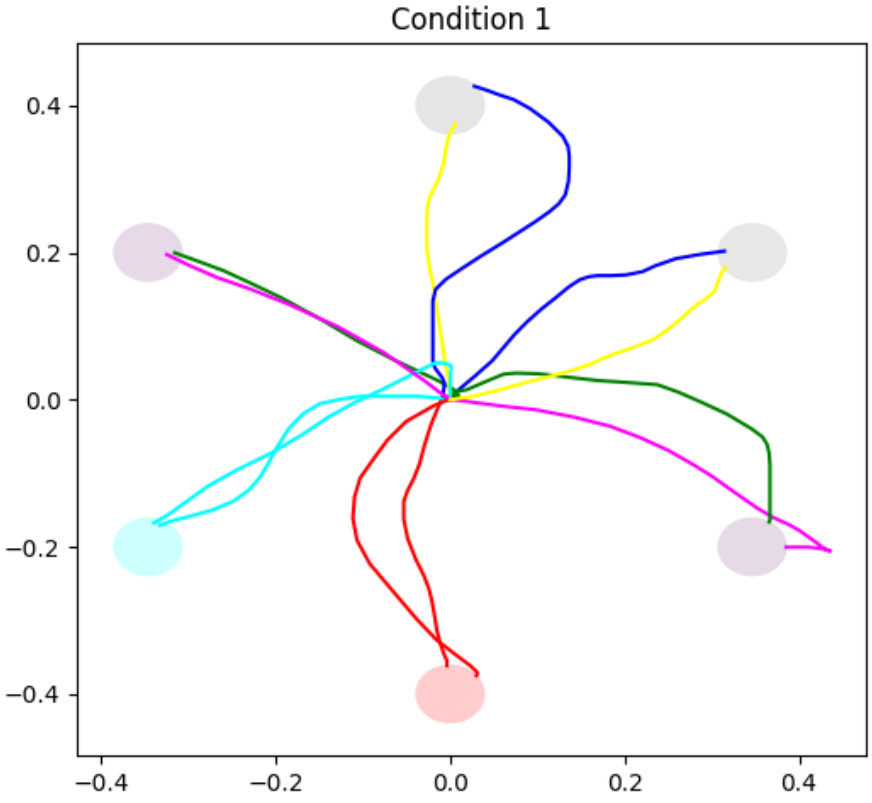
For example, a scatter plot of the mouse positions for each trial, labelled by the condition, trial number and repetition number:

12.6 Plot of all trials combined for each condition

Here we instead make one plot for each set of conditions in `trialList`, and super-impose all of the corresponding results:

[13]:
```
colors = ["blue", "green", "red", "cyan", "magenta", "yellow", "black", "orange"]
nConditions = len(psydata.trialList)
nTrials, nReps = psydata.sequenceIndices.shape
fig, axs = plt.subplots(nConditions, 1, figsize=(6, 6 * nConditions))
axs = np.reshape(axs, (nConditions))  # ensure axs is a 1d-array
for trial in range(nTrials):
    for rep in range(nReps):
        loc = (trial, rep)
        condition = psydata.sequenceIndices[loc]
        target_radius = psydata.trialList[condition]["target_size"]
        ax = axs[condition]
        ax.set_title(f"Condition {condition}")
        for positions, target_pos, color in zip(
            psydata.data["to_target_mouse_positions"][loc],
            psydata.data["target_pos"][loc],
            colors,
        ):
            ax.plot(positions[:, 0], positions[:, 1], color=color)
            ax.add_patch(
                plt.Circle(
                    target_pos,
                    target_radius,
                    edgecolor="none",
                    facecolor=color,
                    alpha=0.1,
                )
            )
plt.show()
```

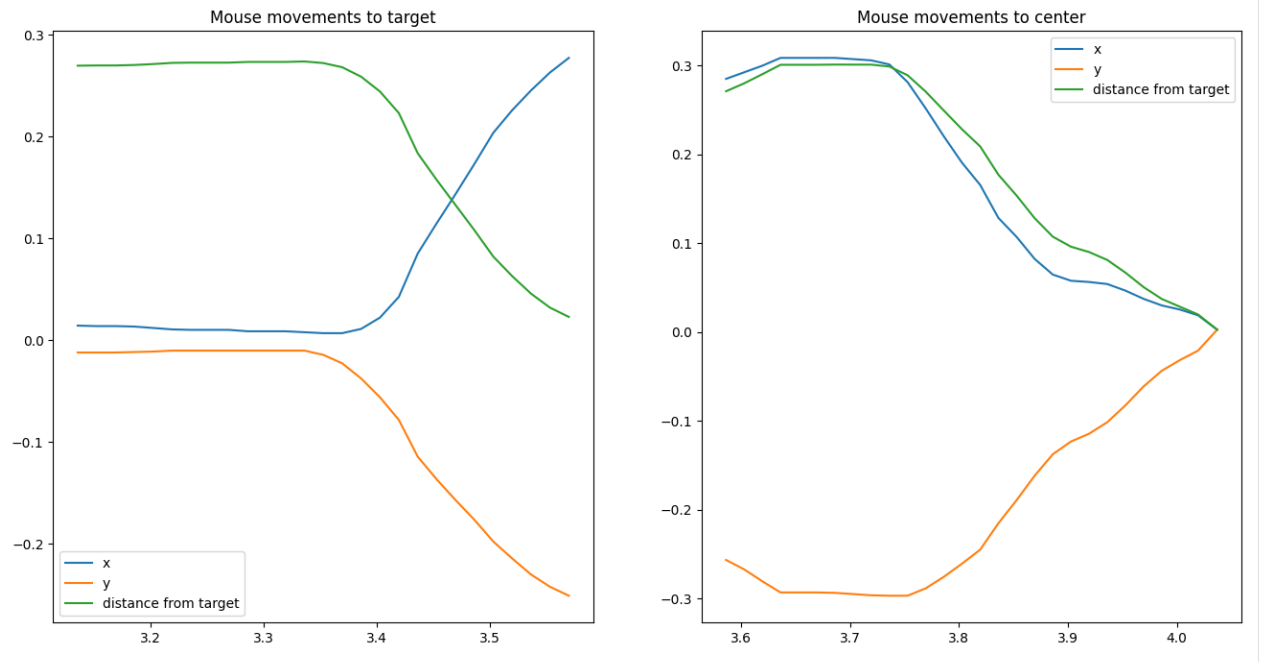

Condition 0
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4


Condition 1
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

12.7 Plot of mouse movements vs time

A plot of `x`, `y`, and `distance from target` versus time for a single move to/from a target

[14]:
```
def dist(xys, xy0):
    return np.sqrt(np.mean(np.power(xys - xy0, 2), axis=1))


fig, axs = plt.subplots(1, 2, figsize=(16, 8))

trial = 0
rep = 0
i_target = 3
loc = (trial, rep)
condition = psydata.sequenceIndices[loc]
target_radius = psydata.trialList[condition]["target_size"]
central_target_radius = psydata.trialList[condition]["central_target_size"]
for dest, ax in zip(["target", "center"], axs):
    positions = psydata.data[f"to_{dest}_mouse_positions"][loc][i_target]
    target = psydata.data["target_pos"][loc][i_target] if dest == "target" else [0, 0]
    times = psydata.data[f"to_{dest}_timestamps"][loc][i_target]
    ax.set_title(f"Mouse movements to {dest}")
    ax.plot(times, positions[:, 0], label="x")
    ax.plot(times, positions[:, 1], label="y")
    ax.plot(times, dist(positions, target), label="distance from target")
    ax.legend()
plt.show()
```



CHAPTER

THIRTEEN

CALCULATE THE AREA OF THE ENCLOSED GEOMETRIC OBJECT WITH PSYDAT FILE

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from psychopy.misc import fromFile
```

13.1 Import

A psydat file can be imported using the psychopy `fromFile` function: If you want to know the detailed content of the data in psydat file, please check the notebook 'raw_data.ipynb'

[2]:
```
psydata = fromFile("example.psydat")
```

13.2 Plot of results for each trial

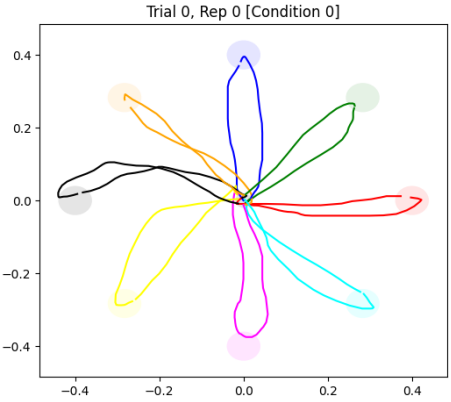
For example, a scatter plot of the mouse positions for each trial, labelled by the condition, trial number and repetition number:

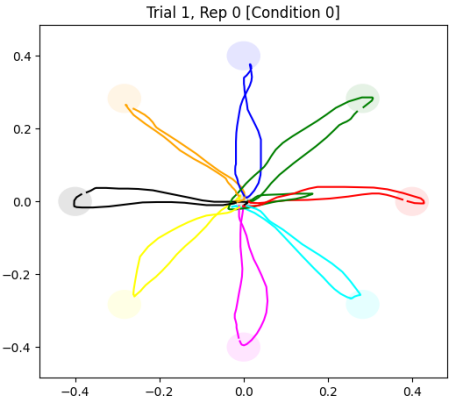
[3]:
```
colors = ["blue", "green", "red", "cyan", "magenta", "yellow", "black", "orange"]

nTrials, nReps = psydata.sequenceIndices.shape
fig, axs = plt.subplots(nTrials, nReps, figsize=(6, 6 * nTrials * nReps))
axs = np.reshape(
    axs, (nTrials, nReps)
)  # ensure axs is a 2d-array even if nTrials or nReps is 1
for trial in range(nTrials):
    for rep in range(nReps):
        loc = (trial, rep)
        condition = psydata.sequenceIndices[loc]
        target_radius = psydata.trialList[condition]["target_size"]
        central_target_radius = psydata.trialList[condition]["central_target_size"]
        ax = axs[loc]
        ax.set_title(f"Trial {trial}, Rep {rep} [Condition {condition}]")
        for positions, target_pos, color in zip(
            psydata.data["to_target_mouse_positions"][loc],
            psydata.data["target_pos"][loc],
```

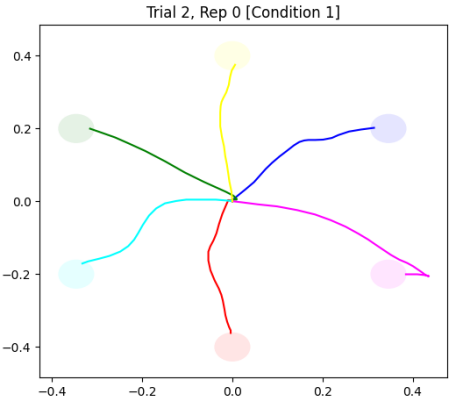
(continues on next page)

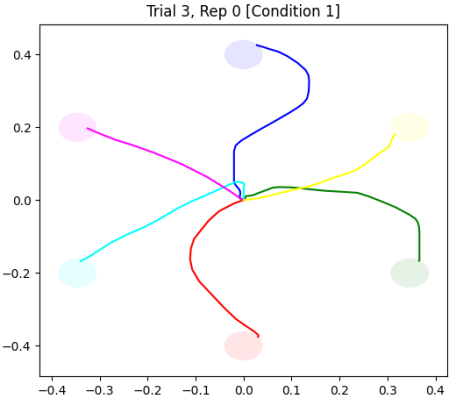
(continued from previous page)

```
            colors,
        ):
            ax.plot(positions[:, 0], positions[:, 1], color=color)
            ax.add_patch(
                plt.Circle(
                    target_pos,
                    target_radius,
                    edgecolor="none",
                    facecolor=color,
                    alpha=0.1,
                )
            )
        if not psydata.trialList[condition]["automove_cursor_to_center"]:
            for positions, color in zip(
                psydata.data["to_center_mouse_positions"][loc],
                colors,
            ):
                ax.plot(positions[:, 0], positions[:, 1], color=color)
            ax.add_patch(
                plt.Circle(
                    [0, 0],
                    central_target_radius,
                    edgecolor="none",
                    facecolor="black",
                    alpha=0.1,
                )
            )

plt.show()
```


Trial 0, Rep 0 [Condition 0]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4


Trial 1, Rep 0 [Condition 0]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4


Trial 2, Rep 0 [Condition 1]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4


Trial 3, Rep 0 [Condition 1]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4

13.3 Plot of area calculation results for each trial

How to calculate the area of irregular geometric object with given coordinates from psydat file?

The build-in operation `area` in the library `shapely` can calculate the area of geometry object. However, only for the valid one->not self intersected.

To tackle the self-intersection problem, the strategy is to split one self intersected object into the union of `LineString`(a geometry type composed of one or more line segments), then construct a bunch of valid polygons from these lines, then calculate the area of each valid polygon, sum them up.

[4]:
```
from typing import List
from typing import Tuple

from shapely.geometry import LineString
from shapely.ops import polygonize
from shapely.ops import unary_union
```

[5]:
```
def get_area_and_polygons(
    to_target_mouse_positions: np.ndarray, to_center_mouse_positions: np.ndarray
) -> Tuple[float, List[np.ndarray]]:
    """
    Calculates the total area enclosed by the mouse positions and the corresponding list␣
↪of closed polygons

    Uses the built-in operation `area` in the library `shapely` to calculate the area of␣
↪geometry object.
    However, this is only available for valid (not self intersected) geometries.
    To tackle the self-intersection problem,
    the strategy is to split one self intersected object into the union of LineString (a␣
↪geometry type composed of one or more line segments),
    then construct a bunch of valid polygons from these lines,
    then calculate the area of each valid polygon and sum them up.

    :param to_target_mouse_positions: x,y mouse positions moving towards the target
    :param to_center_mouse_positions: x,y mouse positions moving towards the center
    :return: area, list of x and y arrays of corresponding closed polygons

    """
    coords = np.concatenate(
        [
            to_target_mouse_positions,
            to_center_mouse_positions,
            to_target_mouse_positions[0:1],
        ]
    )
    polygons = polygonize(unary_union(LineString(coords)))
    area = sum(polygon.area for polygon in polygons)
    xy_arrays = [np.array(xy) for polygon in polygons for xy in polygon.exterior.xy]
    return area, xy_arrays
```

[6]:
```
def plot_and_calculate_area(data):
    colors = ["blue", "green", "red", "cyan", "magenta", "yellow", "black", "orange"]
```

(continues on next page)

(continued from previous page)

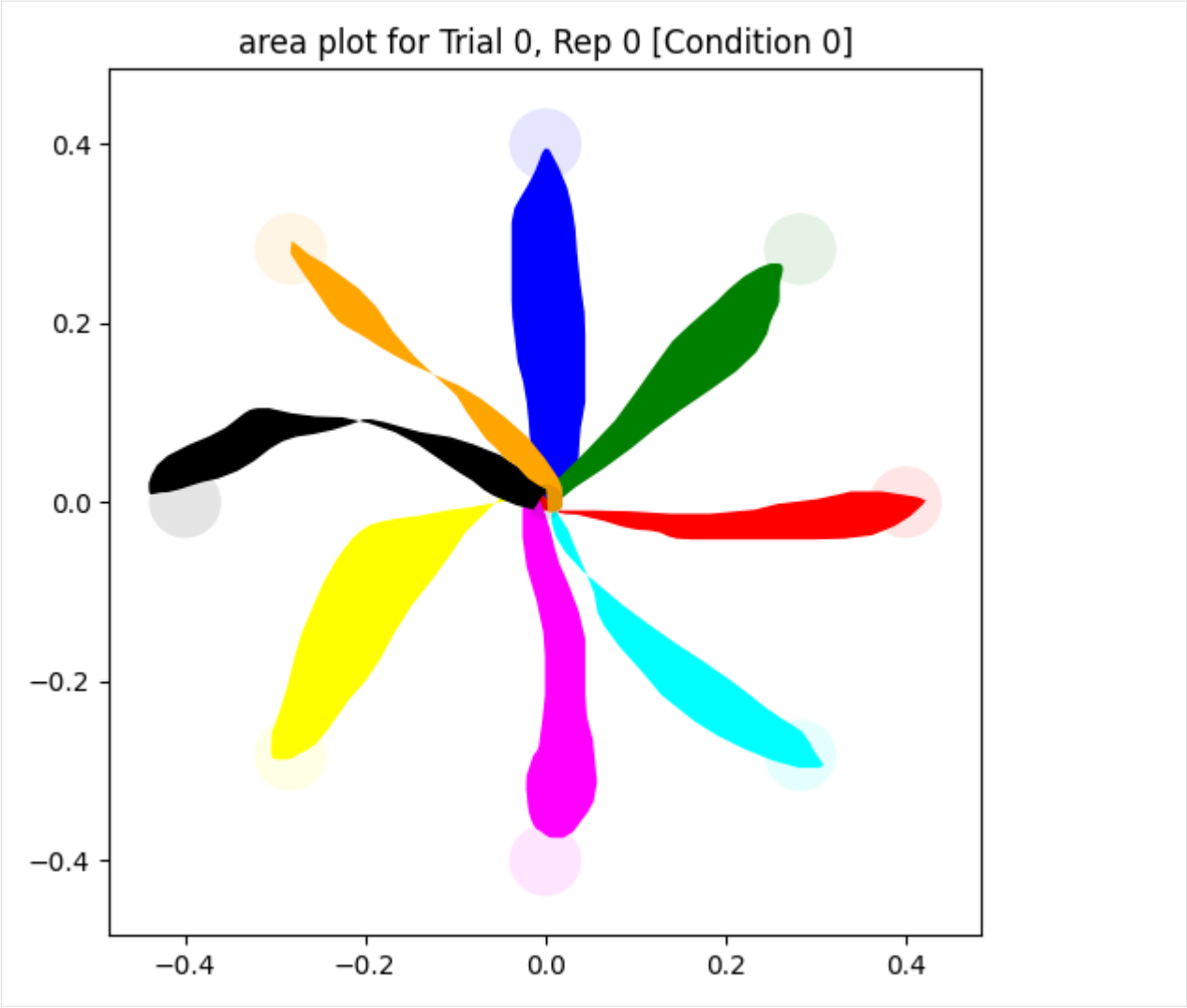
```
    nTrials, nReps = data.sequenceIndices.shape
    for trial in range(nTrials):
        for rep in range(nReps):
            loc = (trial, rep)
            condition = data.sequenceIndices[loc]
            target_radius = data.trialList[condition]["target_size"]
            central_target_radius = data.trialList[condition]["central_target_size"]

            # if condition "automove_cursor_to_center" is deselected, plot the line to␣
↪center, fill the enclosed area and output the area
            if not data.trialList[condition]["automove_cursor_to_center"]:
                fig, ax = plt.subplots(1, 1, figsize=(6, 6))
                ax.set_xbound(-0.5, 0.5)
                ax.set_ybound(-0.5, 0.5)
                ax.set_title(
                    f"area plot for Trial {trial}, Rep {rep} [Condition {condition}]"
                )
                print("-----------------------------------------------")
                print(
                    "area of Trial %d, Rep %d [Condition %s]" % (trial, rep, condition)
                )
                for (
                    to_target_mouse_positions,
                    to_center_mouse_positions,
                    target_pos,
                    color,
                ) in zip(
                    data.data["to_target_mouse_positions"][loc],
                    data.data["to_center_mouse_positions"][loc],
                    data.data["target_pos"][loc],
                    colors,
                ):
                    area, polygons = get_area_and_polygons(
                        to_target_mouse_positions, to_center_mouse_positions
                    )
                    ax.fill(*polygons, facecolor=color)

                    ax.add_patch(
                        plt.Circle(
                            target_pos,
                            target_radius,
                            edgecolor="none",
                            facecolor=color,
                            alpha=0.1,
                        )
                    )
                    print(f"{color}, area: {area:f}")

                ax.add_patch(
                    plt.Circle(
                        [0, 0],
                        central_target_radius,
```

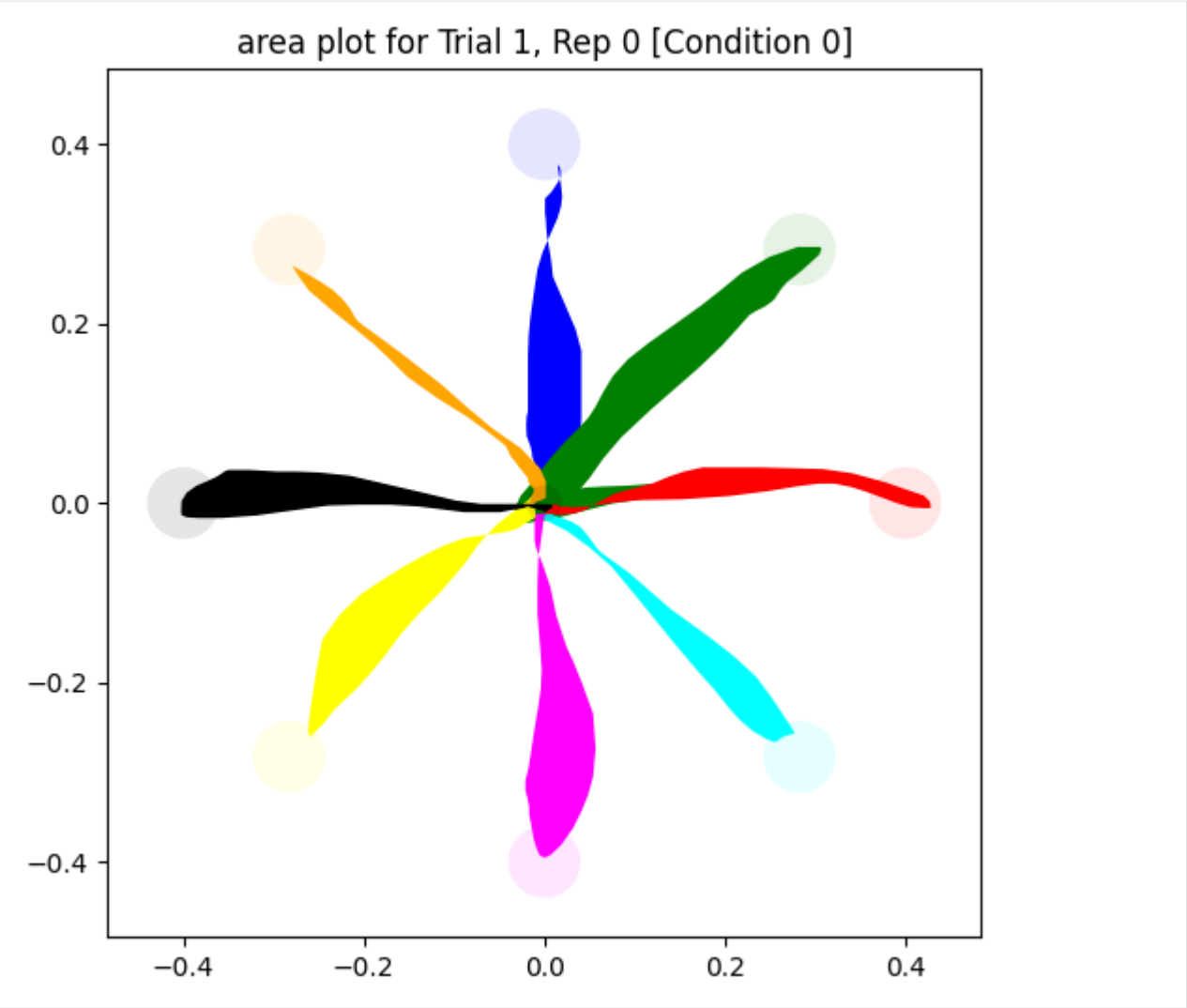
(continues on next page)

(continued from previous page)

```
                        edgecolor="none",
                        facecolor="black",
                        alpha=0.1,
                    )
                )

    plt.show()


plot_and_calculate_area(psydata)
```

```
--------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
blue, area: 0.023840
green, area: 0.016878
red, area: 0.012411
cyan, area: 0.015122
magenta, area: 0.018386
yellow, area: 0.026584
black, area: 0.014029
orange, area: 0.009765
--------------------------------------------
area of Trial 1, Rep 0 [Condition 0]
blue, area: 0.012711
green, area: 0.022543
red, area: 0.008020
cyan, area: 0.008811
magenta, area: 0.014737
yellow, area: 0.014034
black, area: 0.011179
orange, area: 0.004766
```


area plot for Trial 0, Rep 0 [Condition 0]
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4



13.4 Special cases

This strategy can also be applied to the following special cases:

13.4.1 no movement

There is no lines in the plot

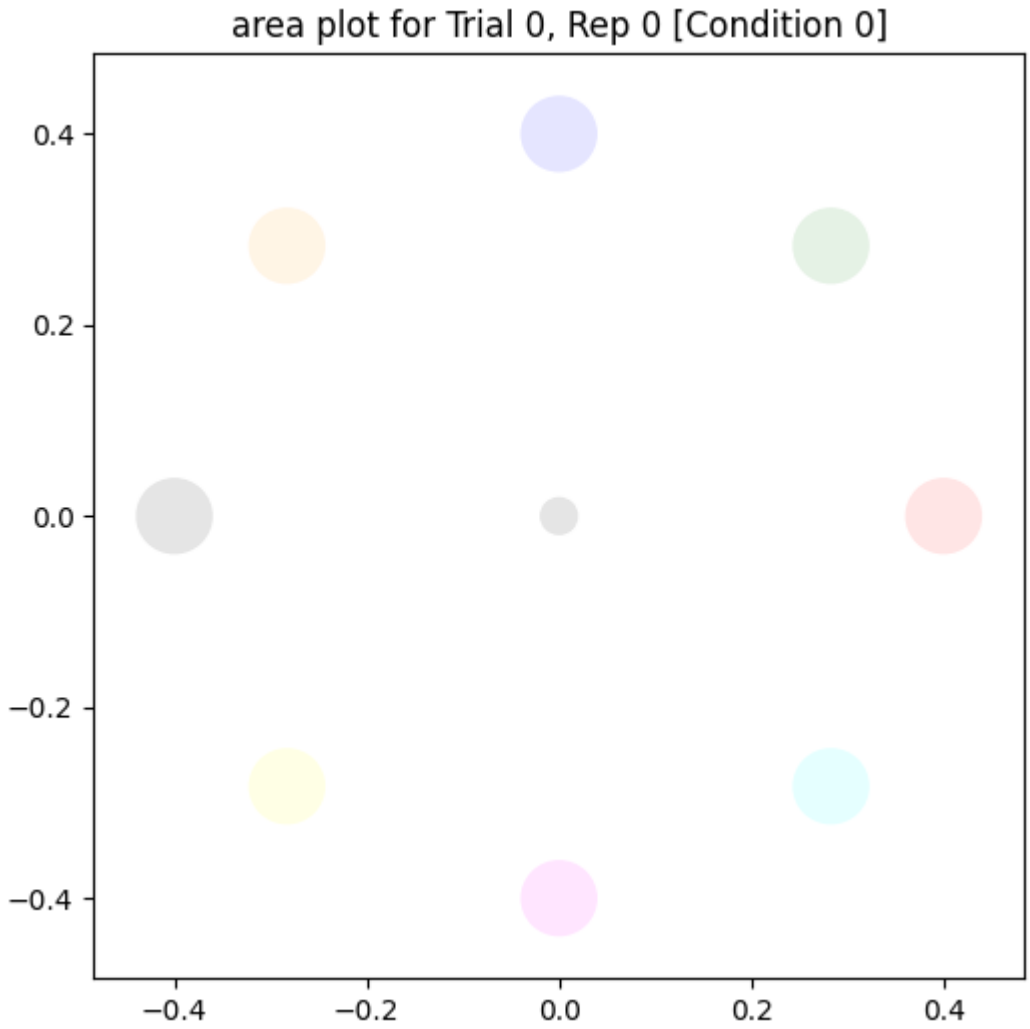
[7]:
```
psydata_no_movement = fromFile("example_no_movement.psydat")
plot_and_calculate_area(psydata_no_movement)
```

```
-----------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
blue, area: 0.000000
green, area: 0.000000
red, area: 0.000000
```

(continues on next page)

(continued from previous page)

```
cyan, area: 0.000000
magenta, area: 0.000000
yellow, area: 0.000000
black, area: 0.000000
orange, area: 0.000000
```



13.4.2 too much movement

The whole plot is full of lines

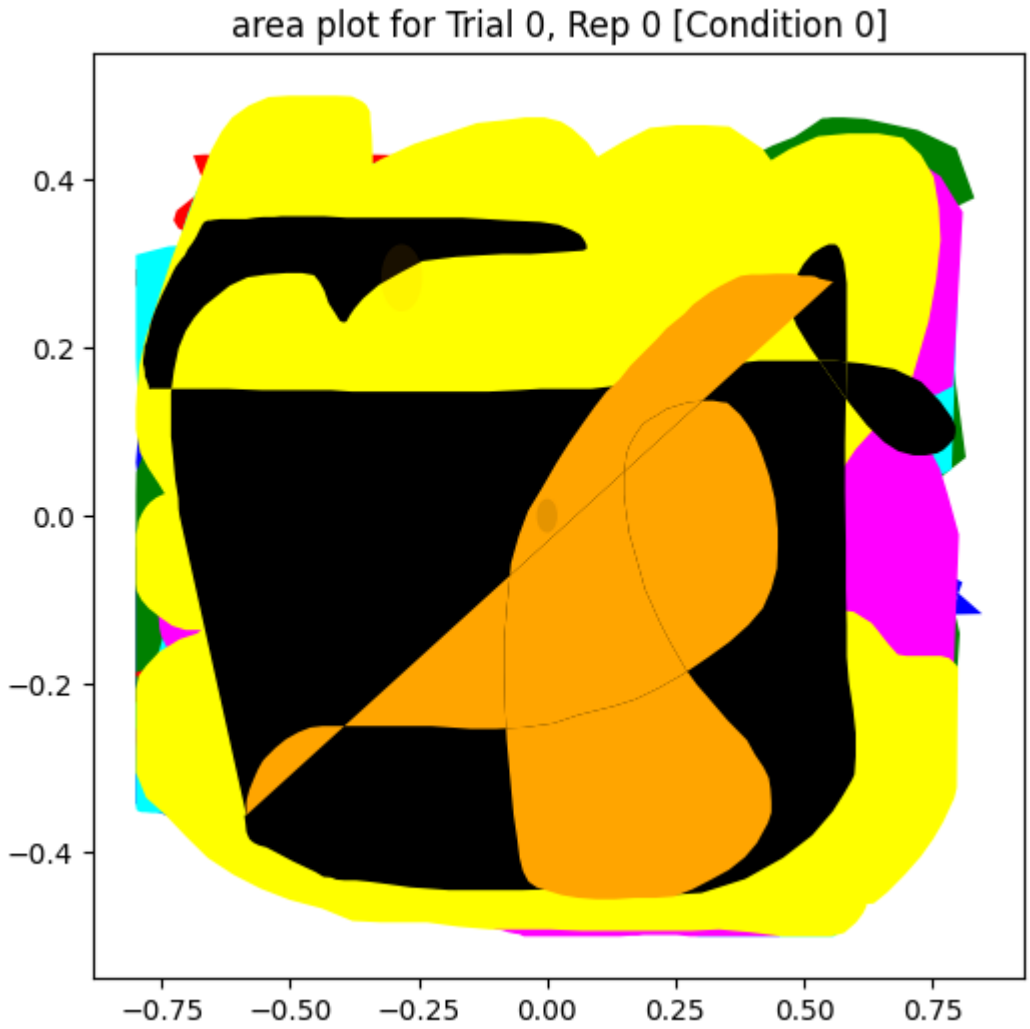
[8]:
```
psydata_over_movement = fromFile("example_over_movement.psydat")
plot_and_calculate_area(psydata_over_movement)
```

```
--------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
blue, area: 0.920733
green, area: 1.062104
red, area: 0.262005
```

(continues on next page)

(continued from previous page)

```
cyan, area: 1.181816
magenta, area: 1.167541
yellow, area: 1.221113
black, area: 0.818421
orange, area: 0.325764
```



13.4.3 some targets got reached, some not

For some targets, no line is approaching them, the corresponding area is 0

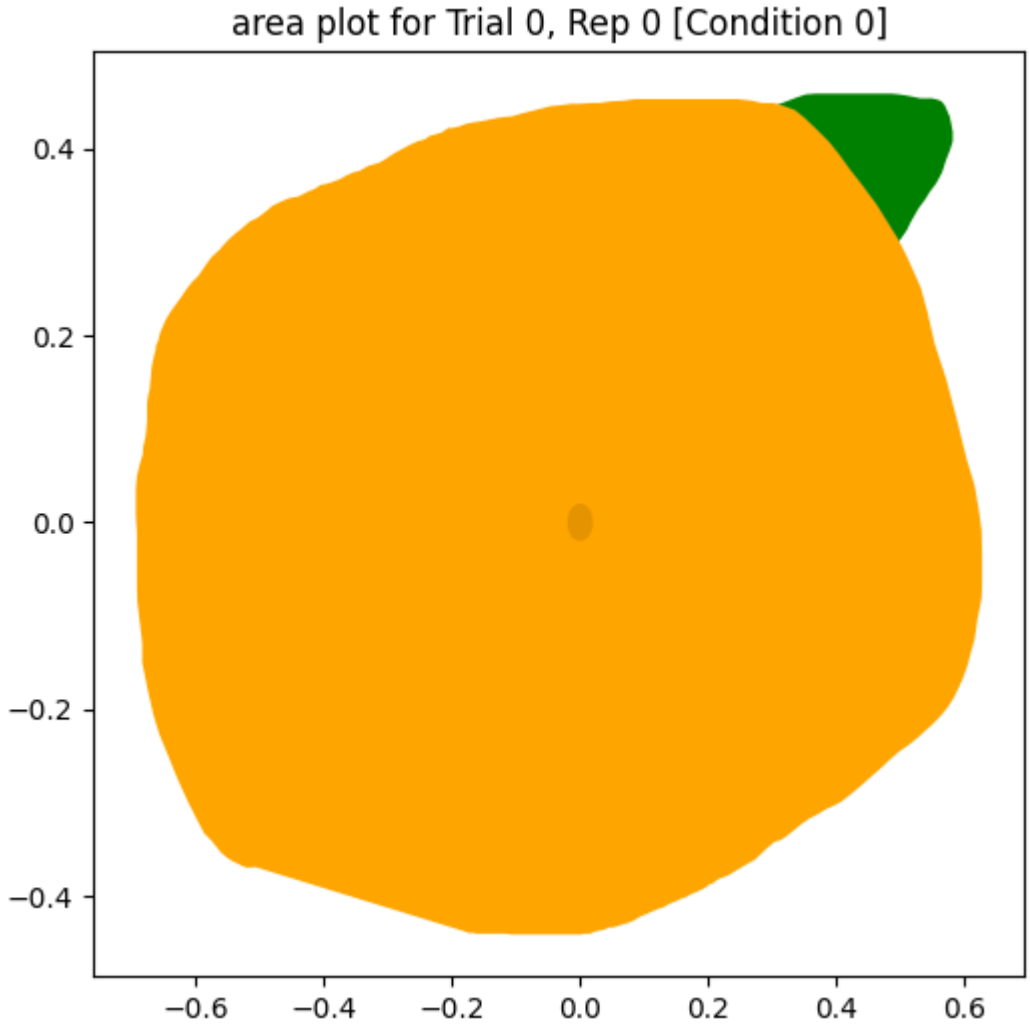
[9]:
```
psydata_target_not_reached = fromFile("example_target_not_reached.psydat")
plot_and_calculate_area(psydata_target_not_reached)
```

```
-------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
blue, area: 0.000000
green, area: 0.136767
red, area: 0.028414
```

(continues on next page)

(continued from previous page)

```
cyan, area: 0.000000
magenta, area: 0.055633
yellow, area: 0.085676
black, area: 0.000000
orange, area: 0.953468
```



13.4.4 select “Automatically move cursor to center” condition

The trials with condition “Automatically move cursor to center” selected will not be drawn, only the trials with condition “Automatically move cursor to center” deselected will be shown in the plot.

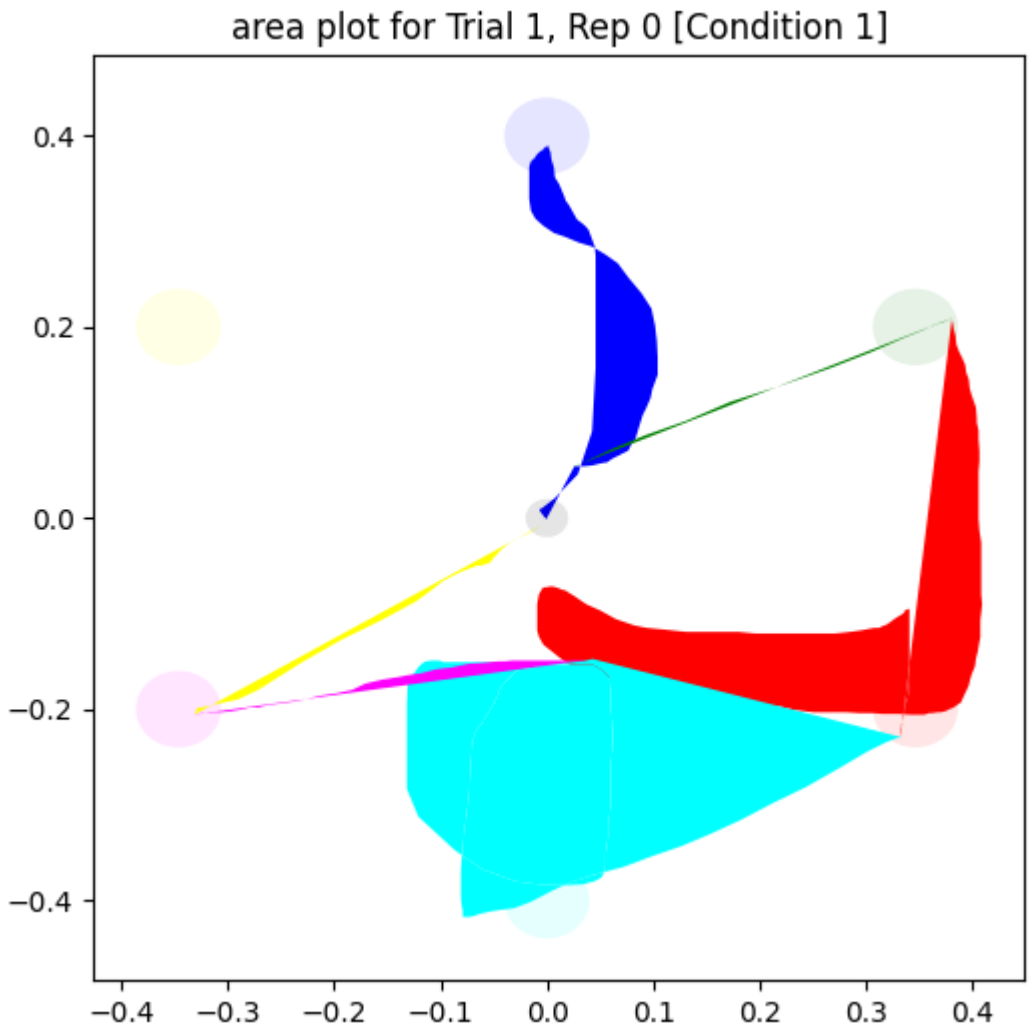
[10]:
```
psydata_select_back_to_center = fromFile("example_select_back_to_center.psydat")
plot_and_calculate_area(psydata_select_back_to_center)
```

```
--------------------------------------------
area of Trial 1, Rep 0 [Condition 1]
blue, area: 0.013264
green, area: 0.000636
```

(continues on next page)

(continued from previous page)

```
red, area: 0.044108
cyan, area: 0.074389
magenta, area: 0.002066
yellow, area: 0.001862
```

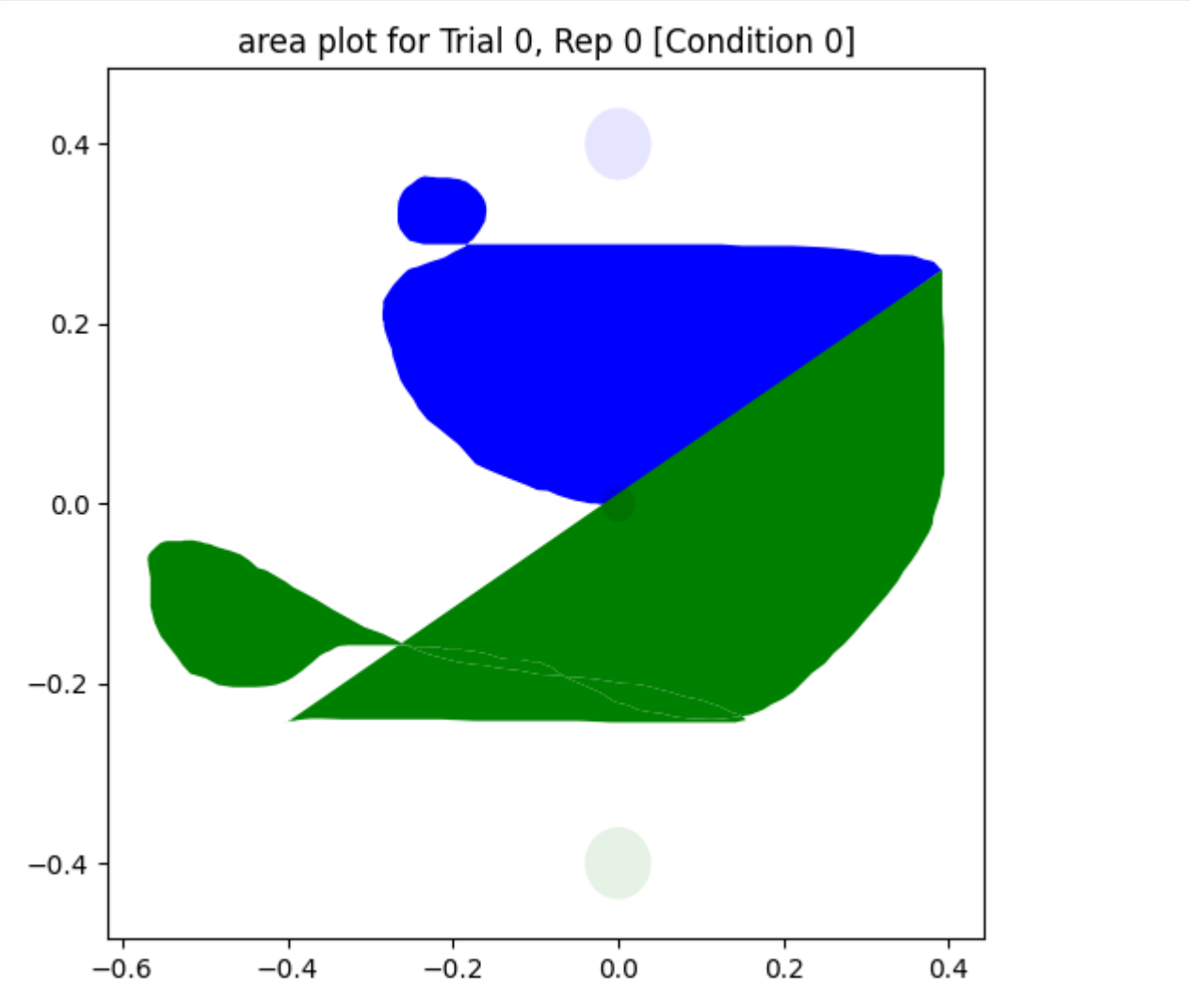


13.4.5 not closed line

For the path from the center to target and back to center is not closed, will be closed by the first and last coordinates automatically.

[11]:
```
psydata_not_closed_line = fromFile("example_not_closed_line.psydat")
plot_and_calculate_area(psydata_not_closed_line)
```

```
--------------------------------------------
area of Trial 0, Rep 0 [Condition 0]
blue, area: 0.132591
green, area: 0.203647
```


area plot for Trial 0, Rep 0 [Condition 0]
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4

CHAPTER

FOURTEEN

DEVELOPER INSTALLATION

14.1 Developer install

To clone the repo, make an editable installation and run the tests:

```
git clone https://github.com/ssciwr/vstt.git
cd vstt
pip install -e .[tests,docs]
xvfb-run pytest
```

To build the docs (in `docs/_build/html/index.html`)

```
cd docs
make
```

14.2 Pre-requisites

The only dependency is psychopy. Unfortunately, psychopy itself has a lot of dependencies, some of which are system libraries.

- **on Ubuntu 22.04 with Python 3.9**
  - `sudo apt-get install swig libasound2-dev portaudio19-dev libpulse-dev libusb-1.0-0-dev libsndfile1-dev libportmidi-dev liblo-dev libgtk-3-dev`
  - `pip install wxPython` (which took a long time to complete - alternative is to install a pre-built wheel)
  - **additionally, at runtime psychtoolbox needs permission on linux to set its priority:**
    - `sudo setcap cap_sys_nice+ep `python -c "import os; import sys; print(os.path.realpath(sys.executable))"``
  - **alternatively simply remove psychtoolbox (it is an optional psychopy dependency):**
    - `pip uninstall psychtoolbox`

CHAPTER

FIFTEEN

FORWARDS COMPATIBILITY

15.1 Adding new features

It seems likely that new features will need to be added in the future, in particular:

- adding a new field in `MotorTaskTrial`
- adding some corresponding task logic that uses this field

Each new feature should have a default value resulting in unchanged behaviour from before the feature was added. This might require a boolean on/off flag in addition to another field with the actual parameter. This is required to ensure forwards compatibility of existing experiments with new versions of the software.

15.2 Unknown or missing fields

When importing a trial:

- Unknown fields are ignored
- Missing fields are replaced with their default values

This is required to ensure forwards compatibility of experiments with new versions of the software, as well as (limited) backwards compatibility of experiments with older versions of the software.

15.3 Forwards compatibility

As a consequence of the two sections above, once an experiment is created using one version of the software, it is guaranteed to continue to behave in the same way with any later version of the software.

15.4 Backwards compatibility

If an experiment is created in a newer version of the software and then opened in an older version of the software, the experiment will still run. However, if it contains non-default values for any fields that did not exist in the older version of the software, the user-visible task will differ.

CHAPTER

SIXTEEN

API REFERENCE

16.1 vstt.common

Functions

import_typed_dict(input_dict, default_typed_dict)

16.1.1 vstt.common.import_typed_dict

vstt.common.**import_typed_dict**(*input_dict: Mapping[str, Any]*, *default_typed_dict: VsttTypedDict*) → VsttTypedDict

16.2 vstt.display

Functions

default_display_options()
display_options_labels()
import_display_options(display_options_dict)

16.2.1 vstt.display.default_display_options

vstt.display.**default_display_options**() → *DisplayOptions*

16.2.2 vstt.display.display_options_labels

vstt.display.**display_options_labels**() → dict[str, str]

16.2.3 vstt.display.import_display_options

vstt.display.**import_display_options**(*display_options_dict: dict*) → *DisplayOptions*

16.3 vstt.experiment

Classes

Experiment([filename])

16.3.1 vstt.experiment.Experiment

class vstt.experiment.**Experiment**(*filename: str | None = None*)

Methods

__init__([filename])	
clear_results()	
create_trialhandler()	
import_and_validate_dicts(filename, ...)	
import_and_validate_trial_handler(trial_hand	
load_excel(filename)	
load_file(filename)	
load_json(filename)	
load_psydat(filename)	
save_excel(filename, data_format)	data_format can be - "trial": one sheet of data exported per trial - "target: one sheet of data exported per target
save_json(filename)	
save_psydat(filename)	

vstt.experiment.Experiment.__init__

Experiment.**__init__**(*filename: str | None = None*)

vstt.experiment.Experiment.clear_results

Experiment.**clear_results**() → None

vstt.experiment.Experiment.create_trialhandler

Experiment.**create_trialhandler**() → TrialHandlerExt

vstt.experiment.Experiment.import_and_validate_dicts

Experiment.**import_and_validate_dicts**(*filename: str*, *metadata_dict: dict*, *display_options_dict: dict*, *trial_dict_list: list[dict]*) → None

vstt.experiment.Experiment.import_and_validate_trial_handler

Experiment.**import_and_validate_trial_handler**(*trial_handler: TrialHandlerExt*) → None

vstt.experiment.Experiment.load_excel

Experiment.**load_excel**(*filename: str*) → None

vstt.experiment.Experiment.load_file

Experiment.**load_file**(*filename: str*) → None

vstt.experiment.Experiment.load_json

Experiment.**load_json**(*filename: str*) → None

vstt.experiment.Experiment.load_psydat

Experiment.**load_psydat**(*filename: str*) → None

vstt.experiment.Experiment.save_excel

Experiment.**save_excel**(*filename: str*, *data_format: str*) → None

data_format can be - “trial”: one sheet of data exported per trial - “target: one sheet of data exported per target

vstt.experiment.Experiment.save_json

Experiment.**save_json**(*filename: str*) → None

vstt.experiment.Experiment.save_psydat

Experiment.**save_psydat**(*filename: str*) → None

16.4 vstt.geom

Functions

equidistant_angles(n_points)	An array of *n_points* equidistantly spaced angles in radians
points_on_circle(n_points, radius[, ...])	
rotate_point(point, angle_radians[, pivot_point])	Rotate *point* by an angle *angle_radians* around the point *pivot_point*
to_target_dists(pos, target_xys, ...)	

16.4.1 vstt.geom.equidistant_angles

vstt.geom.**equidistant_angles**(*n_points: int*) → ndarray

An array of *n_points* equidistantly spaced angles in radians

16.4.2 vstt.geom.points_on_circle

vstt.geom.**points_on_circle**(*n_points: int*, *radius: float*, *include_centre: bool = False*) → ndarray

16.4.3 vstt.geom.rotate_point

vstt.geom.**rotate_point**(*point: tuple[float, float]*, *angle_radians: float*, *pivot_point: tuple[float, float] = (0, 0)*) → tuple[float, float]

Rotate *point* by an angle *angle_radians* around the point *pivot_point*

16.4.4 vstt.geom.to_target_dists

vstt.geom.**to_target_dists**(*pos: ndarray*, *target_xys: ndarray*, *target_index: int*, *has_central_target: bool*) → tuple[float, float]

Classes

JoystickPointUpdater(angle_degrees, max_speed)	Update a point (x,y) according to a velocity vector (vx, vy), with optional rotation of the vector
PointRotator(angle_degrees)	Rotate a point (x,y) by a fixed angle in degrees around the origin

16.4.5 vstt.geom.JoystickPointUpdater

class vstt.geom.**JoystickPointUpdater**(*angle_degrees: float*, *max_speed: float*, *window_size: np.ndarray | None = None*)

Update a point (x,y) according to a velocity vector (vx, vy), with optional rotation of the vector

vx, vy should lie in the range [-1, 1]

The returned point is given by

x -> x + *max_speed* * vx' y -> y + *max_speed* * vy'

Where (vx',vy') is (vx, vy) rotated by *angle_degrees*

The returned point is clipped to lie within the screen, where the screen has height 1 and (0,0) lies in the centre of the screen So the y-range is always [-0.5, 0.5], and the x-range is this multiplied by (width/height)

Methods

__init__(angle_degrees, max_speed[, window_size])	

vstt.geom.JoystickPointUpdater.__init__

JoystickPointUpdater.**__init__**(*angle_degrees: float*, *max_speed: float*, *window_size: np.ndarray | None = None*)

16.4.6 vstt.geom.PointRotator

class vstt.geom.**PointRotator**(*angle_degrees: float*)

Rotate a point (x,y) by a fixed angle in degrees around the origin

Methods

__init__(angle_degrees)

vstt.geom.PointRotator.__init__

PointRotator.**__init__**(*angle_degrees: float*)

16.5 vstt.meta

Functions

default_metadata()
import_metadata(metadata_dict)
metadata_labels()

16.5.1 vstt.meta.default_metadata

vstt.meta.**default_metadata**() → *Metadata*

16.5.2 vstt.meta.import_metadata

vstt.meta.**import_metadata**(*metadata_dict: dict*) → *Metadata*

16.5.3 vstt.meta.metadata_labels

vstt.meta.**metadata_labels**() → dict

16.6 vstt.stats

Functions

append_stats_data_to_excel(df, writer, ...)	data_format can be - "trial": one sheet of data exported per trial - "target: one sheet of data exported per target
concatenate_mouse_positions(x)	concatenate the "to_target_mouse_positions" and "to_center_mouse_positions"
get_acceleration(mouse_times, mouse_positions)	get acceleration
get_closed_polygon(...)	connect the to target path and to center path to a closed polygon
get_derivative(y, x)	get derivative dy/dx
get_first_movement_index(mouse_times, ...)	get index of the first movement in mouse_times
get_movement_length(...)	calculate the length of the paths connecting the target and the center if only 1 path exists, another path is the distance between the head and tail of the existing path.
get_velocity(mouse_times, mouse_positions)	get velocity
list_dest_stat_label_units()	
preprocess_mouse_positions(mouse_positions)	reshape the mouse position to (0, 2) if the mouse position is empty, to prevent error happens in the following concatenate() function
stats_dataframe(trial_handler)	

16.6.1 vstt.stats.append_stats_data_to_excel

vstt.stats.**append_stats_data_to_excel**(*df: DataFrame*, *writer: Any*, *data_format: str*) → None

data_format can be - "trial": one sheet of data exported per trial - "target: one sheet of data exported per target

16.6.2 vstt.stats.concatenate_mouse_positions

vstt.stats.**concatenate_mouse_positions**(*x: ndarray*) → ndarray

concatenate the "to_target_mouse_positions" and "to_center_mouse_positions"

Parameters

x – the data to concatenate

Returns

the concatenated result

16.6.3 vstt.stats.get_acceleration

vstt.stats.**get_acceleration**(*mouse_times: ndarray*, *mouse_positions: ndarray*) → ndarray

get acceleration

Parameters

- **mouse_times** – The array of timestamps
- **mouse_positions** – The array of mouse positions

Returns

the array of acceleration

16.6.4 vstt.stats.get_closed_polygon

vstt.stats.**get_closed_polygon**(*to_target_mouse_positions: ndarray*, *to_center_mouse_positions: ndarray*) → ndarray

connect the to target path and to center path to a closed polygon

Parameters

- **to_target_mouse_positions** – x,y mouse positions moving towards the target
- **to_center_mouse_positions** – x,y mouse positions moving towards the center

Returns

x,y mouse positions of the closed polygon

16.6.5 vstt.stats.get_derivative

vstt.stats.**get_derivative**(*y: ndarray*, *x: ndarray*) → ndarray

get derivative dy/dx

Parameters

- **y** – the array of y
- **x** – the array of x

Returns

the array of dy/dx

16.6.6 vstt.stats.get_first_movement_index

vstt.stats.**get_first_movement_index**(*mouse_times: np.ndarray*, *mouse_positions: np.ndarray*, *to_target_num_timestamps_before_visible: int*) → int | None

get index of the first movement in mouse_times

Parameters

- **mouse_times** – The array of timestamps
- **mouse_positions** – The array of mouse positions
- **to_target_num_timestamps_before_visible** – The index of the first timestamp where the target is visible

Returns

the first movement index in mouse_times

16.6.7 vstt.stats.get_movement_length

vstt.stats.**get_movement_length**(*to_target_mouse_positions: ndarray*, *to_center_mouse_positions: ndarray*) → float

calculate the length of the paths connecting the target and the center if only 1 path exists, another path is the distance between the head and tail of the existing path.

Parameters

- **to_target_mouse_positions** – x,y mouse positions moving towards the target

- **to_center_mouse_positions** – x,y mouse positions moving towards the center

Returns
length of the movement

16.6.8 vstt.stats.get_velocity

vstt.stats.**get_velocity**(*mouse_times: ndarray*, *mouse_positions: ndarray*) → ndarray

get velocity

Parameters

- **mouse_times** – The array of timestamps
- **mouse_positions** – The array of mouse positions

Returns
the array of velocity

16.6.9 vstt.stats.list_dest_stat_label_units

vstt.stats.**list_dest_stat_label_units**() → list[tuple[str, list[tuple[str, str, str]]]]

16.6.10 vstt.stats.preprocess_mouse_positions

vstt.stats.**preprocess_mouse_positions**(*mouse_positions: ndarray*) → ndarray

reshape the mouse position to (0, 2) if the mouse position is empty, to prevent error happens in the following concatenate() function

Parameters
mouse_positions – x,y mouse positions during movement

Returns
x,y mouse positions after preprocess

16.6.11 vstt.stats.stats_dataframe

vstt.stats.**stats_dataframe**(*trial_handler: TrialHandlerExt*) → DataFrame

16.7 vstt.task

Functions

add_trial_data_to_trial_handler(trial_data, ...)

16.7.1 vstt.task.add_trial_data_to_trial_handler

vstt.task.**add_trial_data_to_trial_handler**(*trial_data:* TrialData, *trial_handler: TrialHandlerExt*) → None

Classes

MotorTask(experiment[, win])	
TrialData(trial, rng)	Stores the data from the trial that will be stored in the psychopy trial_handler
TrialManager(win, trial)	Stores the drawable elements and other objects needed during a trial

16.7.2 vstt.task.MotorTask

class vstt.task.**MotorTask**(*experiment:* Experiment, *win: Window | None = None*)

Methods

__init__(experiment[, win])	
run()	

vstt.task.MotorTask.__init__

MotorTask.**__init__**(*experiment:* Experiment, *win: Window | None = None*)

vstt.task.MotorTask.run

MotorTask.**run**() → bool

16.7.3 vstt.task.TrialData

class vstt.task.**TrialData**(*trial: dict[str, Any]*, *rng: Generator*)

Stores the data from the trial that will be stored in the psychopy trial_handler

Methods

__init__(trial, rng)

vstt.task.TrialData.__init__

TrialData.**__init__**(*trial: dict[str, Any]*, *rng: Generator*)

16.7.4 vstt.task.TrialManager

class vstt.task.**TrialManager**(*win: Window*, *trial:* Trial)

Stores the drawable elements and other objects needed during a trial

Methods

__init__(win, trial)
cursor_path_add_vertex(vertex[, clear_existing])

vstt.task.TrialManager.__init__

TrialManager.**__init__**(*win: Window*, *trial:* Trial)

vstt.task.TrialManager.cursor_path_add_vertex

TrialManager.**cursor_path_add_vertex**(*vertex: tuple[float, float]*, *clear_existing: bool = False*) → None

16.8 vstt.trial

Functions

default_trial()
describe_trial(trial)
describe_trials(trials)
get_trial_from_user([initial_trial])
import_and_validate_trial(trial_or_dict)
trial_labels()

16.8.1 vstt.trial.default_trial

vstt.trial.**default_trial**() → *Trial*

16.8.2 vstt.trial.describe_trial

vstt.trial.**describe_trial**(*trial:* Trial) → str

16.8.3 vstt.trial.describe_trials

vstt.trial.**describe_trials**(*trials: list[*Trial*]*) → str

16.8.4 vstt.trial.get_trial_from_user

vstt.trial.**get_trial_from_user**(*initial_trial:* Trial | *None* = *None*) → *Trial* | None

16.8.5 vstt.trial.import_and_validate_trial

vstt.trial.**import_and_validate_trial**(*trial_or_dict: Mapping[str, Any]*) → *Trial*

16.8.6 vstt.trial.trial_labels

vstt.trial.**trial_labels**() → dict

16.9 vstt.vtypes

Classes

DisplayOptions
Metadata
Trial

16.9.1 vstt.vtypes.DisplayOptions

class vstt.vtypes.**DisplayOptions**

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, default is returned if given, otherwise KeyError is raised
popitem()	Remove and return a (key, value) pair as a 2-tuple.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

vstt.vtypes.DisplayOptions.__init__

DisplayOptions.**__init__**(**args*, ***kwargs*)

vstt.vtypes.DisplayOptions.clear

DisplayOptions.**clear**() → None. Remove all items from D.

vstt.vtypes.DisplayOptions.copy

DisplayOptions.**copy**() → a shallow copy of D

vstt.vtypes.DisplayOptions.fromkeys

DisplayOptions.**fromkeys**(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

vstt.vtypes.DisplayOptions.get

DisplayOptions.**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.DisplayOptions.items

DisplayOptions.**items**() → a set-like object providing a view on D's items

vstt.vtypes.DisplayOptions.keys

DisplayOptions.**keys**() → a set-like object providing a view on D's keys

vstt.vtypes.DisplayOptions.pop

DisplayOptions.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, default is returned if given, otherwise KeyError is raised

vstt.vtypes.DisplayOptions.popitem

DisplayOptions.**popitem**()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

vstt.vtypes.DisplayOptions.setdefault

DisplayOptions.**setdefault**(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.DisplayOptions.update

DisplayOptions.**update**([*E*,]**F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

vstt.vtypes.DisplayOptions.values

DisplayOptions.**values**() → an object providing a view on D's values

Attributes

to_target_paths
to_center_paths
targets
central_target
to_target_reaction_time
to_center_reaction_time
to_target_movement_time
to_center_movement_time
to_target_time
to_center_time
to_target_distance
to_center_distance
to_target_rmse
to_center_rmse
to_target_success
to_center_success
averages
area
normalized_area
peak_velocity
peak_acceleration
to_target_spatial_error
to_center_spatial_error
movement_time_at_peak_velocity
total_time_at_peak_velocity
movement_distance_at_peak_velocity
rmse_movement_at_peak_velocity

vstt.vtypes.DisplayOptions.to_target_paths

DisplayOptions.**to_target_paths: bool**

vstt.vtypes.DisplayOptions.to_center_paths

DisplayOptions.**to_center_paths: bool**

vstt.vtypes.DisplayOptions.targets

DisplayOptions.**targets: bool**

vstt.vtypes.DisplayOptions.central_target

DisplayOptions.**central_target: bool**

vstt.vtypes.DisplayOptions.to_target_reaction_time

DisplayOptions.**to_target_reaction_time: bool**

vstt.vtypes.DisplayOptions.to_center_reaction_time

DisplayOptions.**to_center_reaction_time: bool**

vstt.vtypes.DisplayOptions.to_target_movement_time

DisplayOptions.**to_target_movement_time: bool**

vstt.vtypes.DisplayOptions.to_center_movement_time

DisplayOptions.**to_center_movement_time: bool**

vstt.vtypes.DisplayOptions.to_target_time

DisplayOptions.**to_target_time: bool**

vstt.vtypes.DisplayOptions.to_center_time

DisplayOptions.**to_center_time: bool**

vstt.vtypes.DisplayOptions.to_target_distance

DisplayOptions.**to_target_distance: bool**

vstt.vtypes.DisplayOptions.to_center_distance

DisplayOptions.**to_center_distance: bool**

vstt.vtypes.DisplayOptions.to_target_rmse

DisplayOptions.**to_target_rmse: bool**

vstt.vtypes.DisplayOptions.to_center_rmse

DisplayOptions.**to_center_rmse: bool**

vstt.vtypes.DisplayOptions.to_target_success

DisplayOptions.**to_target_success: bool**

vstt.vtypes.DisplayOptions.to_center_success

DisplayOptions.**to_center_success: bool**

vstt.vtypes.DisplayOptions.averages

DisplayOptions.**averages: bool**

vstt.vtypes.DisplayOptions.area

DisplayOptions.**area: bool**

vstt.vtypes.DisplayOptions.normalized_area

DisplayOptions.**normalized_area: bool**

vstt.vtypes.DisplayOptions.peak_velocity

DisplayOptions.**peak_velocity: bool**

vstt.vtypes.DisplayOptions.peak_acceleration

DisplayOptions.**peak_acceleration: bool**

vstt.vtypes.DisplayOptions.to_target_spatial_error

DisplayOptions.**to_target_spatial_error: bool**

vstt.vtypes.DisplayOptions.to_center_spatial_error

DisplayOptions.**to_center_spatial_error: bool**

vstt.vtypes.DisplayOptions.movement_time_at_peak_velocity

DisplayOptions.**movement_time_at_peak_velocity: bool**

vstt.vtypes.DisplayOptions.total_time_at_peak_velocity

DisplayOptions.**total_time_at_peak_velocity: bool**

vstt.vtypes.DisplayOptions.movement_distance_at_peak_velocity

DisplayOptions.**movement_distance_at_peak_velocity: bool**

vstt.vtypes.DisplayOptions.rmse_movement_at_peak_velocity

DisplayOptions.**rmse_movement_at_peak_velocity: bool**

16.9.2 vstt.vtypes.Metadata

class vstt.vtypes.**Metadata**

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, default is returned if given, otherwise KeyError is raised
popitem()	Remove and return a (key, value) pair as a 2-tuple.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

vstt.vtypes.Metadata.__init__

Metadata.**__init__**(**args*, ***kwargs*)

vstt.vtypes.Metadata.clear

Metadata.**clear**() → None. Remove all items from D.

vstt.vtypes.Metadata.copy

Metadata.**copy**() → a shallow copy of D

vstt.vtypes.Metadata.fromkeys

Metadata.**fromkeys**(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

vstt.vtypes.Metadata.get

Metadata.**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.Metadata.items

Metadata.**items**() → a set-like object providing a view on D's items

vstt.vtypes.Metadata.keys

Metadata.**keys**() → a set-like object providing a view on D's keys

vstt.vtypes.Metadata.pop

Metadata.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, default is returned if given, otherwise KeyError is raised

vstt.vtypes.Metadata.popitem

Metadata.**popitem**()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

vstt.vtypes.Metadata.setdefault

Metadata.**setdefault**(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.Metadata.update

Metadata.**update**([*E*,]**F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

vstt.vtypes.Metadata.values

Metadata.**values**() → an object providing a view on D's values

Attributes

name
subject
date
author
display_title
display_text1
display_text2
display_text3
display_text4
display_duration
show_delay_countdown
enter_to_skip_delay

vstt.vtypes.Metadata.name

Metadata.**name: str**

vstt.vtypes.Metadata.subject

Metadata.**subject: str**

vstt.vtypes.Metadata.date

Metadata.**date: str**

vstt.vtypes.Metadata.author

Metadata.**author: str**

vstt.vtypes.Metadata.display_title

Metadata.**display_title: str**

vstt.vtypes.Metadata.display_text1

Metadata.**display_text1: str**

vstt.vtypes.Metadata.display_text2

Metadata.**display_text2: str**

vstt.vtypes.Metadata.display_text3

Metadata.**display_text3: str**

vstt.vtypes.Metadata.display_text4

Metadata.**display_text4: str**

vstt.vtypes.Metadata.display_duration

Metadata.**display_duration: float**

vstt.vtypes.Metadata.show_delay_countdown

Metadata.**show_delay_countdown: bool**

vstt.vtypes.Metadata.enter_to_skip_delay

Metadata.**enter_to_skip_delay: bool**

16.9.3 vstt.vtypes.Trial

class vstt.vtypes.**Trial**

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, default is returned if given, otherwise KeyError is raised
popitem()	Remove and return a (key, value) pair as a 2-tuple.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

vstt.vtypes.Trial.__init__

Trial.**__init__**(**args*, ***kwargs*)

vstt.vtypes.Trial.clear

Trial.**clear**() → None. Remove all items from D.

vstt.vtypes.Trial.copy

Trial.**copy**() → a shallow copy of D

vstt.vtypes.Trial.fromkeys

Trial.**fromkeys**(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

vstt.vtypes.Trial.get

Trial.**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.Trial.items

Trial.**items**() → a set-like object providing a view on D's items

vstt.vtypes.Trial.keys

Trial.**keys**() → a set-like object providing a view on D's keys

vstt.vtypes.Trial.pop

Trial.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, default is returned if given, otherwise KeyError is raised

vstt.vtypes.Trial.popitem

Trial.**popitem**()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

vstt.vtypes.Trial.setdefault

Trial.**setdefault**(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

vstt.vtypes.Trial.update

Trial.**update**([*E*,]***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

vstt.vtypes.Trial.values

`Trial.values()` → an object providing a view on D's values

Attributes

weight
condition_timeout
num_targets
target_order
target_indices
add_central_target
hide_target_when_reached
show_target_labels
target_labels
fixed_target_intervals
target_duration
central_target_duration
pre_target_delay
pre_central_target_delay
pre_first_target_extra_delay
target_distance
target_size
central_target_size
show_inactive_targets
ignore_incorrect_targets
play_sound
use_joystick

continues on next page

Table 1 – continued from previous page

joystick_max_speed
show_cursor
cursor_size
show_cursor_path
automove_cursor_to_center
freeze_cursor_between_targets
cursor_rotation_degrees
post_trial_delay
post_trial_display_results
post_block_delay
post_block_display_results
show_delay_countdown
enter_to_skip_delay

vstt.vtypes.Trial.weight

Trial.**weight: int**

vstt.vtypes.Trial.condition_timeout

Trial.**condition_timeout: float**

vstt.vtypes.Trial.num_targets

Trial.**num_targets: int**

vstt.vtypes.Trial.target_order

Trial.**target_order: str | list**

vstt.vtypes.Trial.target_indices

Trial.**target_indices: str**

vstt.vtypes.Trial.add_central_target

Trial.**add_central_target: bool**

vstt.vtypes.Trial.hide_target_when_reached

Trial.**hide_target_when_reached: bool**

vstt.vtypes.Trial.show_target_labels

Trial.**show_target_labels: bool**

vstt.vtypes.Trial.target_labels

Trial.**target_labels: str**

vstt.vtypes.Trial.fixed_target_intervals

Trial.**fixed_target_intervals: bool**

vstt.vtypes.Trial.target_duration

Trial.**target_duration: float**

vstt.vtypes.Trial.central_target_duration

Trial.**central_target_duration: float**

vstt.vtypes.Trial.pre_target_delay

Trial.**pre_target_delay: float**

vstt.vtypes.Trial.pre_central_target_delay

Trial.**pre_central_target_delay: float**

vstt.vtypes.Trial.pre_first_target_extra_delay

Trial.**pre_first_target_extra_delay: float**

vstt.vtypes.Trial.target_distance

Trial.**target_distance: float**

vstt.vtypes.Trial.target_size

Trial.**target_size: float**

vstt.vtypes.Trial.central_target_size

Trial.**central_target_size: float**

vstt.vtypes.Trial.show_inactive_targets

Trial.**show_inactive_targets: bool**

vstt.vtypes.Trial.ignore_incorrect_targets

Trial.**ignore_incorrect_targets: bool**

vstt.vtypes.Trial.play_sound

Trial.**play_sound: bool**

vstt.vtypes.Trial.use_joystick

Trial.**use_joystick: bool**

vstt.vtypes.Trial.joystick_max_speed

Trial.**joystick_max_speed: float**

vstt.vtypes.Trial.show_cursor

Trial.**show_cursor: bool**

vstt.vtypes.Trial.cursor_size

Trial.**cursor_size: float**

vstt.vtypes.Trial.show_cursor_path

Trial.**show_cursor_path: bool**

vstt.vtypes.Trial.automove_cursor_to_center

Trial.**automove_cursor_to_center: bool**

vstt.vtypes.Trial.freeze_cursor_between_targets

Trial.**freeze_cursor_between_targets: bool**

vstt.vtypes.Trial.cursor_rotation_degrees

Trial.**cursor_rotation_degrees: float**

vstt.vtypes.Trial.post_trial_delay

Trial.**post_trial_delay: float**

vstt.vtypes.Trial.post_trial_display_results

Trial.**post_trial_display_results: bool**

vstt.vtypes.Trial.post_block_delay

Trial.**post_block_delay: float**

vstt.vtypes.Trial.post_block_display_results

Trial.**post_block_display_results: bool**

vstt.vtypes.Trial.show_delay_countdown

Trial.**show_delay_countdown: bool**

vstt.vtypes.Trial.enter_to_skip_delay

Trial.**enter_to_skip_delay: bool**

16.10 vstt.vis

Functions

display_drawables(display_time_seconds, ...)	
display_results(display_time_seconds, ...[, ...])	
draw_and_flip(win, drawables, kb[, kb_stop_key])	
get_successful_target_fraction(stats_df, dest)	get successful target fraction
get_successful_trial_fraction(stats_df, dest)	get successful trial fraction
make_cursor(window, cursor_size)	
make_target_labels(window, n_circles, ...)	
make_targets(window, n_circles, radius, ...)	
splash_screen(display_time_seconds, ...[, win])	
update_target_colors(targets, ...[, index])	
update_target_label_colors(target_labels, ...)	

16.10.1 vstt.vis.display_drawables

vstt.vis.**display_drawables**(*display_time_seconds: float*, *enter_to_skip_delay: bool*, *show_delay_countdown: bool*, *drawables: list[BaseVisualStim]*, *win: Window*, *close_window_when_done: bool*, *mouse: Mouse | None = None*, *mouse_pos: tuple[float, float] | None = None*, *return_screenshot: bool = False*) → Image | None

16.10.2 vstt.vis.display_results

vstt.vis.**display_results**(*display_time_seconds: float*, *enter_to_skip_delay: bool*, *show_delay_countdown: bool*, *trial_handler: TrialHandlerExt | None*, *display_options:* DisplayOptions, *i_trial: int*, *all_trials_for_this_condition: bool*, *win: Window | None = None*, *mouse: Mouse | None = None*, *mouse_pos: tuple[float, float] | None = None*, *return_screenshot: bool = False*) → Image | None

16.10.3 vstt.vis.draw_and_flip

vstt.vis.**draw_and_flip**(*win: Window*, *drawables: list[BaseVisualStim]*, *kb: Keyboard | None*, *kb_stop_key: str = 'escape'*) → None

16.10.4 vstt.vis.get_successful_target_fraction

vstt.vis.**get_successful_target_fraction**(*stats_df: DataFrame*, *dest: str*) → float

get successful target fraction

Parameters

- **stats_df** – stats dataframe
- **dest** – "target" or "center"

Returns

successful target fraction

16.10.5 vstt.vis.get_successful_trial_fraction

vstt.vis.**get_successful_trial_fraction**(*stats_df: DataFrame*, *dest: str*) → float

get successful trial fraction

Parameters

- **stats_df** – stats dataframe
- **dest** – "target" or "center"

Returns

successful trial fraction

16.10.6 vstt.vis.make_cursor

vstt.vis.**make_cursor**(*window: Window*, *cursor_size: float*) → ShapeStim

16.10.7 vstt.vis.make_target_labels

vstt.vis.**make_target_labels**(*window: Window*, *n_circles: int*, *radius: float*, *point_radius: float*, *labels_string: str*) → list[TextBox2]

16.10.8 vstt.vis.make_targets

vstt.vis.**make_targets**(*window: Window*, *n_circles: int*, *radius: float*, *point_radius: float*, *add_central_target: bool*, *center_point_radius: float*) → ElementArrayStim

16.10.9 vstt.vis.splash_screen

vstt.vis.**splash_screen**(*display_time_seconds: float*, *enter_to_skip_delay: bool*, *show_delay_countdown: bool*, *metadata:* Metadata, *win: Window | None = None*) → None

16.10.10 vstt.vis.update_target_colors

vstt.vis.**update_target_colors**(*targets: ElementArrayStim*, *show_inactive_targets: bool*, *index: int | None = None*) → None

16.10.11 vstt.vis.update_target_label_colors

vstt.vis.**update_target_label_colors**(*target_labels: list[TextBox2]*, *show_inactive_targets: bool*, *index: int | None = None*) → None

Exceptions

MotorTaskCancelledByUser

16.10.12 vstt.vis.MotorTaskCancelledByUser

exception vstt.vis.**MotorTaskCancelledByUser**

PYTHON MODULE INDEX

V

INDEX

Symbols

A

C

D

E

F

R

S

T

U

V

W