

USC Institute for  
Creative Technologies

# Proxemics Guide

REU 2017

T.A. Nguyen  
tnguyen@ict.usc.edu

## Summary

---

Proxemics is a Python package that was designed for analyzing positional data of users interacting with Brad and Rachel. It leverages a powerful data analysis library called pandas (version 0.20.3). The primary functions of Proxemics are calculating and plotting position and speed.

## How Proxemics.py developed

---

Proxemics was developed over the course of nearly 6 weeks during the summer of REU 2017 under the guidance of Dr. Krum at the Mixed Reality Lab. The library makes use of pandas and matplotlib to organize tracking data collected in the Fall of 2016.

Developing Proxemics involved several major hurdles including:

- Learning pandas and using pandas functions correctly (many hours were spent reading forums – Reddit, Quora, Stack Overflow- to produce desired graphs with the desired characteristics).
- Learning matplotlib and a bit about numpy. This process likewise involved many, many hours reading documentation and testing out code.
- Defining questions of interest and letting the data inform what questions are worth asking
- Determining what constitutes stillness and what constitutes movement (setting 0 m/s as stillness didn't work as even a slight head wobble could result in a speed > 0 m/s).
- Calculating speed and setting an appropriate threshold speed to detect user motion
- Last but not least, testing and debugging the library to make sure it's robust enough to handle varying data

# Pandas in a nutshell

---

Pandas is an open source data analysis library that is suitable for the following needs:

- Reading and writing data from CSV, txt and other file formats
- Aligning, indexing, and reshaping data
- Performing operations on large data sets (pivoting, merging, etc)

Pandas offers two data structures to organize data. A pandas **Series** may be thought of as a labeled Python dict. The following example from official pandas documentation illustrates a Series:

```
In [3]: s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

```
In [4]: s
```

```
Out[4]:
```

```
a    0.2941
b    0.2869
c    1.7098
d   -0.2126
e    0.2696
dtype: float64
```

The index parameter allows users to label individual data points. To access data, instead of relying on integer indices (like in a Python list), you can access data using the custom index. Pretty cool!

```
In [16]: s['a']
```

```
Out[16]: 0.29413876297575337
```

For large and complex data sets, pandas provides a **DataFrame** data structure. A DataFrame is a 2D object that resembles a spreadsheet. Rows (called the 'index') and columns make up a DataFrame, and are used to access data in the set. A DataFrame can be made from a list of dicts, a group of Series, and a couple additional methods.

```
In [47]: data2 = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
```

```
In [48]: pd.DataFrame(data2)
```

```
Out[48]:
```

```
   a  b    c
0  1  2  NaN
1  5 10 20.0
```

The method used by Proxemics was `read_csv()`, which automatically parsed thousands of data points from a CSV file into a single DataFrame. This efficiency is a major reason why pandas was used to build Proxemics!

#####	40	PrimingJoystickWalking	Brad
time-sec	distance-meters	hmd-from-origin	vhuman-from-origin
0.004	2.554	2.554	0
0.01	2.554	2.554	0
0.021	2.554	2.554	0
0.032	2.554	2.554	0
0.043	2.554	2.554	0



```
import pandas as pd
df = pd.read_csv("0040_PrimingJoystickWalking.csv", skiprows = 1)
```

```
In [6]: df
Out[6]:
```

	time-seconds	distance-meters	hmd-from-origin	vhuman-from-origin
0	0.004	2.554	2.554	0.000
1	0.010	2.554	2.554	0.000
2	0.021	2.554	2.554	0.000
3	0.032	2.554	2.554	0.000
4	0.043	2.554	2.554	0.000
5	0.055	2.555	2.555	0.000
6	0.066	2.555	2.555	0.000
7	0.077	2.555	2.555	0.000
8	0.088	2.555	2.555	0.000
9	0.099	2.555	2.555	0.000
10	0.111	2.555	2.555	0.000
11	0.122	2.556	2.556	0.000
12	0.133	2.556	2.556	0.000
13	0.144	2.555	2.555	0.000
14	0.155	2.556	2.556	0.000
15	0.166	2.556	2.556	0.000
16	0.178	2.556	2.556	0.000
17	0.189	2.556	2.556	0.000
18	0.200	2.556	2.556	0.000
19	0.211	2.556	2.556	0.000
20	0.222	2.556	2.556	0.000
21	0.234	2.556	2.556	0.000
22	0.245	2.556	2.556	0.000
23	0.256	2.556	2.556	0.000
24	0.267	2.556	2.556	0.000
25	0.278	2.556	2.556	0.000
26	0.289	2.556	2.556	0.000
27	0.301	2.556	2.556	0.000
28	0.312	2.556	2.556	0.000
29	0.323	2.556	2.556	0.000
...	...	...	...	...
42107	476.488	0.494	0.504	0.017
42108	476.499	0.494	0.504	0.017
42109	476.511	0.494	0.504	0.017
42110	476.522	0.494	0.504	0.017
42111	476.533	0.494	0.504	0.017
42112	476.544	0.494	0.504	0.017
42113	476.555	0.494	0.504	0.017
42114	476.566	0.494	0.504	0.017
42115	476.578	0.494	0.504	0.017
42116	476.589	0.494	0.504	0.017
42117	476.600	0.493	0.503	0.017
42118	476.611	0.493	0.503	0.017

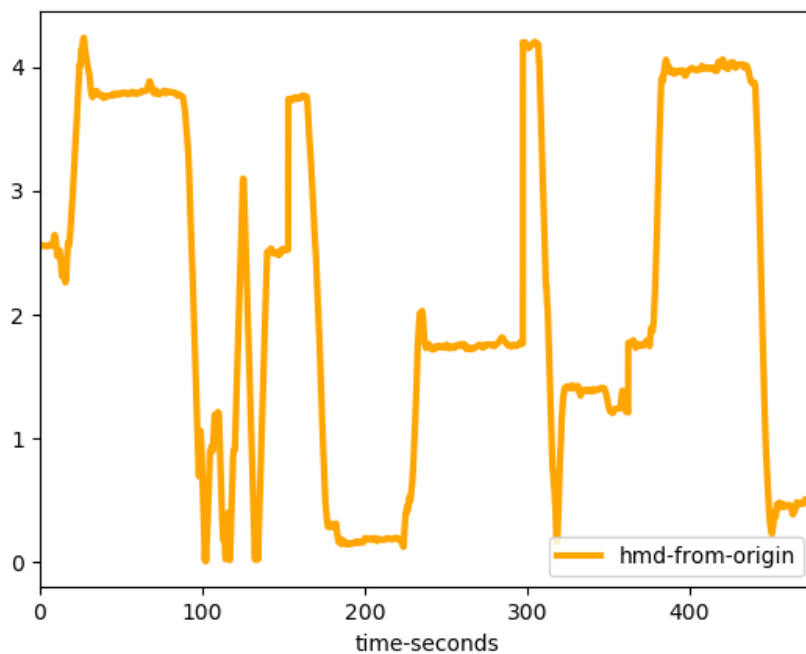
Just as there are multiple ways to instantiate a DataFrame, there are several ways to access data from a DataFrame.

Operation	Syntax	Result
Select column	<code>df[col]</code>	Series
→ Select row by label	<code>df.loc[label]</code>	Series
→ Select row by integer location	<code>df.iloc[loc]</code>	Series
Slice rows	<code>df[5:10]</code>	DataFrame
Select rows by boolean vector	<code>df[bool_vec]</code>	DataFrame

The two slicing methods used for Proxemics are indicated by the arrows. In addition, Proxemics borrows plotting methods from pandas, which provide all the functionality of `matplotlib`.

To generate a plot from a DataFrame, only x and y data labels are required (column names can be supplied, which is a major advantage compared to plotting using `matplotlib`, which requires numpy arrays or lists as inputs). Optional parameters customize the plot (linewidth, title, etc).

```
df.plot('time-seconds', 'hmd-from-origin', c = 'orange', linewidth=3.0)
```



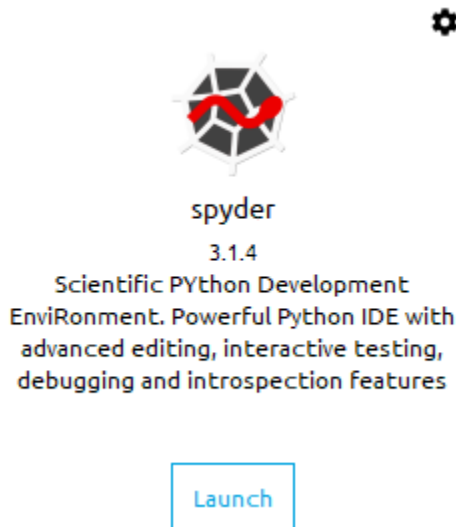
# Tutorial

Proxemics was built using Spyder, a powerful IDE designed for scientific applications. In addition, Spyder was created to support `matplotlib` and features an interactive Python interpreter (IPython). We will be using IPython to execute all our commands. To use the Proxemics library, download Spyder through Anaconda: <https://pythonhosted.org/spyder/installation.html>

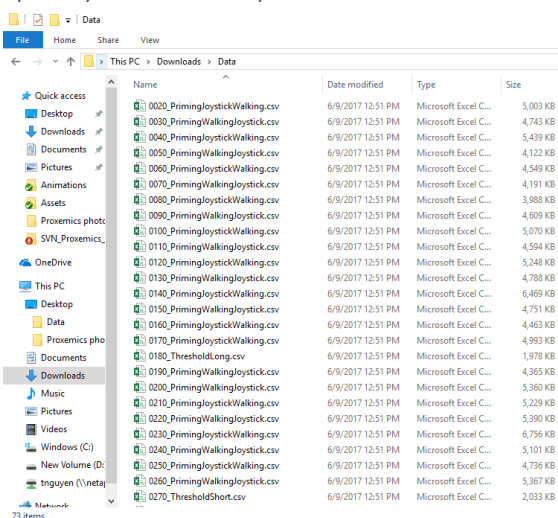
1. After installing Anaconda, search for the Anaconda Navigator. If your machine runs Windows 10, just type in “anaconda” in the Windows search bar. Hit Enter to open the Navigator.



2. Click on the Launch button beneath the Spyder icon. An alternative is to simply type “spyder” into the Windows search bar and launch the IDE.



3. Open `Proxemics.py` in Spyder.
4. Specify the directory where data files are found. Use double backslashes.



Directory of data files

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os, re
*****

CSV files need to match format given in 0040_PrimingJoystickWalking.csv
*****

class Proxemics:
    os.chdir("C:\\Users\\tnguyen.ICT2000\\Downloads\\Data")
    def __init__(self, file):
        self.file = file
        self.df = pd.read_csv(self.file, skiprows = 1)
        if (self.file.endswith("Walking.csv")): # joystick happens first
            self.trials = ("BeBehindDoorJoystick", "BeBehindDoor")
        else:
            self.trials = ("BeBehindDoor", "BeBehindDoorJoystick")

```

os.chdir changes current working directory to the path specified

5. Run Proxemics.py script by pressing the F5 key. A runfile command will appear in the IPython console.

```

IPython console
Console 1/A
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

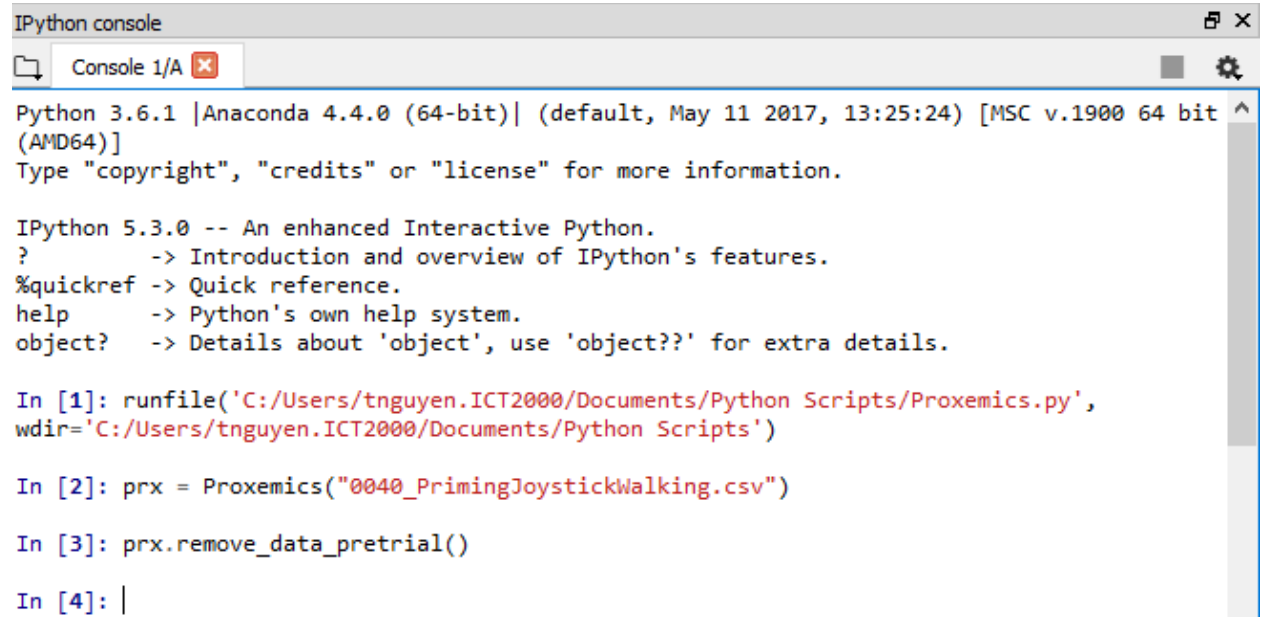
In [1]: runfile('C:/Users/tnguyen.ICT2000/Documents/Python Scripts/Proxemics.py',
wdir='C:/Users/tnguyen.ICT2000/Documents/Python Scripts')

In [2]: |

```

6. Instantiate a Proxemics object by providing a desired file name. To begin data analysis, call `remove_data_pretrial()`.

```
prx = Proxemics("0040_PrimingJoystickWalking.csv")
prx.remove_data_pretrial()
```



```
IPython console
Console 1/A
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/Users/tnguyen.ICT2000/Documents/Python Scripts/Proxemics.py',
wdir='C:/Users/tnguyen.ICT2000/Documents/Python Scripts')

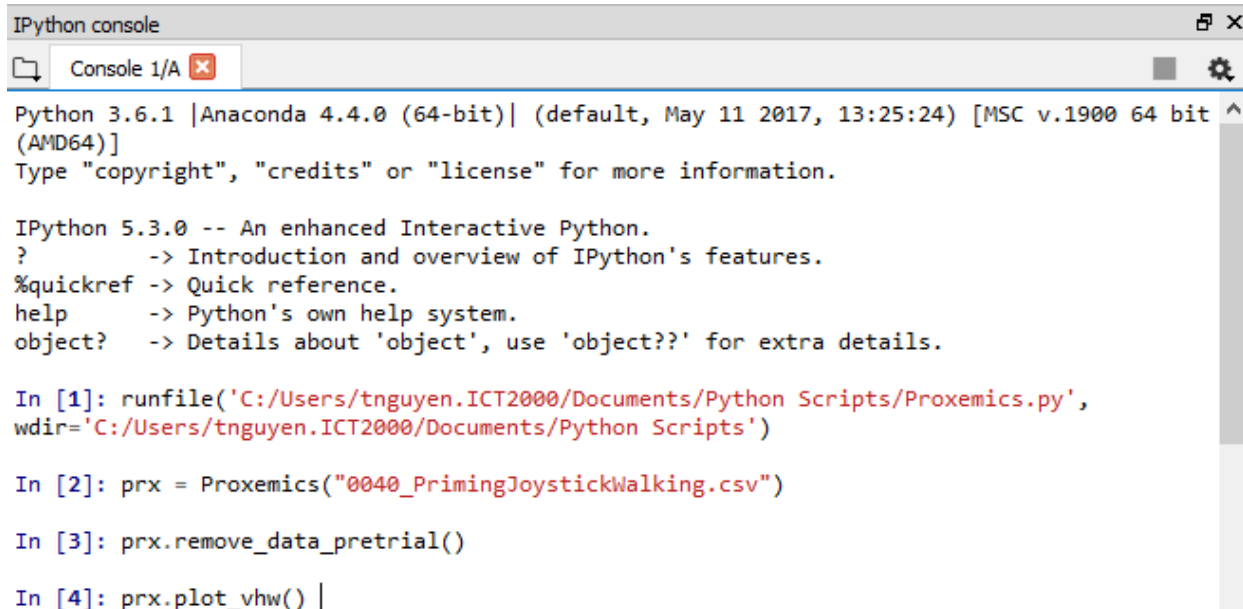
In [2]: prx = Proxemics("0040_PrimingJoystickWalking.csv")

In [3]: prx.remove_data_pretrial()

In [4]: |
```

7. Call any function in Proxemics using dot notation. For example, to plot data from VHumanWalk1 trials, type `[object].plot_vhw()` into IPython.

```
prx.plot_vhw()
```



```
IPython console
Console 1/A
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/Users/tnguyen.ICT2000/Documents/Python Scripts/Proxemics.py',
wdir='C:/Users/tnguyen.ICT2000/Documents/Python Scripts')

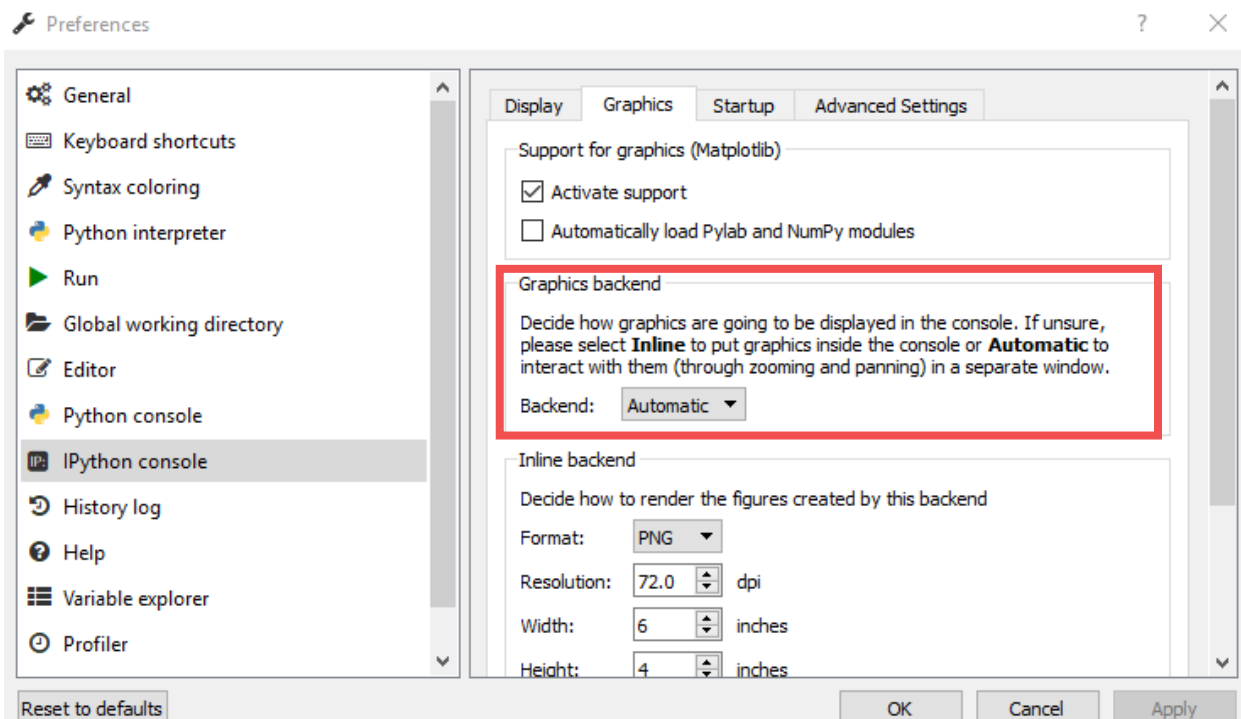
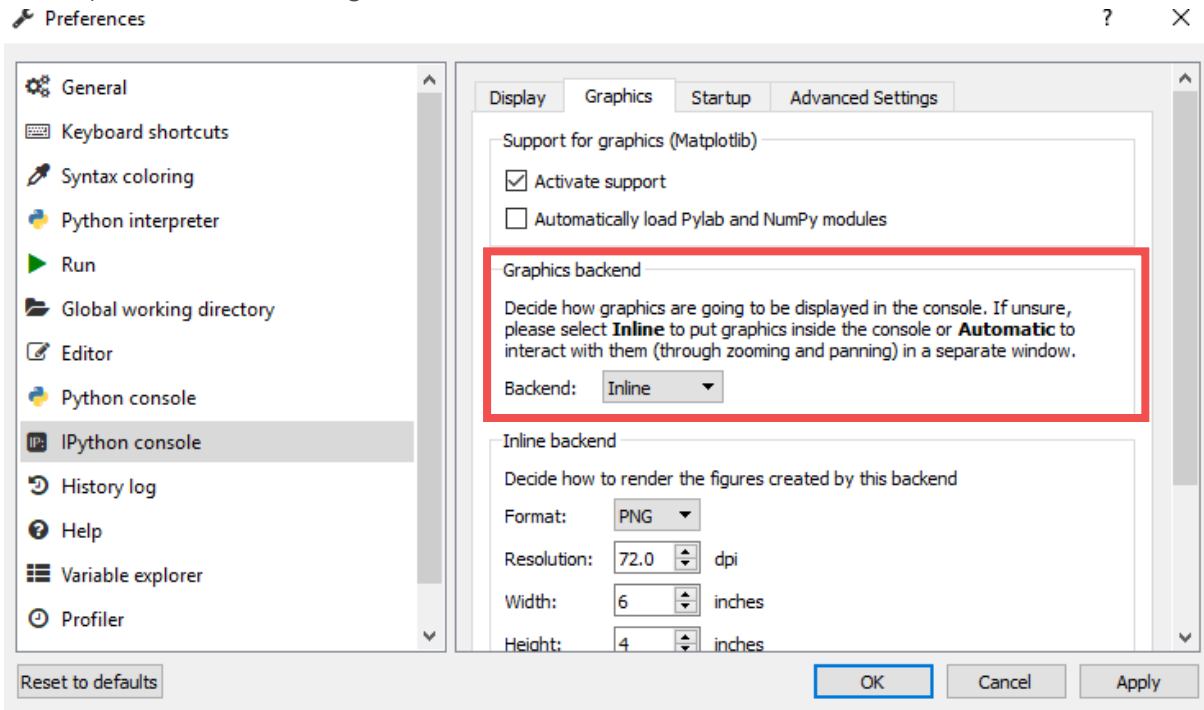
In [2]: prx = Proxemics("0040_PrimingJoystickWalking.csv")

In [3]: prx.remove_data_pretrial()

In [4]: prx.plot_vhw() |
```



8. By default, plots created in Spyder will appear inside the IPython console. To make plots appear as separate windows, go to Tools > Preferences > IPython console. Navigate to the Graphics tab and change "Inline" to "Automatic". Hit OK.



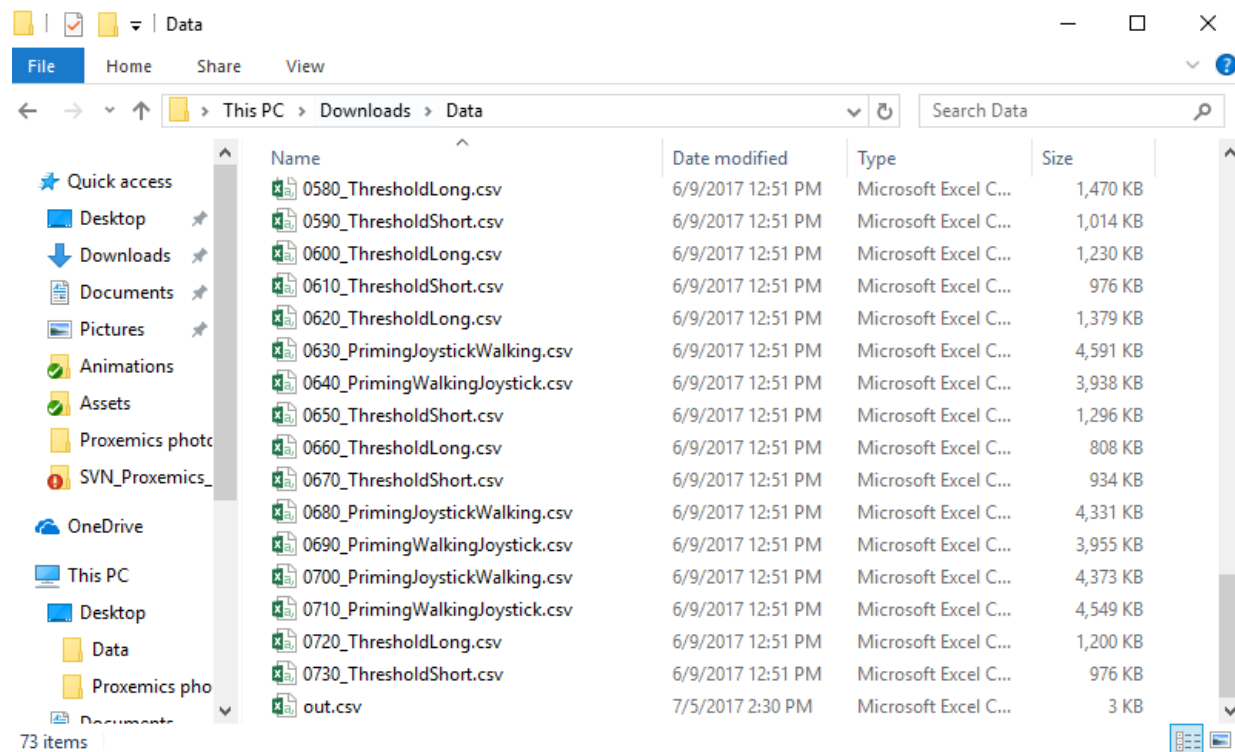
9. To generate a csv for analysis in SPSS, open `vhw_csv.py`. Again, specify the working directory. Be sure to modify the `files` variable to the desired working directory as well.

```
import pandas as pd
import os
from Proxemics import Proxemics

import time
start_time = time.time()

os.chdir("C:\\Users\\tnguyen.ICT2000\\Downloads\\Data")
files = os.listdir("C:\\Users\\tnguyen.ICT2000\\Downloads\\Data")
```

10. The resulting csv will appear in the working directory as `out.csv`.



# API Reference

---

Objects of Proxemics all have the following attributes:

- `self.file` : name of data file as a string (must end with “.csv”)
- `self.df` : a pandas DataFrame object with all proxemics data in the file
- `self.trials` : a tuple of trials listed in the order of the experiment (i.e. (“BeBehindDoor”, “BeBehindDoorJoystick”) and vice versa)

## Processing Data

---

`times_list()`: returns a list of all timestamps

`get_begin_index()`: returns row number in `self.df` that marks when the first trial began

`remove_data_pretrial()`: deletes all data from phases before the first trial (i.e. StandFar, JoyTraining, etc). This function must be called after instantiating a Proxemics object.

`times_by_phase()`: returns a dictionary of time intervals corresponding to each phase of an experiment. The format of the dictionary is: {phase: (phase start time, phase end time)}. The first VHumanWalk phase is denoted VHumanWalk1 and the second VHumanWalk phase is denoted VHumanWalk2.

`relative_zeros_by_phase()`: leverages `time_by_phase()` to make a dictionary of times relative to the start time of the first trial. i.e.: Let's say the first trial starts at 300s and `time_by_phase()` returns {VHumanWalk: (800, 1000)}. `relative_zeros_by_phase()` would return {VHumanWalk1: (500, 700)}.

`trialdf(trial_num)`: `trial_num` takes either a 1 or 2 as input. Returns a DataFrame of the specified trial. Used as a helper function.

`speed_human_trial(trial_num = 1)`: returns a list of lists in the following format: [list of times, list of speeds sampled pair-wise].

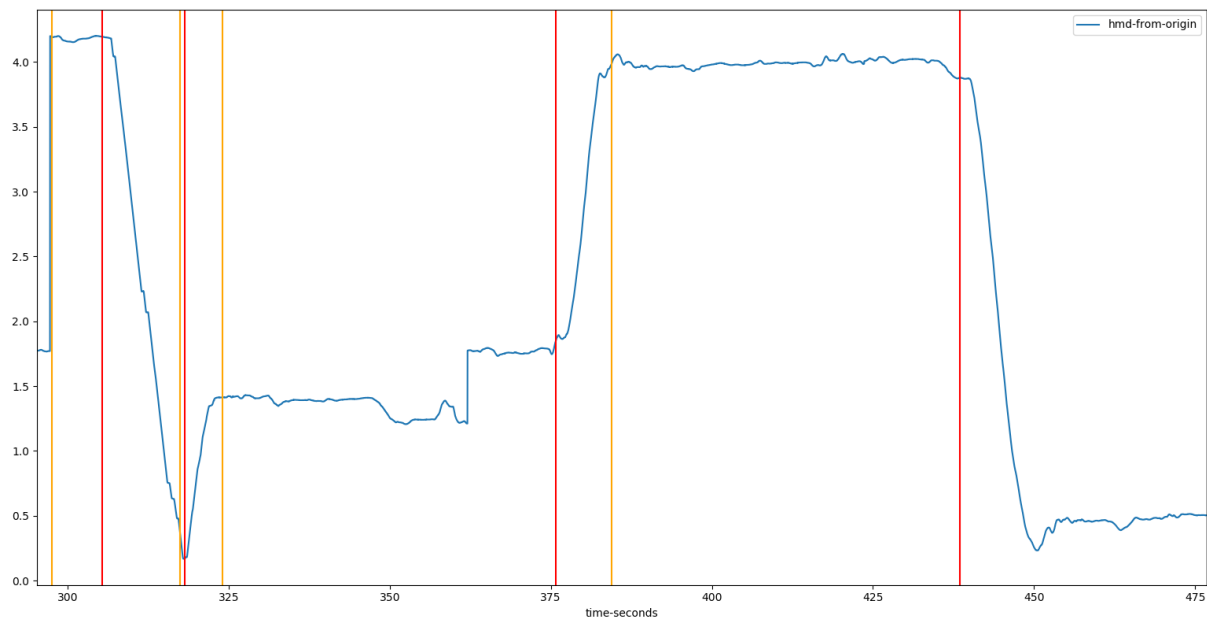
`time_speed_human()`: returns a list of lists in the following format: [list of times, list of speeds sampled pair-wise, list of teleportation times].

`time_window_speed_human(window_size = 350, threshold_speed = 0.04)`: makes use of `time_speed_human()` to sample speed according to given `window_size`. The mean speed across a sliding window is stored in a list. The middle timestamp is stored in a separate list at the same index. Returns a list of lists: [window\_times, mean\_spd\_window]

# Plotting Data

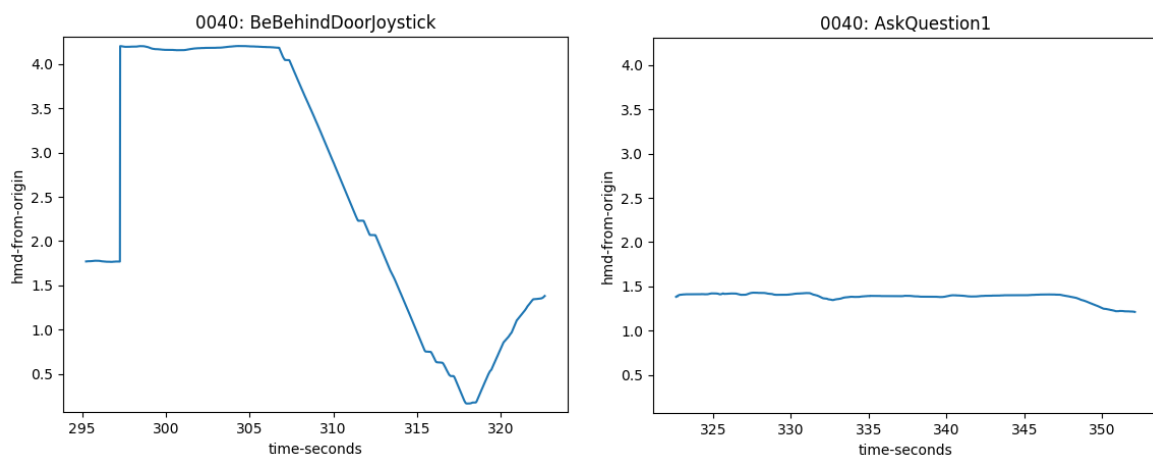
`stand_still(window_size = 350, threshold_speed = 0.04)`: makes use of `time_window_speed_human()` to return a list of times in which subject stood still. i.e.: [(still start time, still end time)]. Plots hmd-from-origin against time and annotates times in which subject stood still. Orange indicates start time, red indicates end time.

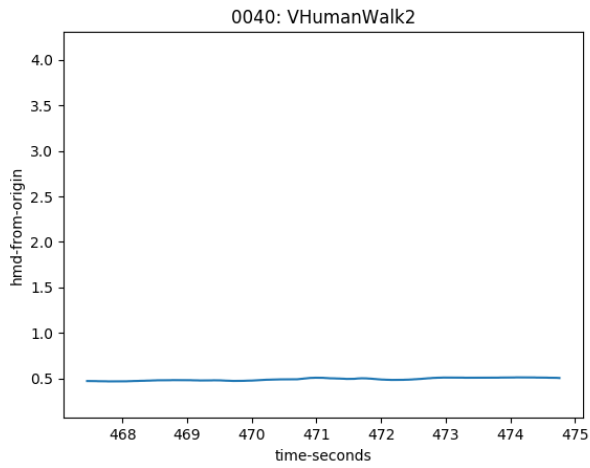
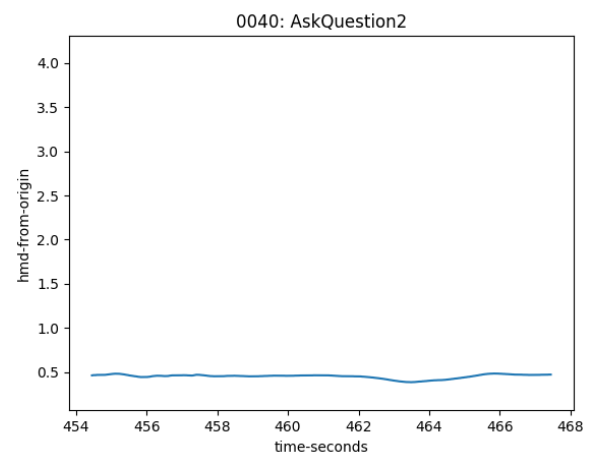
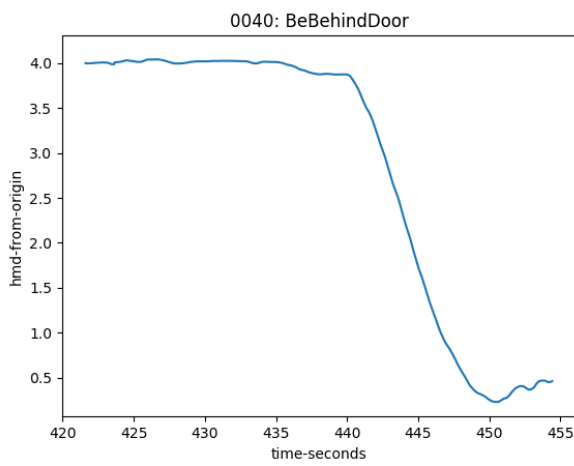
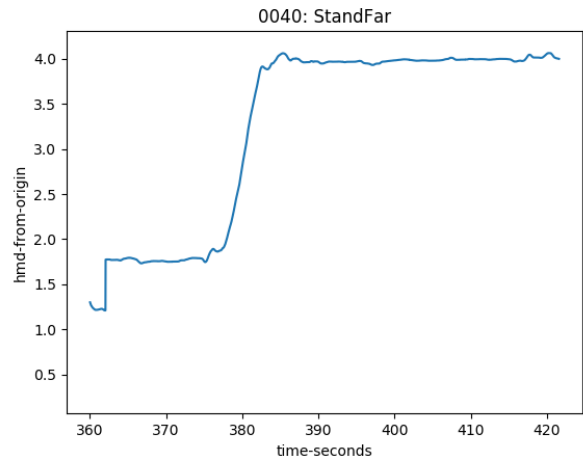
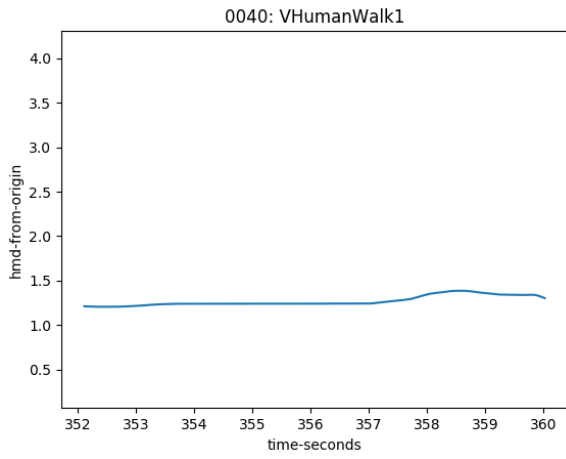
```
prx.stand_still(400, 0.04)
```



`plot_by_phase(phase_input = None)`: plots hmd-from-origin against time for phase specified in `phase_input` as a string. To inspect `VHumanWalk`, trial number must be specified (i.e.: "VHumanWalk1"). By default, all phases will be displayed in separate windows.

```
prx.plot_by_phase()
```

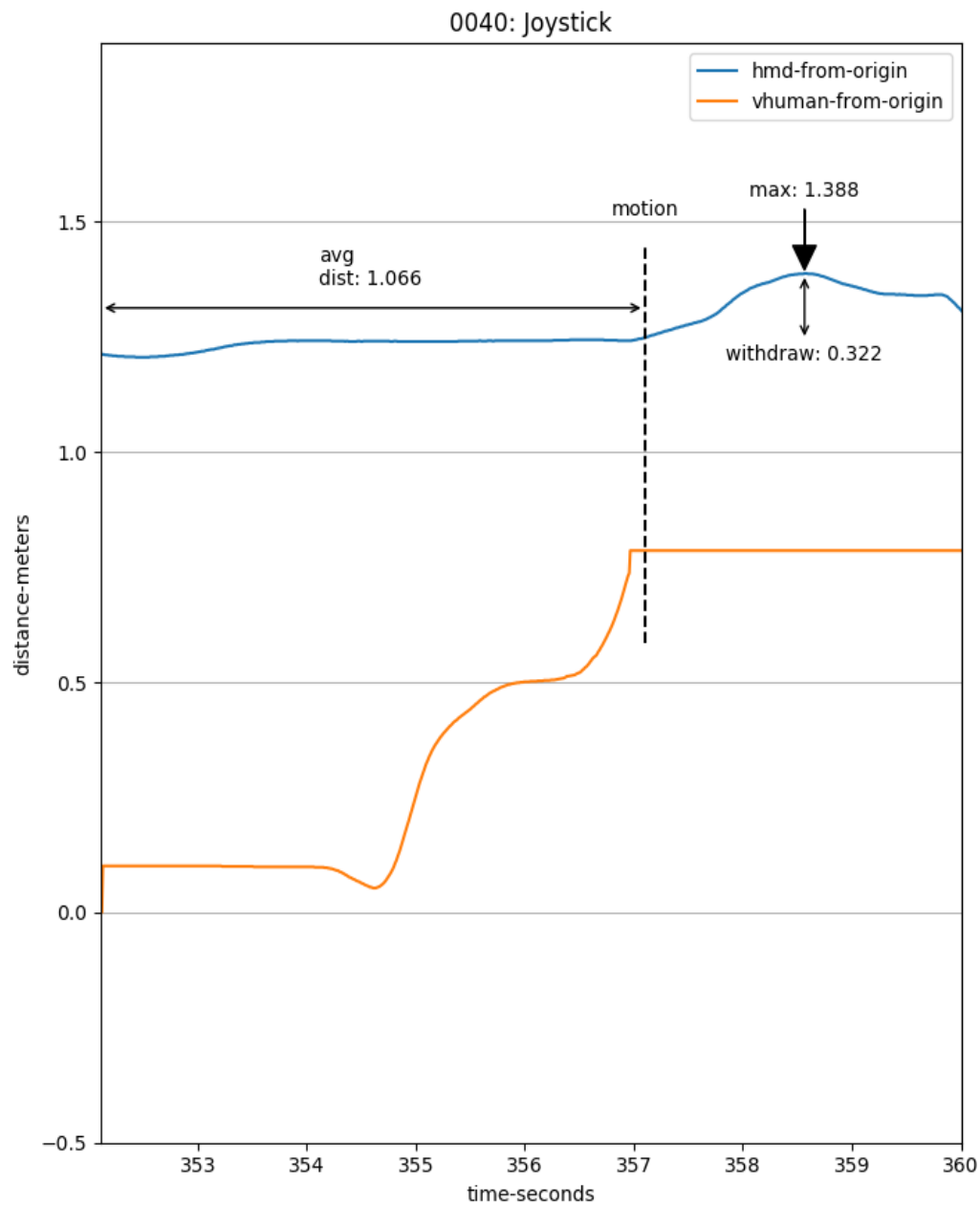


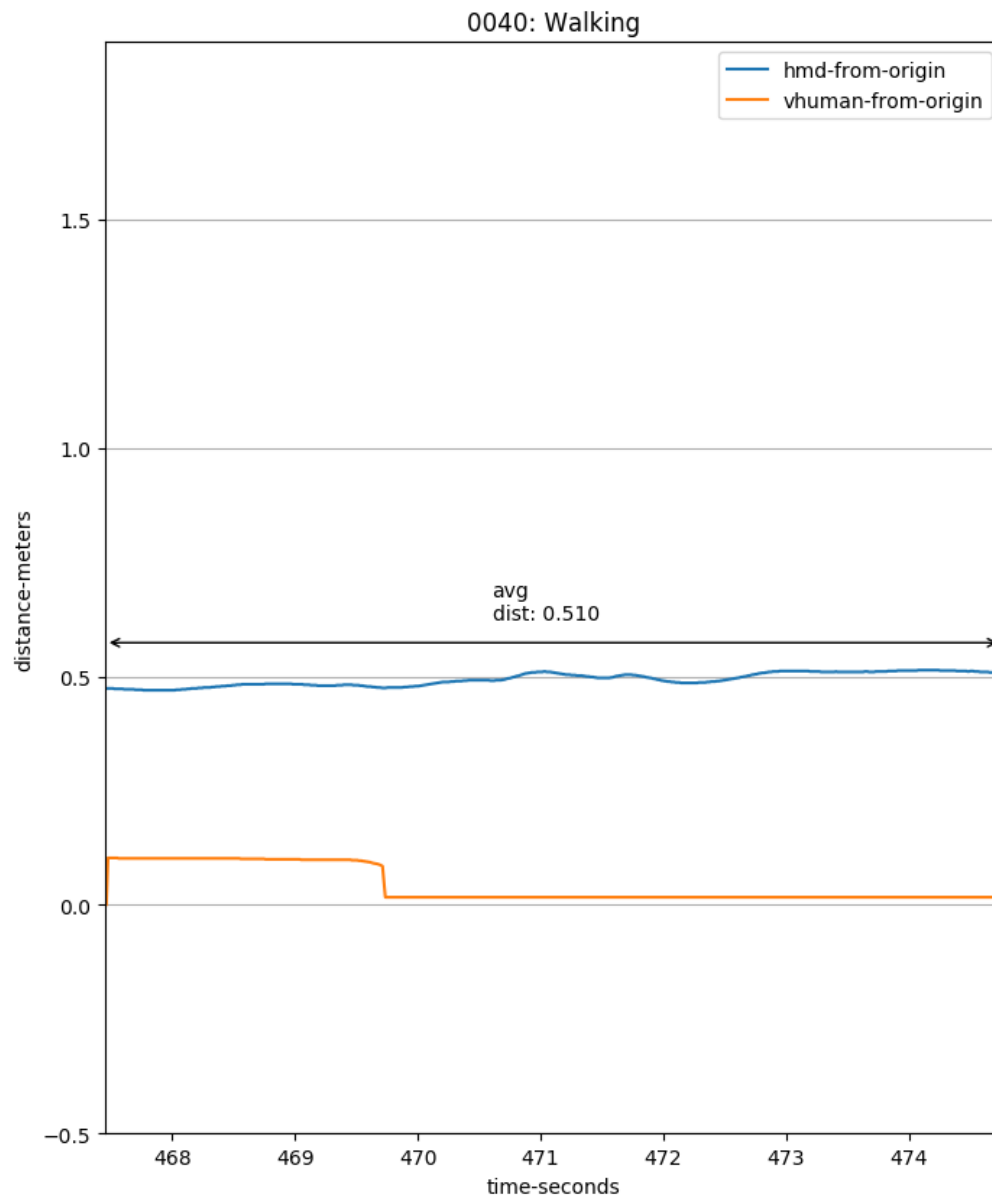


`plot_vhw()`: plots VHumanWalk1 and VHumanWalk2 in separate windows. The x-axis is time-seconds and y-axis is distance-meters (from origin). The plots are annotated with arrows and text indicating average distance maintained between hmd and vhuman. If subject reacted to vhuman walking, the distance subject withdrew is also annotated.

The following example shows VHumanWalk for both trials. Subject does not move in response to vhuman walking in the second trial, so only average distance maintained is annotated.

```
prx = Proxemics("0040_PrimingJoystickWalking.csv")  
prx.remove_data_pretrial()  
prx.plot_vhw()
```

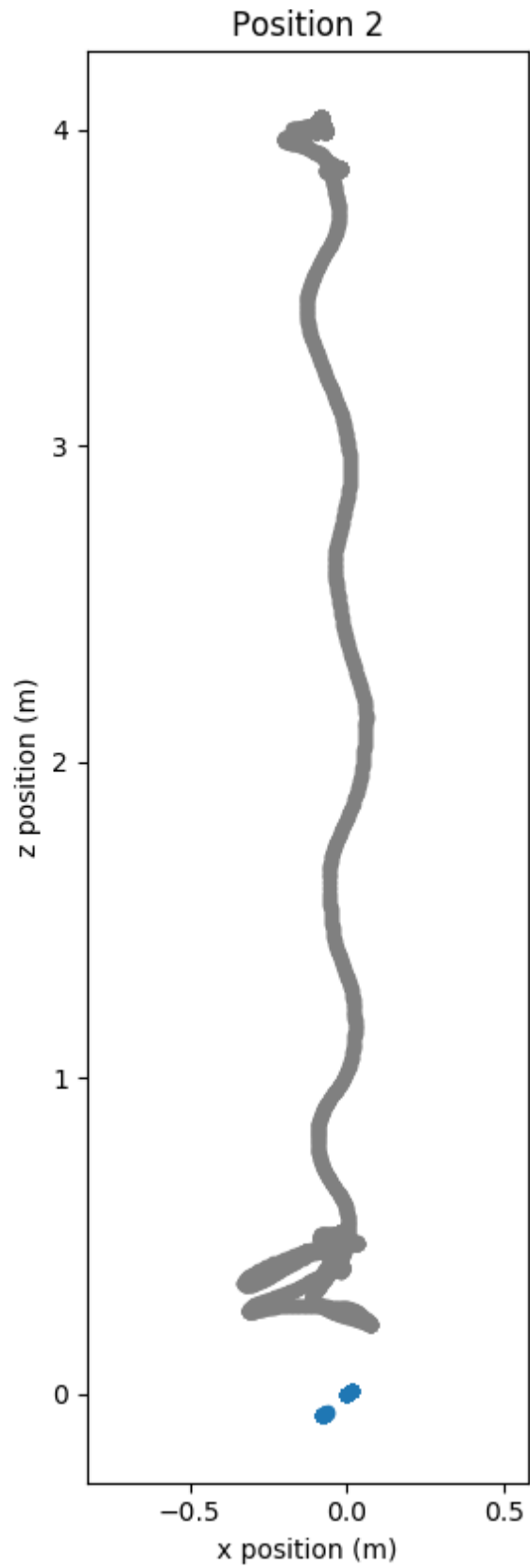
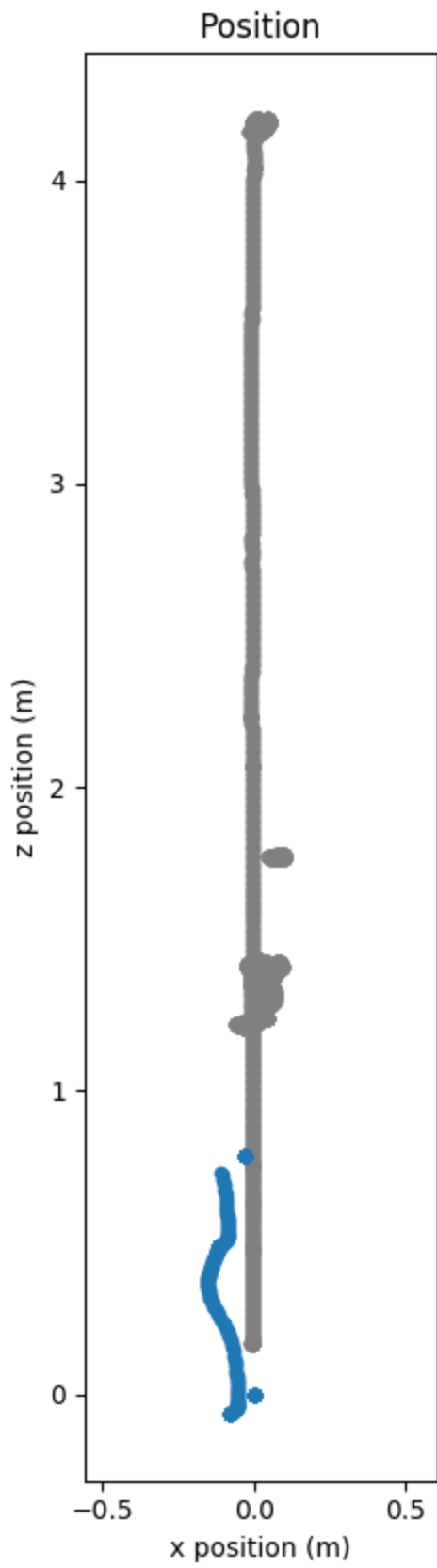




`plot_trials`: plots each trial in a separate window. Hmd-from-origin and vhuman-from-origin are plotted against time for both trials. Uses `trialdf()`.

`hum_vhum_pos(self, trial_num = None)`: plots hmd-from-origin and vhuman-from-origin as scatterplots on the same graph. The x-axis corresponds to x coordinates of both agents and the y-axis corresponds to z coordinates of both agents. If no `trial_num` is specified, both plots will be displayed on separate windows (trial number indicated in titles). Vhuman is shown in blue, subject is shown in grey.

```
prx.hum_vhum_pos()
```



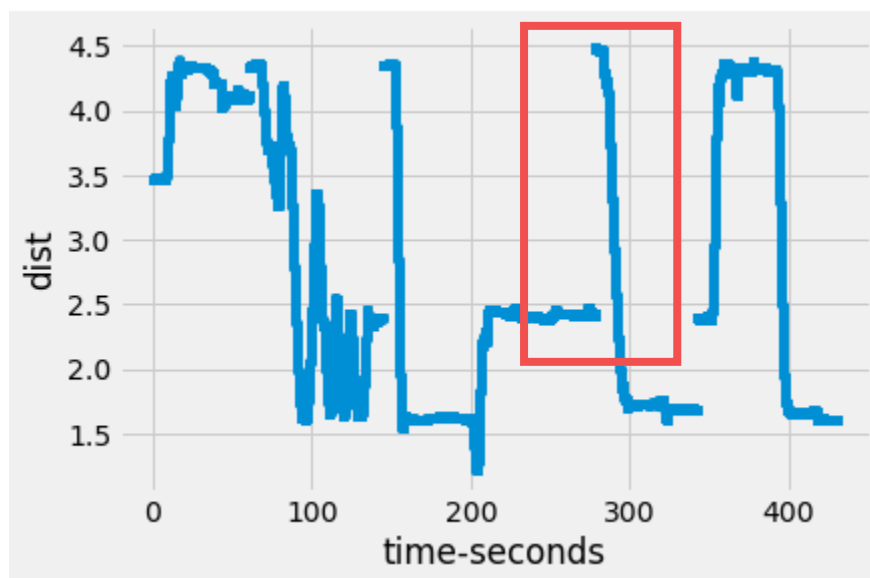


# Helper Functions

## Additional Notes

### Neglecting Teleportation

In the study, a reset was done in between each trial, which instantly 'repositioned' the subject. In analyzing the speed of the user, teleportation had to be filtered out. Below is a plot of distance-meters against time-seconds with teleportation highlighted by the red box. Teleportation was defined as  $\geq 10$  m/s.



### Calculating Speed and Stillness

Speed was first calculated pair-wise, using hmd-from-origin and time-seconds data. Let's pretend the following table belongs in the Proxemics data set.

time-seconds	hmd-from-origin
1	2
2	1.5
3	1
4	0.5
5	0.25
6	0.2

Pair-wise speed calculations would yield the following result:

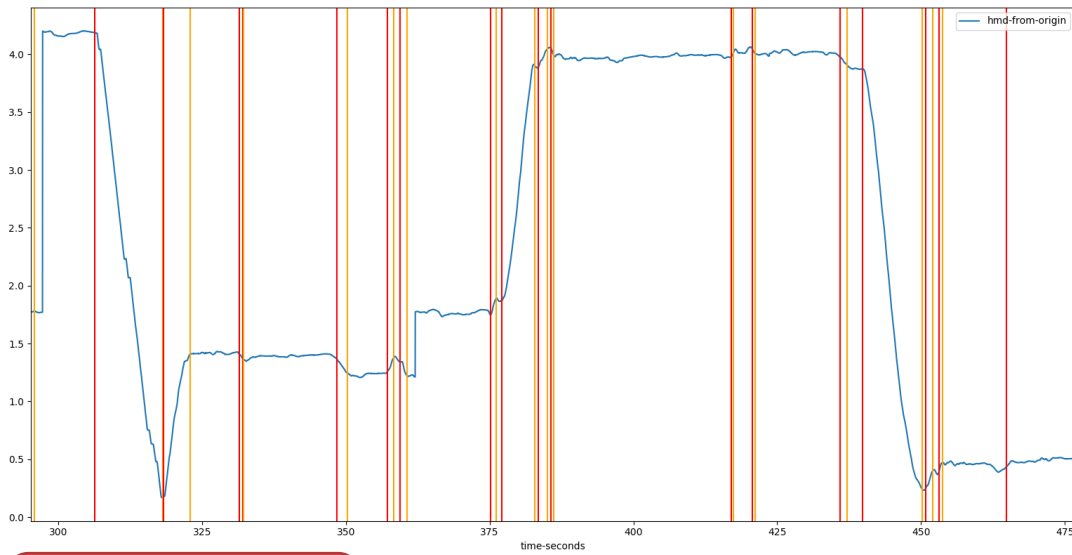
$$speed = \frac{\Delta hmd-from-origin}{\Delta time-seconds}$$

time-seconds	hmd-from-origin	pairwise speed
1	2	
2	1.5	-0.5 m/s
3	1.3	-0.2 m/s
4	1.2	-0.1 m/s
5	0.25	-0.95 m/s
6	0.2	-0.05 m/s

Since tracking data was recorded at 90 Hz, the pairwise speed calculations were approximately 0 m/s, but not exactly zero. To detect stillness, a sliding average (window) was used to smooth out the speed calculations. The `window_size` parameter in Proxemics corresponds to the number of pair-wise speeds to be averaged. Using the previous data set with a window size of 3 results in the following “window-speed” calculations:

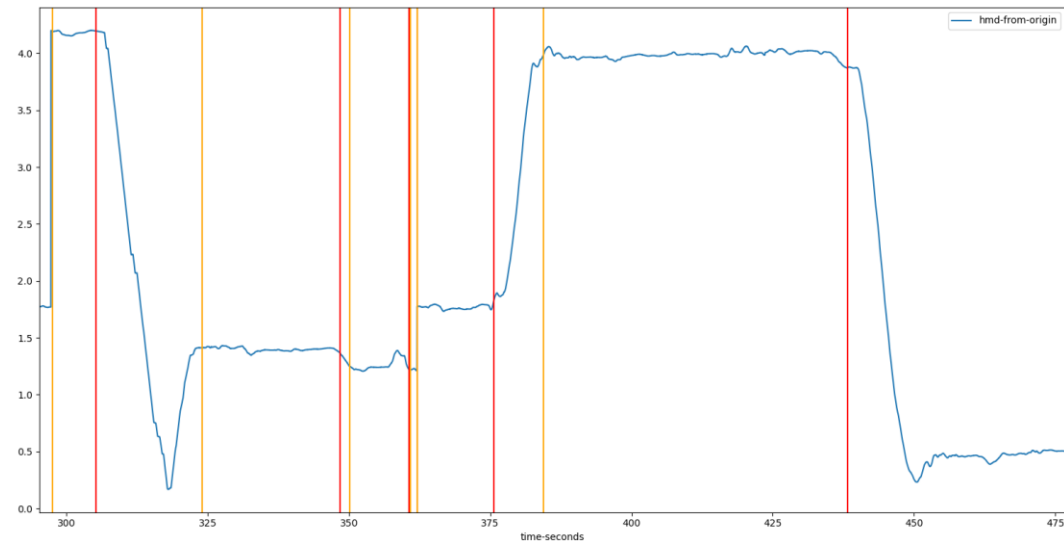
time-seconds	hmd-from-origin	pairwise speed	window-speed(3)
1	2	0 m/s	
2	1.5	-0.5 m/s	-0.23 m/s
3	1.3	-0.2 m/s	-0.27 m/s
4	1.2	-0.1 m/s	-0.625 m/s
5	0.25	-0.95 m/s	-0.37 m/s
6	0.2	-0.05 m/s	

Fine-tuning this window-speed calculation involved examining several data sets and adjusting the `window_size` and `threshold_speed` parameters. Through a lot of trial and error, the best representation of speed resulted when a `window_size` of 350 – 400 was used with a `threshold_speed` of 0.04 m/s. The example below depicts `stand_still(100, 0.05)`



window size = 100,  
threshold speed = 0.05

Setting the window size to 400 and threshold speed to 0.04 m/s results in a much cleaner and more accurate depiction of user stillness.



window size = 400,  
threshold speed = 0.04

## Detecting Motion

Understanding how to represent stillness means that user motion can be detected by the computer (using a `window_size` of 350 - 400) as:

- window speeds that exceed 0.04 m/s
- less than 10 m/s

The default `window_size` in Proxemics is 350.

## Acknowledgements

Special thanks to:

- Dr. Krum
- Thai Pham
- Rhys Yahata
- Dr. Rosenberg
- Ryan Spicer