thumbs project

Juli Furjes

10/29/2021

Mouse Tracking using MouseTrap - wuhu!

Install the package 'mousetrap' and load the library.

```
# loading packages
pacman::p_load(mousetrap, tidyverse, plyr)
```

Load the data files into R and briefly inspect it.

```
df <- read_csv('subject-4.csv')
```

```
## Rows: 8 Columns: 94

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (44): acc, accuracy, average_response_time, avg_rt, background, bidi, ca...
## dbl (46): compensation, correct, correct_mouse, count_block_loop, count_bloc...
## lgl  (4): response_end_of_experiment, response_time_end_of_experiment, time_...

##
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
myfiles <-  list.files(pattern="*.csv", full.names=TRUE)
df <-  ldply(myfiles, read_csv)
```

```
## Rows: 8 Columns: 95

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (45): acc, accuracy, average_response_time, avg_rt, background, bidi, ca...
## dbl (46): compensation, correct, correct_mouse, count_block_loop, count_bloc...
## lgl  (4): response_end_of_experiment, response_time_end_of_experiment, time_...

##
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Rows: 8 Columns: 94

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (44): acc, accuracy, average_response_time, avg_rt, background, bidi, ca...
## dbl (46): compensation, correct, correct_mouse, count_block_loop, count_bloc...
## lgl  (4): response_end_of_experiment, response_time_end_of_experiment, time_...
```

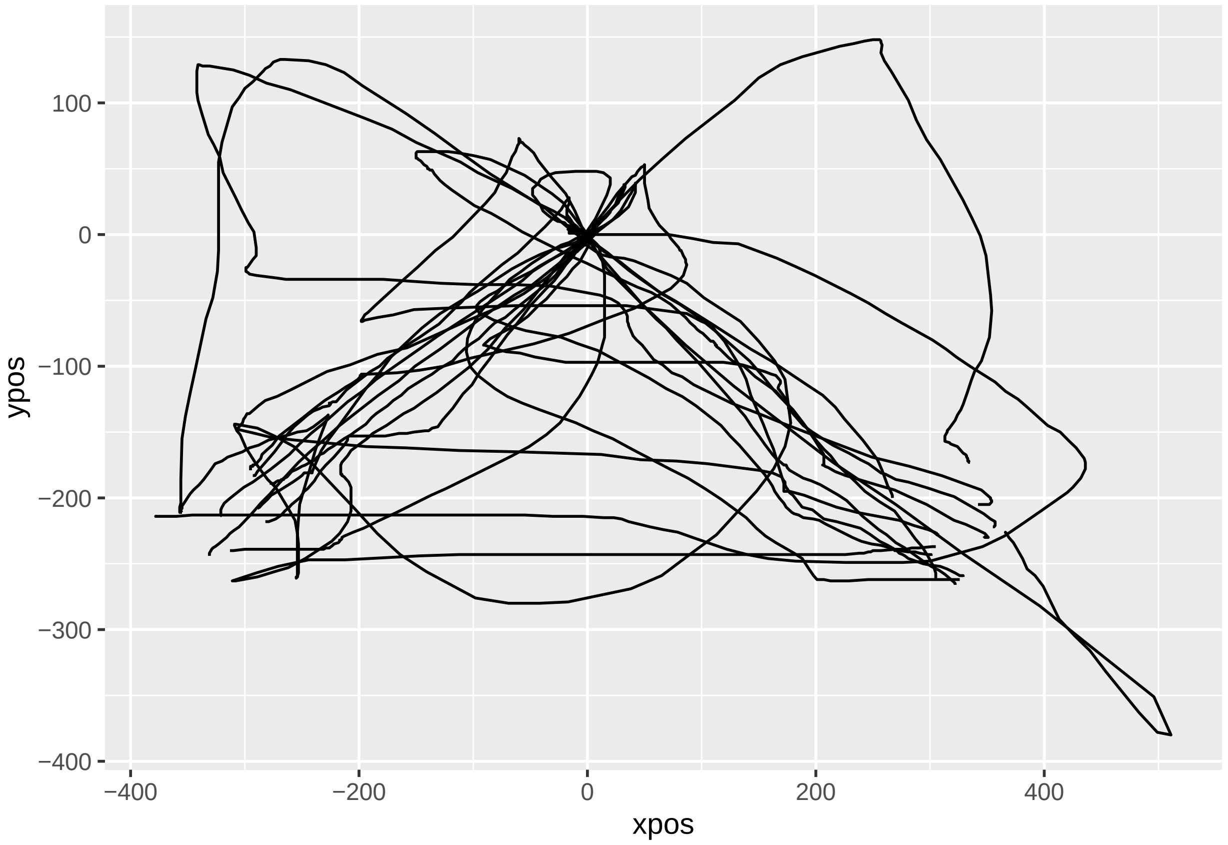
```
##
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Rows: 8 Columns: 94

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (44): acc, accuracy, average_response_time, avg_rt, background, bidi, ca...
## dbl (46): compensation, correct, correct_mouse, count_block_loop, count_bloc...
## lgl  (4): response_end_of_experiment, response_time_end_of_experiment, time_...

##
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1. Turn the raw data into a mousetrap object. Now take a moment to look at the structure that it created. What are the elements, what information does it provide?

```r
m <- mt_import_mousetrap(df)
```

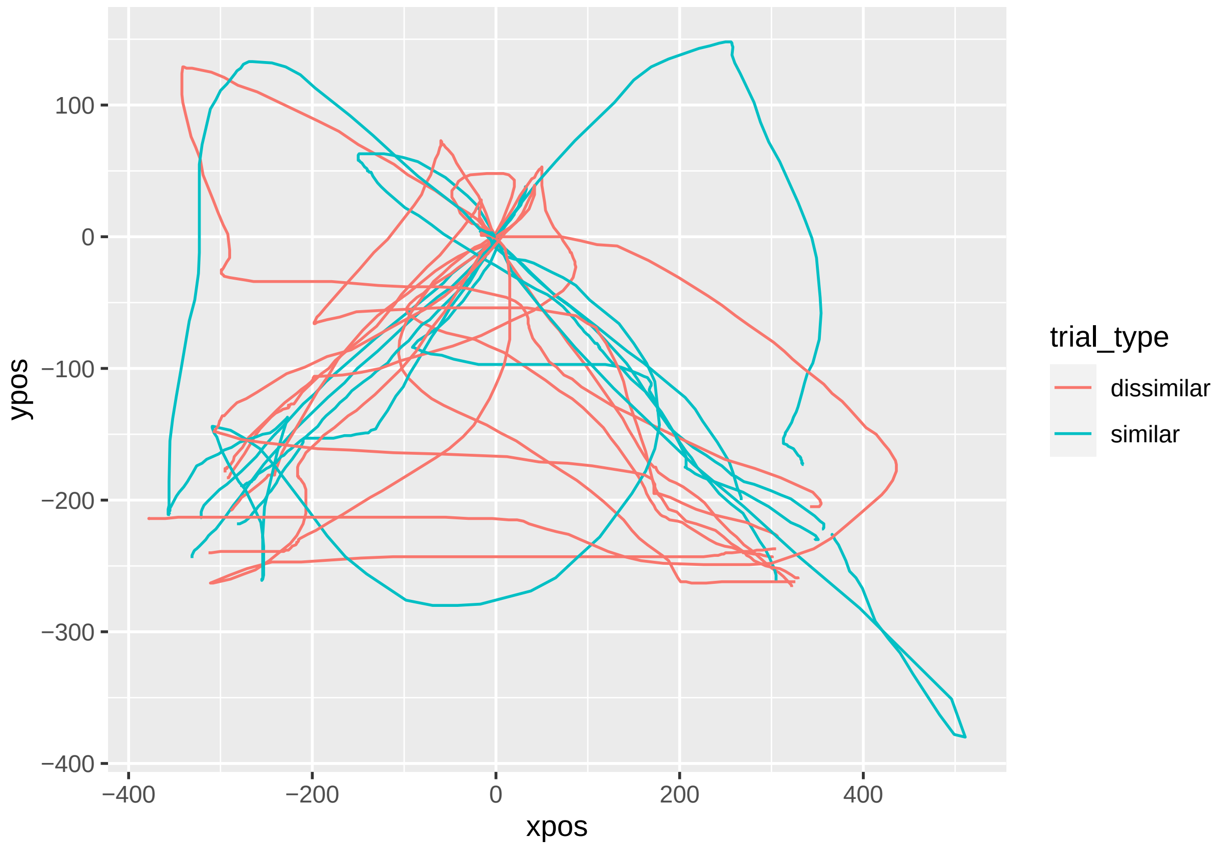
2. Make a plot. Try to understand what the lines mean and how to "read" the graph. What seems weird about it when you think about how the experiment task looked like?

```r
# initial plot
mt_plot(m)
```



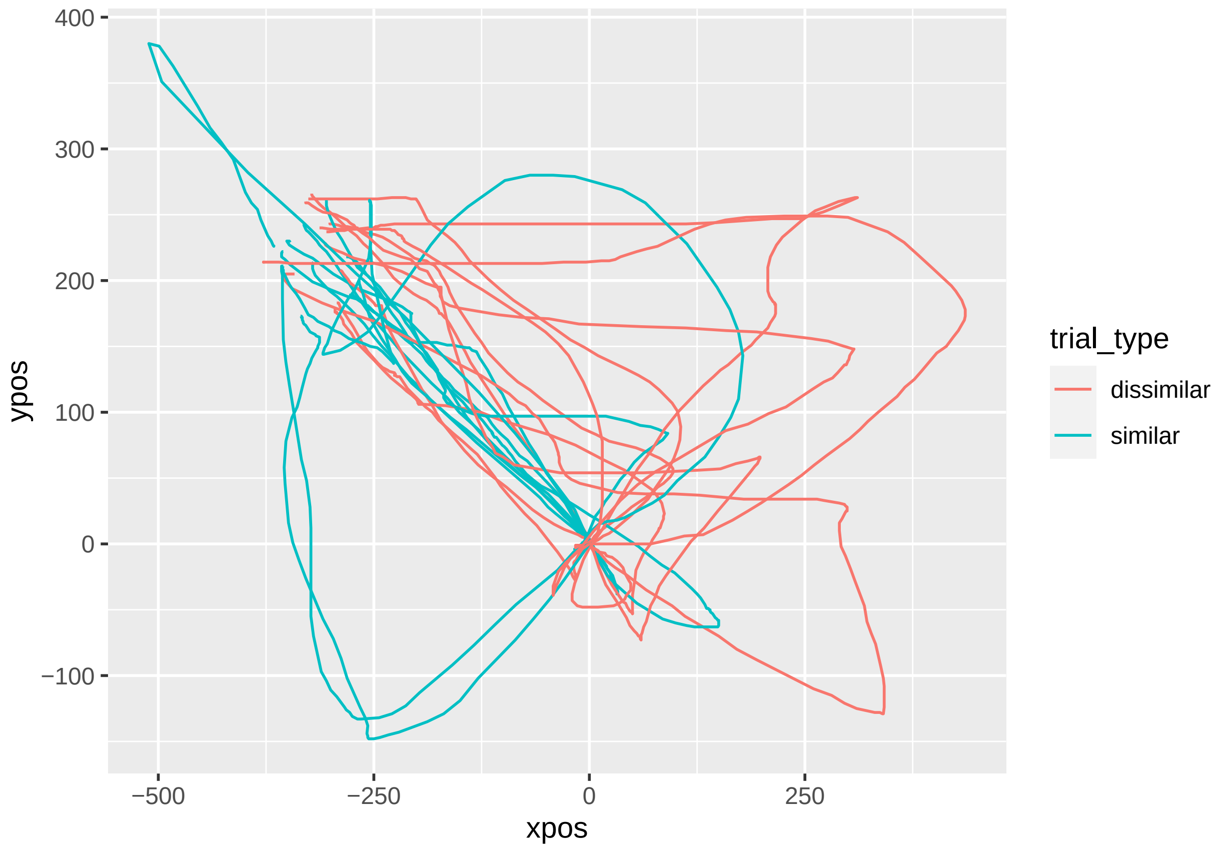
3.
The demo-experiment had different trial types for which we have different predictions. Make a plot that distinguishes these two conditions, e.g. by different colors.

```r
# plotting and distinguishing by trial type
mt_plot(
  m,
  color = 'trial_type' # the magic happens here
)
```



4. Find a function that does a mirror-symmetric mapping of all the movements from the right side to the left side so that all movements overlap. Plot again. What does this function also do? If you've done everything correctly, your figure should something like what you can find in the exercise sheet under task 4.

```r
# align the mouse trajectories to one side
m <- mt_remap_symmetric(
  m,
  use = 'trajectories',
  remap_xpos = "left"
  )
# plot again
mt_plot(
  m,
  use = 'trajectories',
  color = 'trial_type'
  )
```

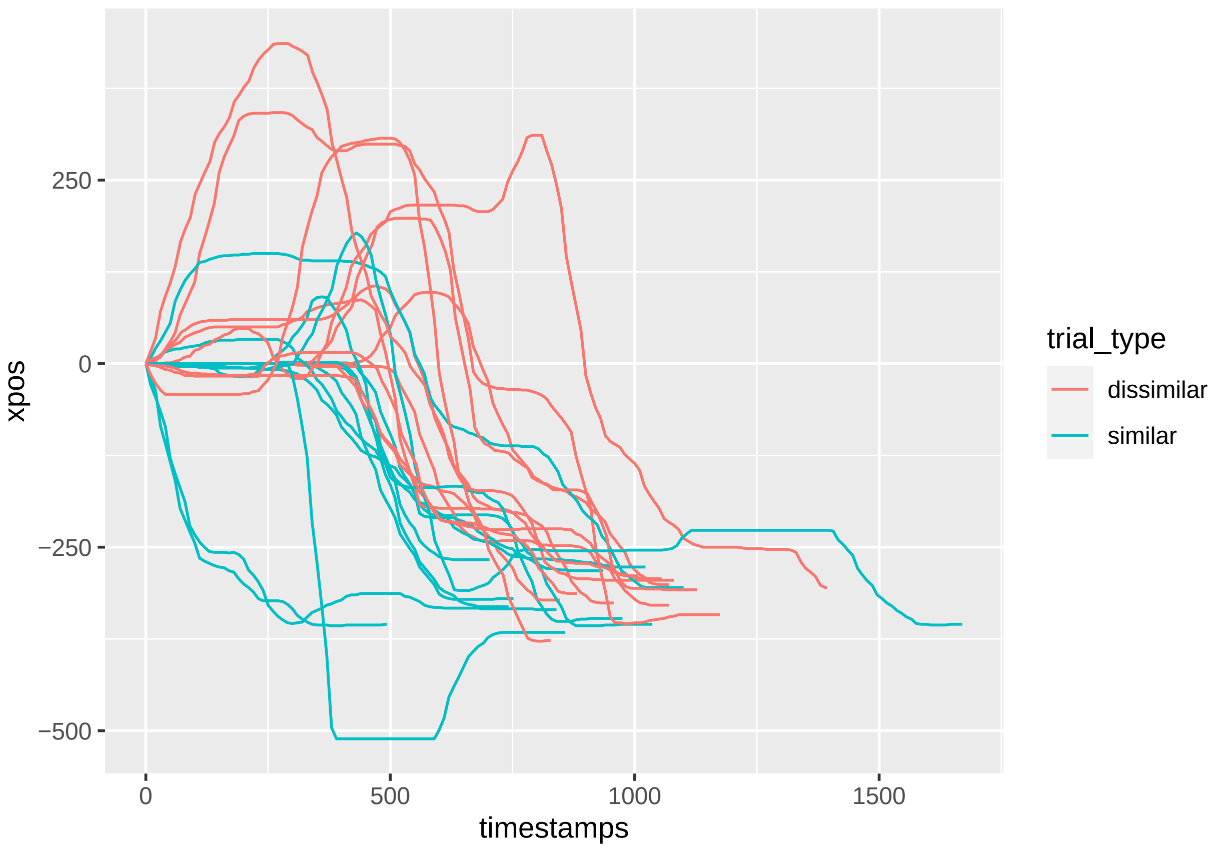


```r
# turns the data upside down, so it resembles the actual mouse tracking task
```

5. The standard plotting function shows x and y coordinates. Modify it so you plot timestamps by xpos. What do you see? What is this line in the beginning?

The line in the beginning is the initial phase of a trial without mouse movement.

```r
# plot with timestamps
mt_plot(
  m,
  x = 'timestamps', # modifying the x-axis to plot timestamps
  y = 'xpos',       # modifying the y-axis to plot the xpos
  use = 'trajectories',
  color = 'trial_type'
)
```

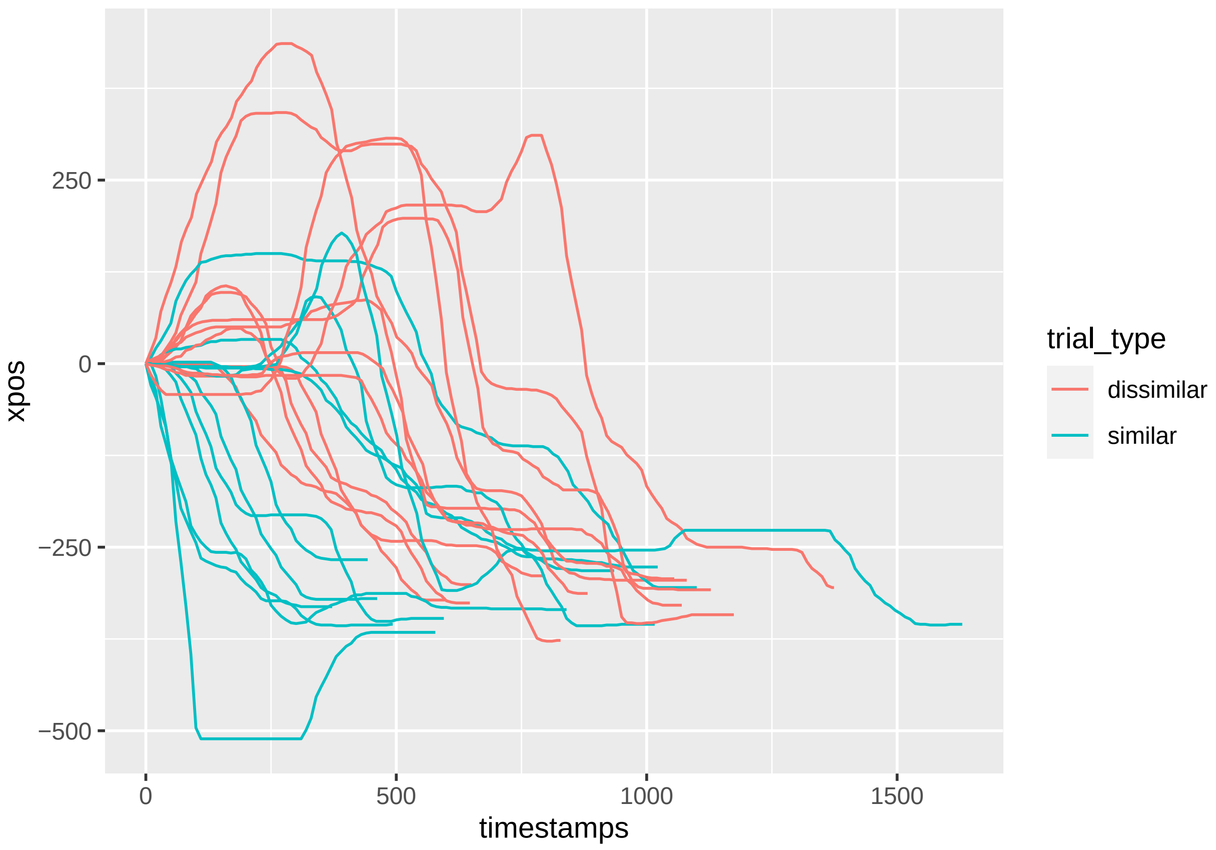


6. Find a function to remove this "line" and plot again.

```
m <- mt_exclude_initiation(m)
```

If you now plotted x and y coordinates again, what would have changed in the plot? Think first, then try it!

```
mt_plot(
  m,
  x = 'timestamps',
  y = 'xpos',
  use = 'trajectories',
  color = 'trial_type'
)
```



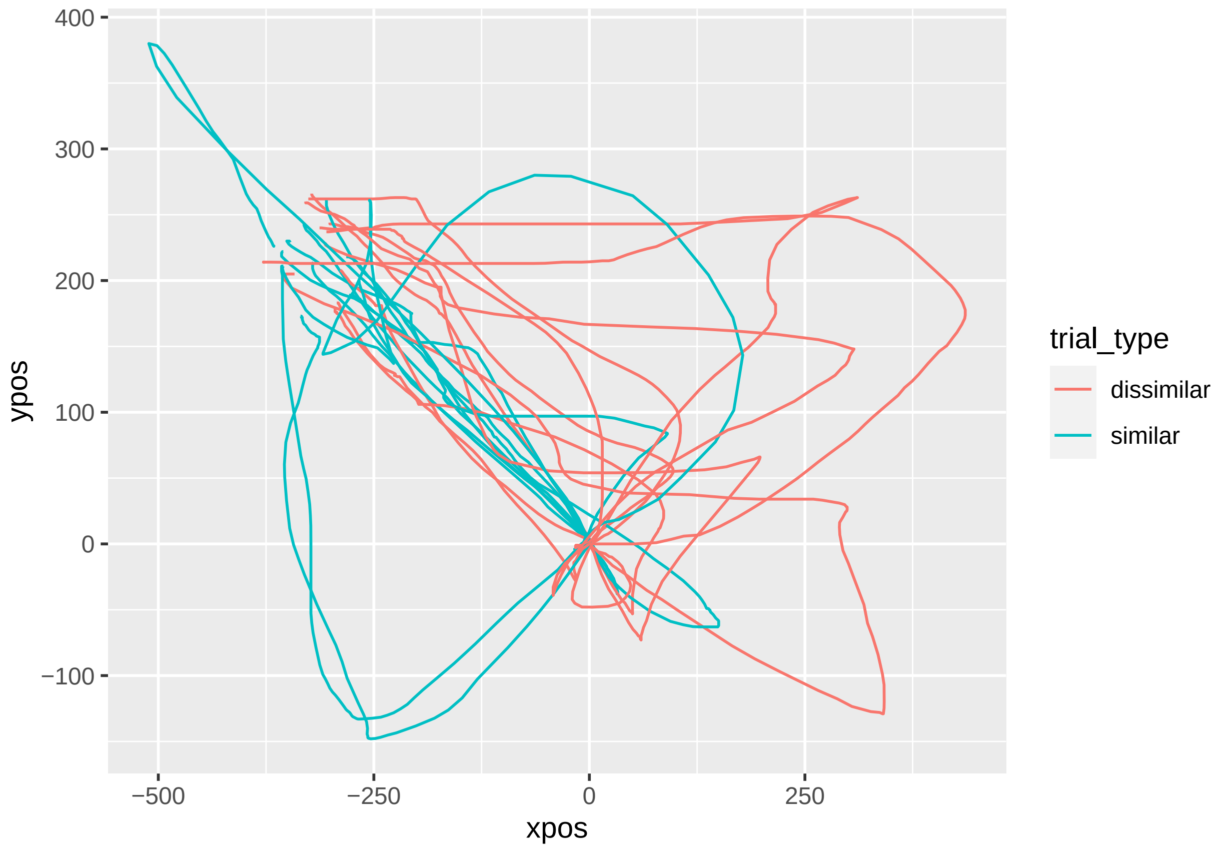
Please call Sigrid over for a brief check-in!

7. Apply the function "mt_time_normalize". Now look at your data variable (the mt object) where a new matrix appeared. What do you notice? What does the function do? After you have thought about it yourself, check the documentation.

```
m <- mt_time_normalize(m)
```

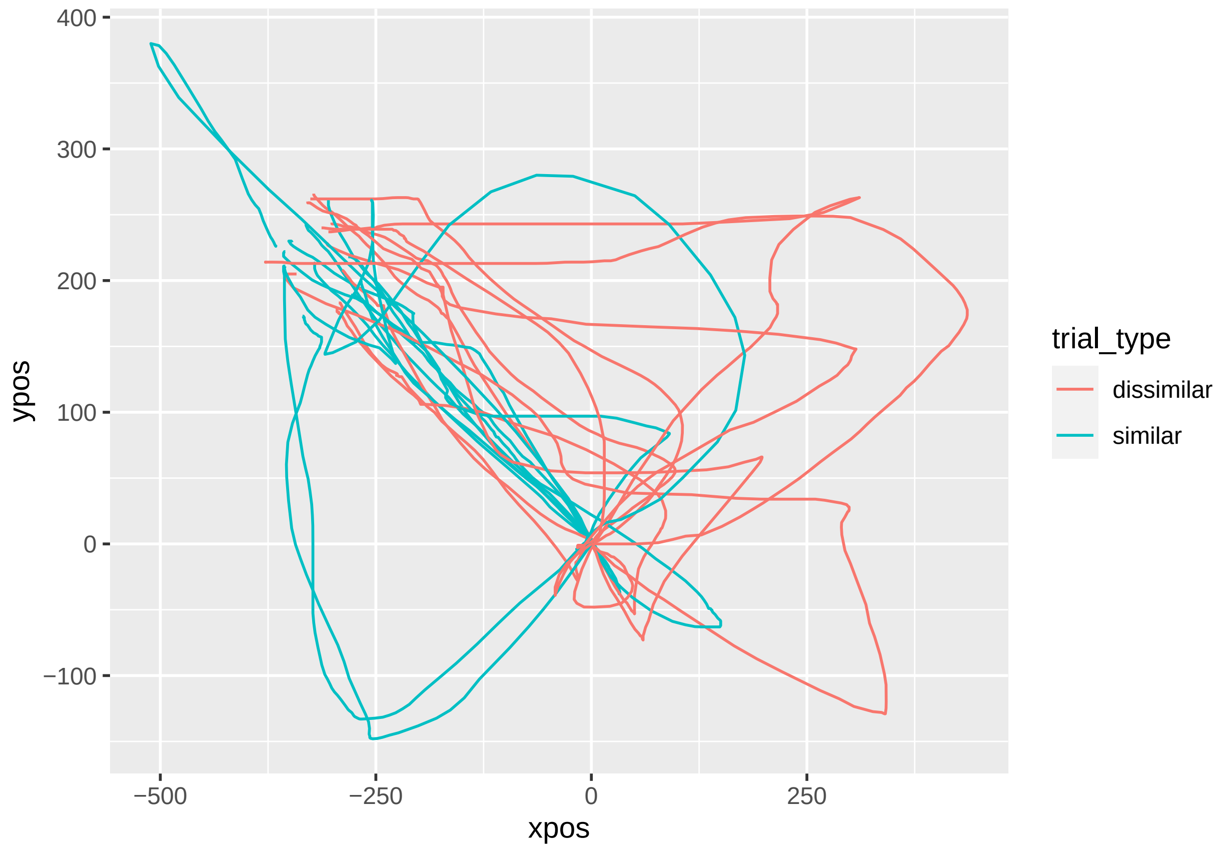
8. Find out how to plot the normalized trajectories instead of the raw data.

```
# time normalized plot
mt_plot(
  m,
  use = 'tn_trajectories',
  color = 'trial_type'
)
```



9. Take a moment to play around with different numbers of steps in "mt_time_normalize" and see how that changes the shape of the trajectories, esp. when using very small numbers. Discuss what the decision about normalizing implies for the data analysis and interpretation. In the end, set the steps back to the default of 101.

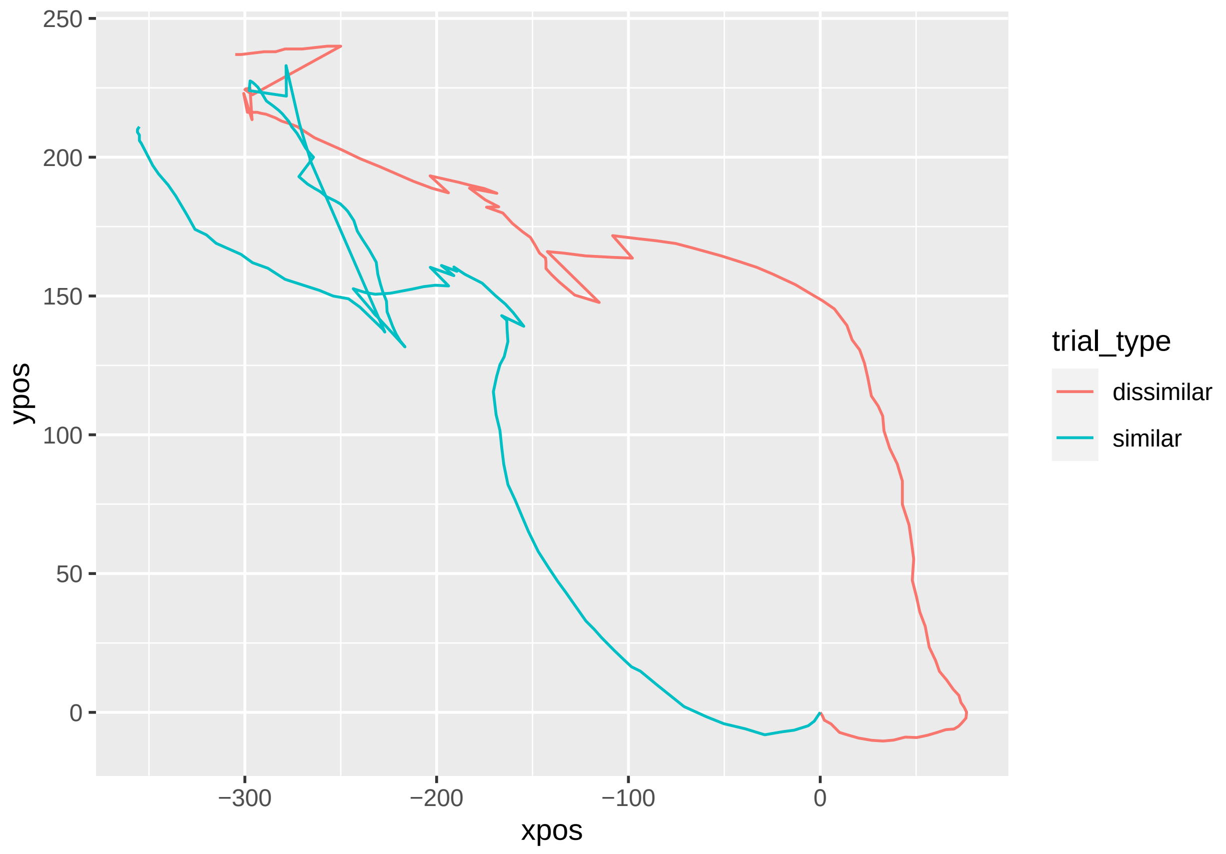
```
m_tn <- mt_time_normalize(
  m,
  nsteps = 101
)
#plot
mt_plot(
  m_tn,
  use = 'tn_trajectories',
  color = 'trial_type'
)
```



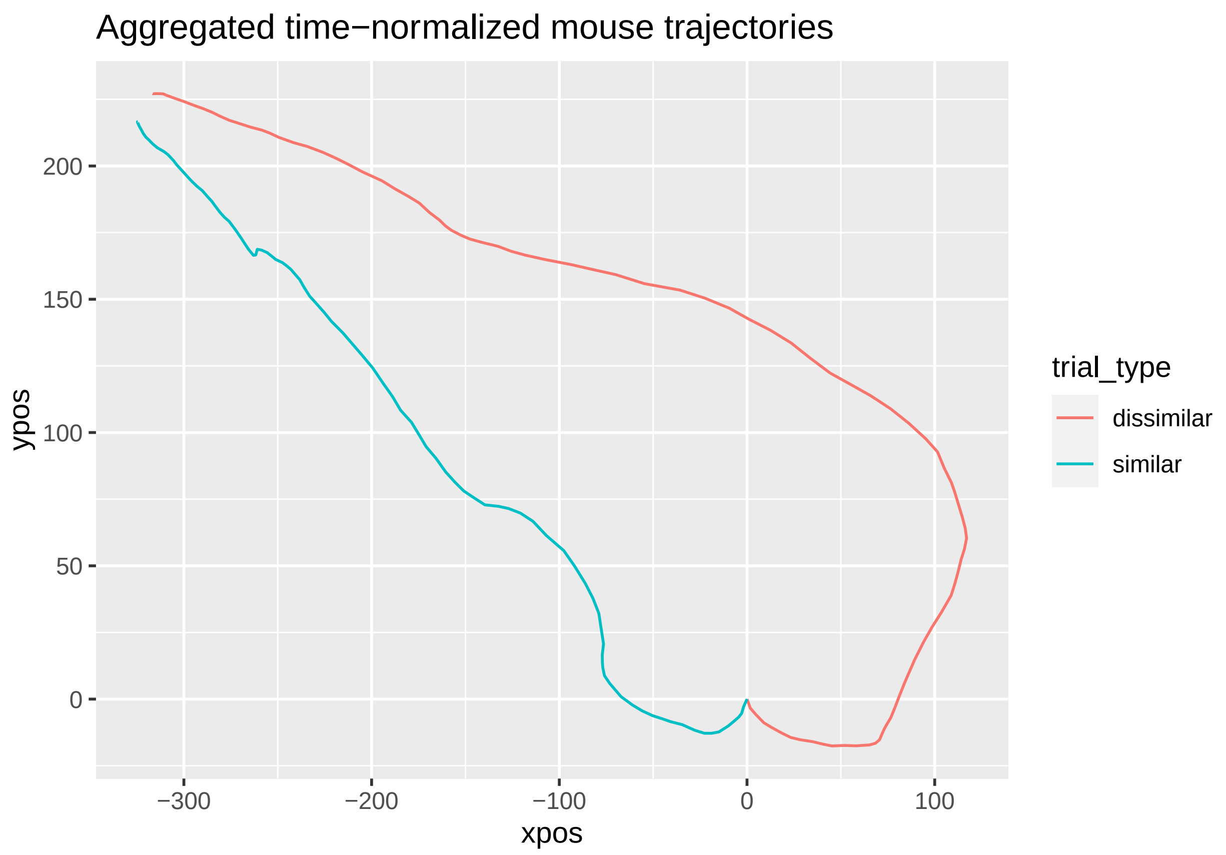
10. Now we want to visualize our “findings”. Find a function that will plot averages of all the “similar” movements and all the “dissimilar” movements. Think: Which trajectories do we need to use, the original or the time normalized? Why? Try plotting both to see whether you were right.

```
mt_plot_aggregate(
  m,
  use = 'trajectories',
  color = 'trial_type'
)
```

```
## Warning in mt_reshape(data = data, use = use, use_variables = use_variables, :
## Trajectories differ in the number of logs. Aggregate trajectory data may be
## incorrect.
```



```
mt_plot_aggregate(
  m,
  use = 'tn_trajectories',
  color = 'trial_type'
) +
  labs(
    title = 'Aggregated time-normalized mouse trajectories')
```



11. Apply the function "mt_measures" and look at the outcome in your data variable.

```
m <- mt_measures(
  m,
  use = 'tn_trajectories')
```

12. Now find a function that helps you aggregate some measures of your pleasing over the two trial_types. See example in the exercise sheet under task 12.

```
# aggregating the data by trial type
mt_measures_ag <- mt_aggregate(
  m,
  use = 'measures',
  use_variables = c('MAD', 'xpos_flips','AUC', 'RT'), # if you want all of the measures, exclude this l
  use2_variables = 'trial_type')
```

13. Take a moment to think what these "results" could mean if this was a proper experiment and didn't just have the experimenter herself run it once ;-) How would you interpret this? Does this match your hypothesis? Especially look at the MAD (maximum absolute deviation from the direct path), the AUC (area under the curve) and the reaction time.

When you've come this far, please call me over for a brief check-in!

CHALLENGE LEVEL

1. Normally, you'd want to remove incorrect trials. As the mouse_trap object does not function with tidyverse, figure out how to remove potential incorrect trials from your mousetrap object.

```
m <- mt_subset(
  m,
  correct == '1'
)
```

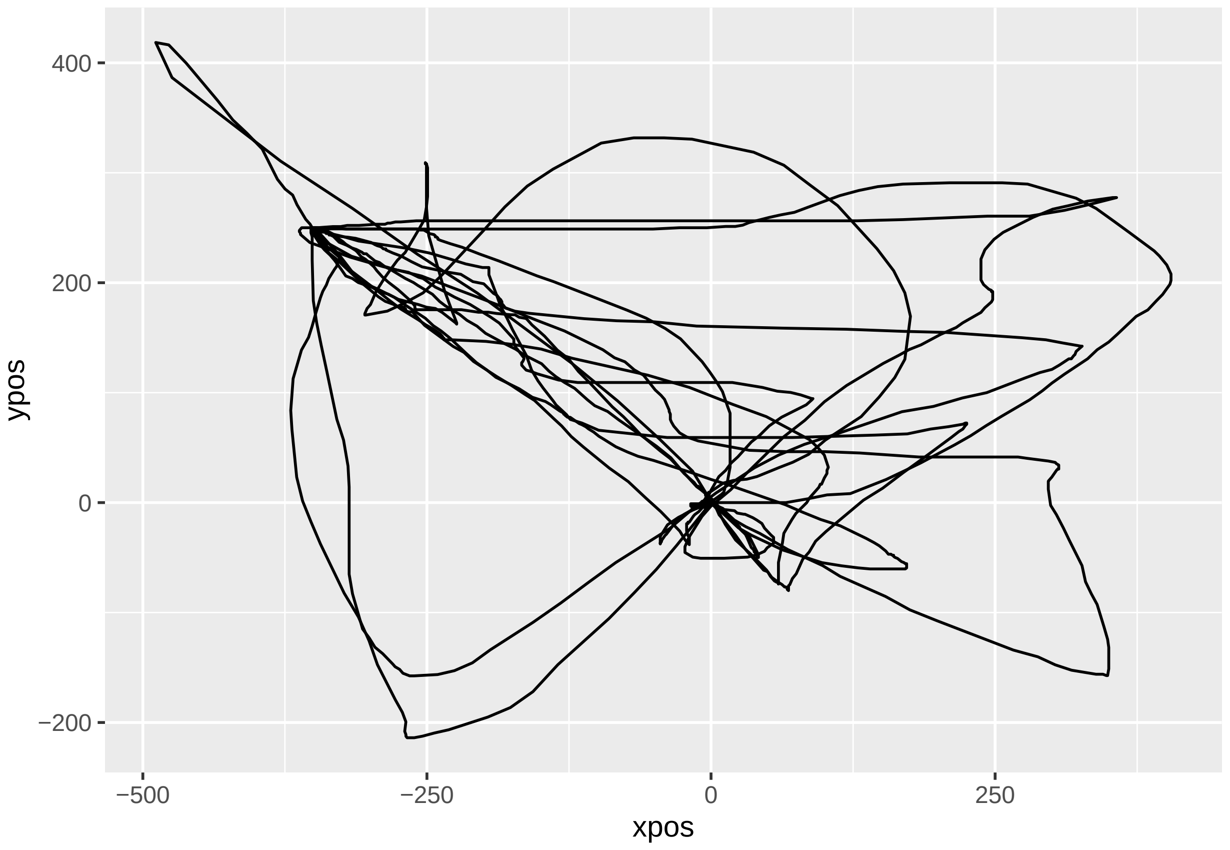
2. Would the function ‘mt_align’ be useful for this data? Why or why not?

It can do the same as mt_remap symmetric, but it can also be used to rescale (here ’space-normalize) the data.

You could align the starting and end position, as we are only interested in the trajectory of the mouse movement, not the endpoint in it self. Also, because the end point could be anywhere within the stimulus (circle or square), it looks nice to drag the trajectories to the same point.

As we only have one participant, I wouldn’t say it is necessary (or beneficial, besides it looks nice).

```
# an example of aligning the data
m_align <- mt_align(
  m,
  use = 'trajectories',
  dimensions = c("xpos", "ypos"),
  coordinates = c(0,0,-350,250),
  align_start = T,
  align_end = T
)

  # plot again
mt_plot(
  m_align,
  use = 'trajectories',
)
```


ypos
xpos