Eye_tracking

May 3, 2017

1 Eye tracking - Quantitative analysis

This part focuses on quantitative analysis on eye tracking data.

1.1 Import and global variables

```
In [1]: import codecs                              # to read '.txt' files files
        import csv                                 # to read and write '.csv' files
        import matplotlib.pyplot as plt            # to plot graphs
        import numpy as np
        import os                                  # to deal with os calls
        import pandas as pd
        import seaborn as sns
        import collections
        from scipy.signal import argrelextrema, argrelmax
        from pylab import savefig
```

```
In [2]: participant = 0              # current participant ID
        recording_name = 1           # name of the recording
        recording_duration = 2       # recording duration
        time_column = 3              # time indication
        gaze_x_column = 4            # x-position of the gaze point
        gaze_y_column = 5            # y-position of the gaze point
        pupil_diam_left = 6          # diameter of left pupil over time
        pupil_diam_right = 7         # diameter of right pupil over time
        mt_column = 8                # movement type column
        md_column = 9                # movement duration column
        mi_column = 10               # movement index column
        event_column = 11            # Event type
```

1.2 Helper functions

```
In [3]: # Read UTF-16 encoded unicode '.txt'. Usefull for cross-platform encoding.
        def load(location):
            data = []
            f=codecs.open(location,"rb","utf-16")
            csvread=csv.reader(f,delimiter='\t')
```

```
        csvread.next()
        for row in csvread:
            data.append(row)

        # Filtering the data where Eye Tracking is not working
        return [line for line in data if not line[mt_column] == "EyesNotFound"]

    # Returns True if string can be converted as float
    def is_number(str):
        try:
            float(str)
            return True
        except:
            return False

    # Returns list of users contained in data
    def get_users(data):
        users = list(set([l[participant] for l in data[1:]]))
        users.sort()
        return users
```

1.3 Analysis of means and standard deviations of fixations

1.3.1 Main functions

```
In [4]: # Defining functions for titles in quantitative analysis

        def fixation_key(user):
            return "all " + str(float(user)) + ' - Fixation time (ms)'

        def saccade_key(user):
            return "all " + str(float(user)) + ' - Saccade time (ms)'

        def gaze_x_key(user):
            return "all " + str(float(user)) + ' - Gaze X (pixels)'

        def gaze_y_key(user):
            return "all " + str(float(user)) + ' - Gaze Y (pixels)'

        keys = {'fixation': fixation_key, 'saccade': saccade_key, 'gaze_x': gaze_x_key, 'gaze_y'
```

```
In [5]: def quantitative(raw_data, user_id, type):
            raw_data = raw_data[1:]

            def analyse_records(data, user_id):
                # Fixation and saccade times
                pd1 = pd.DataFrame(list(set([float(line[md_column]) for line in data if line[mt_
                                   columns=[str(user_id) + ' - Fixation time (ms)']).describe(in
                pd2 = pd.DataFrame(list(set([float(line[md_column]) for line in data if line[mt_
```

```
                        columns=[str(user_id) + ' - Saccade time (ms)']).describe(inc

            # X and Y coordinates of gaze points
            pd3 = pd.DataFrame([float(line[gaze_x_column]) for line in data if is_number(lin
                        columns=[str(user_id) + ' - Gaze X (pixels)']).describe(inclu
            pd4 = pd.DataFrame([float(line[gaze_y_column]) for line in data if is_number(lin
                        columns=[str(user_id) + ' - Gaze Y (pixels)']).describe(inclu

            return pd.concat([pd1, pd2, pd3, pd4], axis=1)

        if type == "all":
            data = [line for line in raw_data if float(line[participant]) == user_id]
            return analyse_records(data, "all " + str(user_id))

        if type == "each":
            records = list(set([line[recording_name] for line in raw_data if float(line[part
            records.sort()
            dfs = pd.DataFrame()
            for rec in records:
                data = [line for line in raw_data if line[recording_name] == rec]
                dfs = pd.concat([dfs, analyse_records(data, str(user_id) + " - " + str(rec))
            return dfs
```

```
In [10]: def describe_quantitative(d, users, save=True):
             plt.clf()

             # Showing: fixation, saccade, gaze X, gaze Y on average
             f, axarr = plt.subplots(4, sharex=False)
             users_int = [int(user) for user in users]
             axarr[0].set_xlim([min(users_int)-1, max(users_int)+1])
             axarr[0].set_title("Average fixation time")
             axarr[0].set_ylabel("time (ms)")
             axarr[1].set_title("Average saccade time")
             axarr[1].set_ylabel("time (ms)")
             axarr[2].set_title("Average Gaze X position")
             axarr[2].set_ylim([0, 1920])
             axarr[2].set_ylabel("position (pixel)")
             axarr[3].set_title("Average Gaze Y position (not reversed)")
             axarr[3].set_ylim([0, 1080])
             axarr[3].set_ylabel("position (pixel)")

             def plot_describe(data_frame, pos):
                 data_array = data_frame.as_matrix()
                 for index in range(0, len(data_array[0])):
                     axarr[pos].errorbar(int(users[index]), data_array[1][index], data_array[2][
                                         color='red', lw=3)
                     axarr[pos].errorbar(int(users[index]), data_array[1][index],
                                 [[data_array[1][index] - data_array[3][index]], [data_array[7]
```

```
                             fmt='.k', color='gray', lw=1)

        def plot_one_value(value, pos):
            data_frame = pd.DataFrame()
            for user in users:
                result = pd.concat([quantitative(d, float(user), "all"), quantitative(d, fl

                # Saving the results into CSV
                if save is True:
                    # base = os.path.dirname(os.path.abspath(__file__))
                    # location = base + "\\..\\results\\" + str(user) + "_results.csv"

                    location = str(user) + "_results.csv"
                    result.to_csv(location, sep=';', encoding='utf-16')

                key = keys[value](user)
                data_frame = pd.concat([data_frame, pd.DataFrame(result, columns=[key])], a
            plot_describe(data_frame, pos)

            return data_frame

        plot_one_value('fixation', 0)
        plot_one_value('saccade', 1)
        plot_one_value('gaze_x', 2)
        plot_one_value('gaze_y', 3)
        plt.tight_layout()
        plt.xlabel('User ID')

        f.set_size_inches(20, 18, forward=True)

        # Save pictures
        savefig('foo.png', bbox_inches='tight')
        savefig('foo.pdf', bbox_inches='tight')

        # Show the curves
        plt.show()
```

1.3.2 Results - application on data

This section aims at reading and performing the analysis of the data captured by the glasses.

Data are exported from the glasses as '.tsv' file and converted using Excel as 'unicode .txt' file in order to enable a cross-platform encoding. The data of all the users (containing all together 2,155 045,rows) are exported in 3 different files due to the size limitation of Excel which cannot handle more than 1,048,576 rows.

After loading the data, some IDs are deleted manually from the list of users to consider since their related participant didn't fulfill all the experiment requirements.

```
In [7]: # Reading the eye tracking data from 3 different files
```

```
if False:
    tmp = load("/media/sf_EyeTracking/data/short_1_35_unicode.txt")
    tmp.extend(load("/media/sf_EyeTracking/data/short_36_51_unicode.txt"))
    tmp.extend(load("/media/sf_EyeTracking/data/short_52_71_unicode.txt"))

print "Data contains " + str(len(tmp)) + " rows."
```

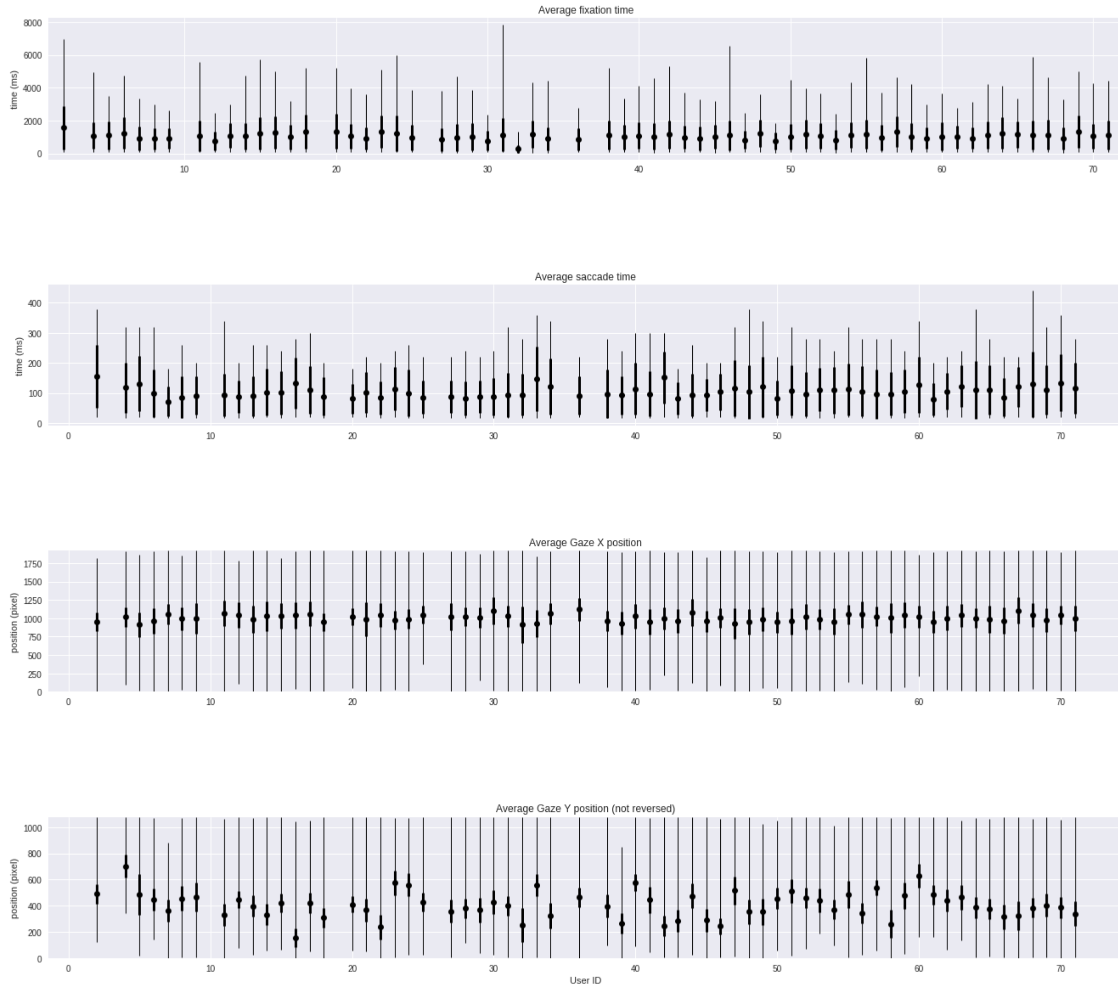
```
Data contains 2155045 rows.
```

```
In [8]: # Getting the list of users
        users_str = get_users(tmp)
        users_float = [int(user) for user in users_str]
        users_float.sort()
        print "Users IDs from the data: " + str(users_float)

        # Some user ID are known as being missing.
        print "Missing IDs in the data: " + str([x for x in range(1,71) if x not in users_float]

        # Some other users are not considered.
        not_users = [35, 37]
        print "IDs not considered: " + str(not_users) +'\n'
        users_float = [user for user in users_float if user not in not_users]
        users_str = [str(user) for user in users_float]

        # List of users for the experiment
        print str(len(users_float)) + " participants to analyze: " + str(users_float)
```

```
Users IDs from the data: [2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 2
Missing IDs in the data: [1, 3, 10, 19, 26]

IDs not considered: [35, 37]

64 participants to analyze: [2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23
```

```
In [11]: # Perform the analysis on the list of users ID selected.
         describe_quantitative(tmp, users_str, save=False)      # If True, csv files with data a
```

```
<matplotlib.figure.Figure at 0x7f91dc0d2750>
```




```
In [12]: # Eventually, the generated files can be merged together as an '.xls' file using this s
         if True:
             import glob2, xlwt

             wb = xlwt.Workbook(encoding='utf-8')
             for filename in glob2.glob("/media/sf_EyeTracking/results/utf-8/*.csv"):
                 (f_path, f_name) = os.path.split(filename)
                 (f_short_name, f_extension) = os.path.splitext(f_name)
                 ws = wb.add_sheet(f_short_name)
                 spamReader = csv.reader(open(filename, 'rb'), delimiter=";")
                 for rowx, row in enumerate(spamReader):
                     for colx, value in enumerate(row):
                         ws.write(rowx, colx, value)
             wb.save("/media/sf_EyeTracking/results/utf-8/compiled.xls")
```

```
In [13]: # TODO print the standard deviation for each user per pixels on each axis
         # Then apply clustering technique
```

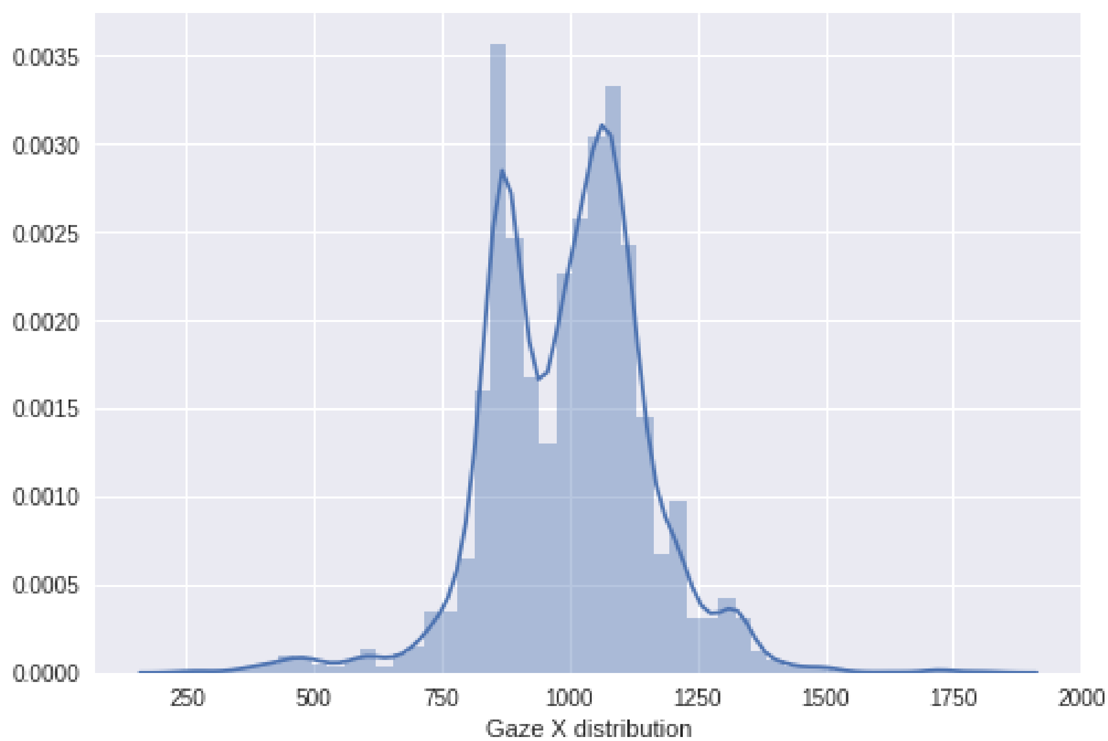
1.4 Analysing gaze points over X and Y coordinates

1.4.1 X coordinates: working over value distribution

In [14]:
```
# Selecting the values for one user and one recording.
x_pos = [line[gaze_x_column] for line in tmp
            if line[participant] == '42'
            and line[recording_name] == 'Recording172']
x_pos = [int(line) for line in x_pos if line != '']
x_pos = np.asarray(x_pos)

print "List of x values: " + str(x_pos)
```

```
List of x values: [ 979  981  982 ..., 1276 1278 1280]
```

In [15]:
```
# Plotting the distribution based on SeaBorn
x_pos = pd.Series(x_pos, name="Gaze X distribution")
x_dist = sns.distplot(x_pos)
plt.show()
```

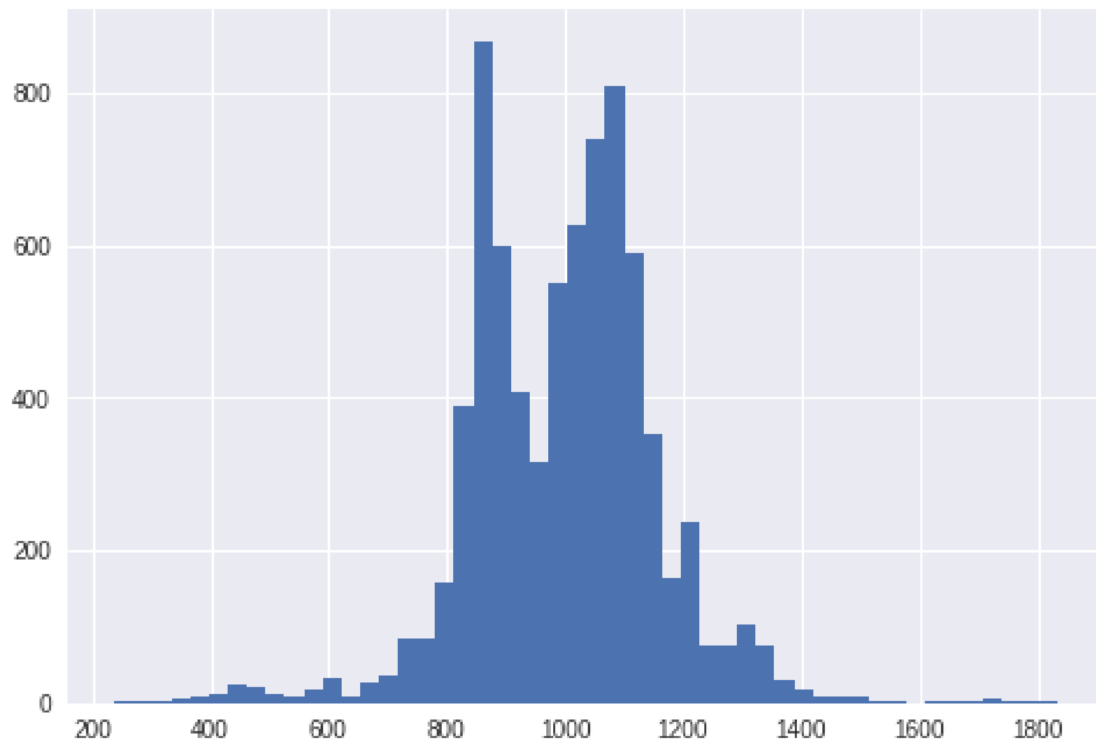


This represents where people look at when driving. They mostly look at their lane, but also check the opposite one, which is more on the left.

It is then interesting to have a look at the distance between the two peaks (at first counted here, in number of bins - arbitrary value, stable over measurement). The ratio between the first and second extrema is then computed.

Some curves might create other extremum that are not related to the driving lanes, this can then be corrected manually by choosing the right extremum.

```
In [16]: # Reconstructing the values based on NumPy
         a = x_pos.hist(bins=50)
         plt.show()
```




```
In [17]: def get_hist(ax):
             n, bins = [],[]
             for rect in ax.patches:
                 ((x0, y0), (x1, y1)) = rect.get_bbox().get_points()
                 n.append(y1-y0)
                 bins.append(x0) # left edge of each bin
             bins.append(x1) # also get right edge of last bin

             return n,bins

         n, bins = get_hist(a)
         n = np.asarray(n)

         def bin_length(bins):
             bin_len = []
             for i in range(0,len(bins)-1):
```

```
            bin_len.append(bins[i + 1] - bins[i])
        return bin_len

    bin_len = bin_length(bins)
    bin_length = bin_len[0]
```

```
In [18]: n_values = n[argrelmax(n)[0]]
         print "Local optimums values: " + str(n_values)

         # Return maximum
         fst_max = np.partition(n_values.flatten(), -1)[-1]
         fst_index = np.argmax(n)
         print "First extremum: " + str(fst_index) + ", " + str(fst_max)

         # Return second value
         scd_max = np.partition(n_values.flatten(), -2)[-2]
         scd_index = np.nonzero(n==scd_max)[0][0]
         print "Second extremum: " + str(scd_index) + ", " + str(scd_max)

         diff_max = fst_max - scd_max
         diff_index = fst_index - scd_index
         print "Difference between optimums positions: " + str(diff_index) + " bins"
         print "Difference between optimums position: " + str(diff_index * bin_length) + " pxls"
         print "Difference between optimums value: " + str(diff_max) + " pxls"
         print "Ratio between optimums: " + str(fst_max/scd_max)
```

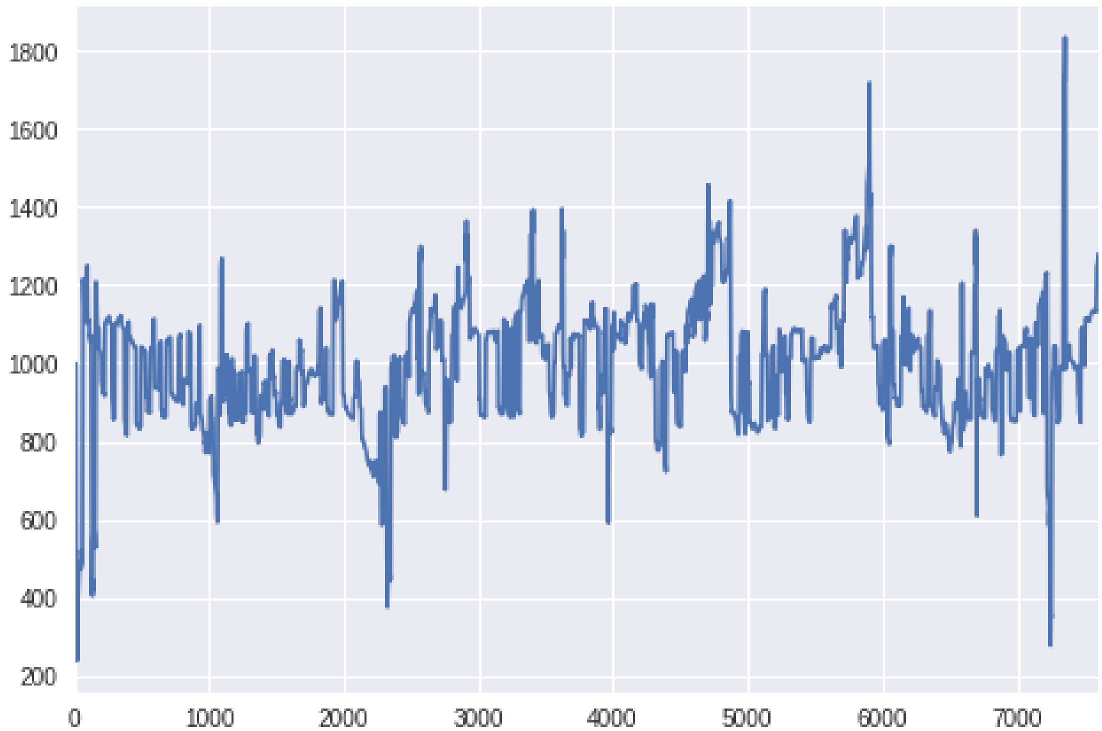
```
Local optimums values: [   3.    25.    34.   866.   808.   236.   104.    10.     6.]
First extremum: 19, 866.0
Second extremum: 26, 808.0
Difference between optimums positions: -7 bins
Difference between optimums position: -223.44 pxls
Difference between optimums value: 58.0 pxls
Ratio between optimums: 1.07178217822
```

Results on participant 42: - recording 169: 5, 456.0 (tr2) - recording 170: 5, 296.0 (tr3) - recording 171: 7, 573.0 (tr1) - recording 172: -7, 58.0 1.071 (tr2)

Results on participant 43: - recording 173: 5, 419.0 1.570 (tr3) - recording 175: -2, 149.0, 1.198 (tr1) - recording 176: -4, 262.0, 1.287 (tr2) - recording 177: 5, 262.0, 1.420 (tr3)

1.4.2 X coordinates: working over time distribution

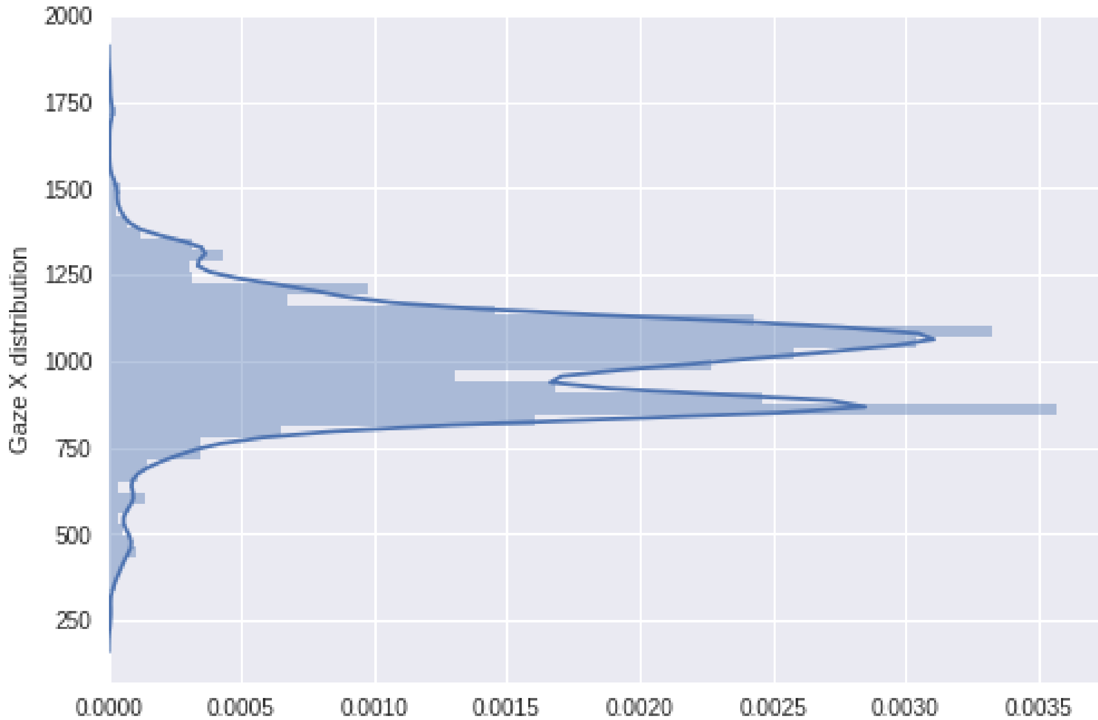
```
In [19]: # Printing with SeaBorn
         sns.tsplot(x_pos)
         plt.show()
```



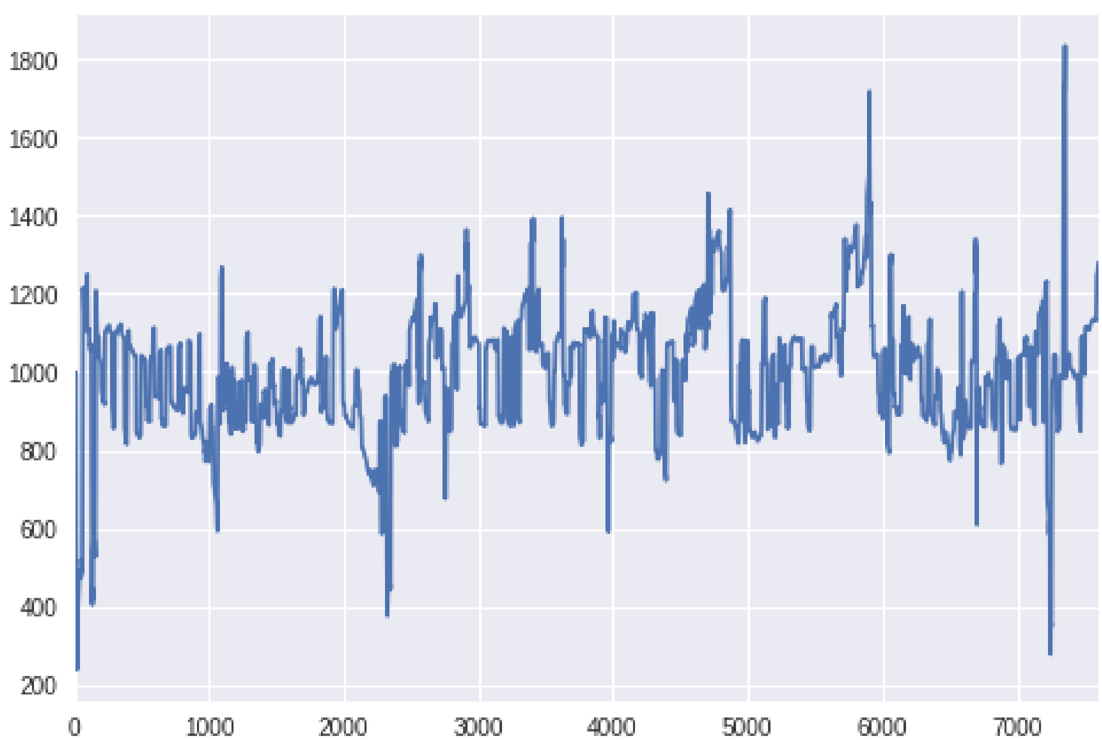
First one might represent looking in the side mirror when starting. Other ones might also represent crossing or checking on the sides.

1.4.3 Y coordinates: working over value distribution

```
In [20]: # Getting the data for one user and one record
         y_pos = [line[gaze_y_column] for line in tmp
                  if line[participant] == '42'
                  and line[recording_name] == 'Recording169']
         y_pos = [int(line) for line in y_pos if line != '']

         # Reversing the data
         y_pos = [1080 - line for line in y_pos]
         y_pos = np.asarray(y_pos)
         y_pos

Out[20]: array([807, 808, 809, ..., 906, 702, 474])

In [102]: recordings = list(set([line[recording_name] for line in tmp]))

In [26]: y_pos = pd.Series(x_pos, name="Gaze X distribution")
         sns.distplot(y_pos, vertical=True)
         # Plotting the distribution based on SeaBorn
         plt.show()
```




```
In [27]: sns.tsplot(y_pos)
         plt.show()
```



Peaks show the number of time the driver is checking the upper mirror. Down shows the checking of commands (biggest ones) or of the speed indication (less big ones).

1.5 Working with heatmap around events

```
In [28]: import matplotlib.cm as cm
         import matplotlib.mlab as ml
         from scipy.stats import kendalltau
```

```
In [29]: def get_time(usr, rec):
             coord = [int(line[time_column]) for line in tmp
                      if line[participant] == usr and line[recording_name] == rec]
             return min(coord), max(coord)
```

```
In [30]: def ms2mins(millis):
             millis = int(millis)
             seconds=(millis/1000)%60
             seconds = int(seconds)
             minutes=(millis/(1000*60))%60
             minutes = int(minutes)
             return ("%dm%ds" % (minutes, seconds))

         def mins2ms(minutes, seconds, millis=0):
             return minutes * 60 * 1000 + seconds * 1000 + millis
```

```
In [31]: print ms2mins(mins2ms(2,32,0))
```

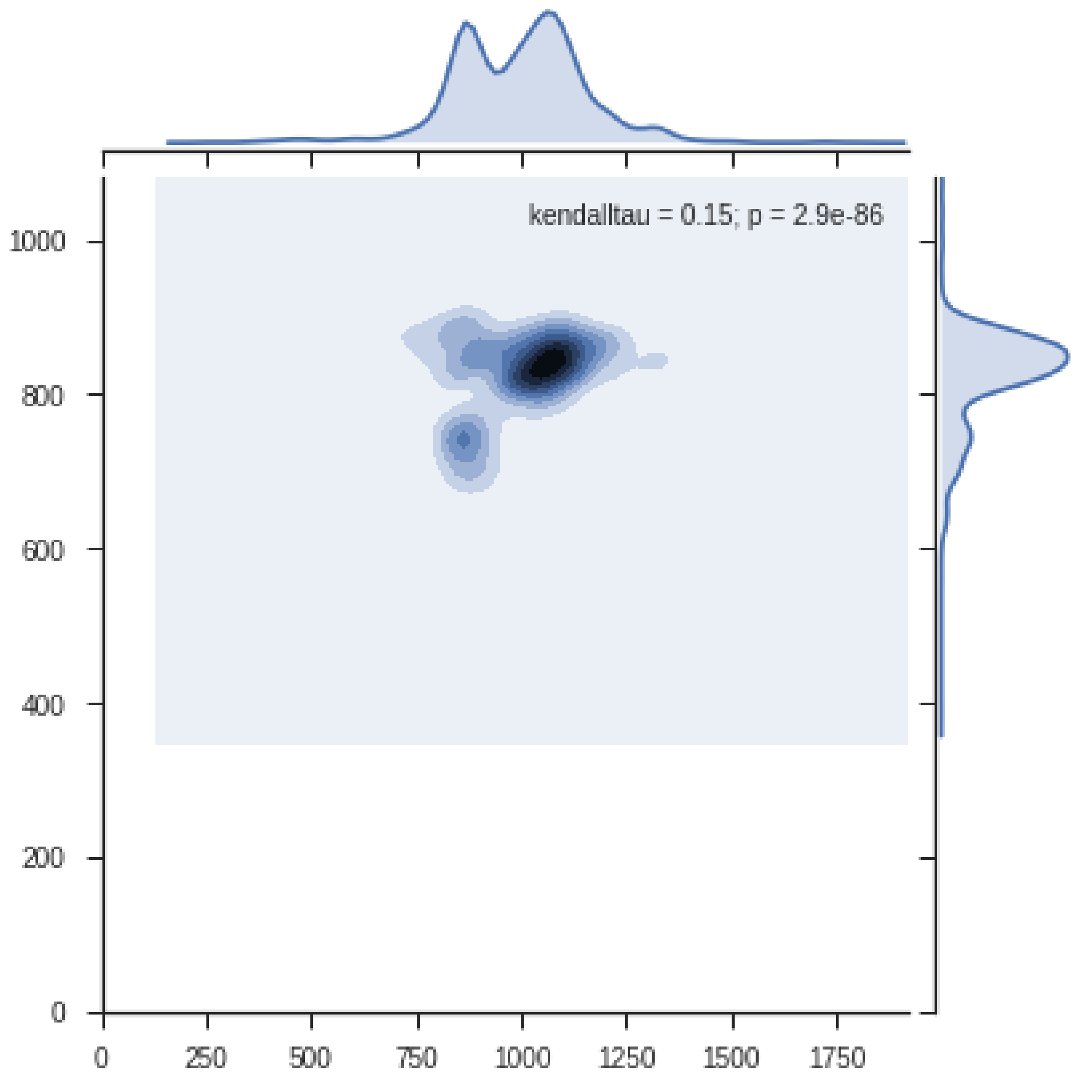
```
2m32s
```

```
In [32]: def heatmap(usr, rec, start=-1, end=-1):

             if start == -1 and end == -1:
                 start, end = get_time(usr, rec)

             # Selecting the values for one user and one recording.
             coord = [[line[gaze_x_column], line[gaze_y_column]] for line in tmp
                      if line[participant] == usr and line[recording_name] == rec
                      and int(line[time_column]) < end
                      and int(line[time_column]) > start]

             # Converting to integers
             def toInt(l):
                 return [int(i) for i in l if is_number(i)]
             coord = [toInt(l) for l in coord if len(toInt(l)) == 2]

             # Extracting the data
```

```
        x = [l[0] for l in coord]
        y = [l[1] for l in coord]
        x = np.asarray(x).astype(np.int)
        y = np.asarray(y).astype(np.int)

        # Printing the plot
        print "Plot from " + ms2mins(start) + " to " + ms2mins(end)
        sns.plt.clf()
        sns.set(style="ticks")
        sns.jointplot(x, 1080-y, stat_func=kendalltau, kind="kde",
                      xlim=(0,1920), ylim=(0,1080))
        sns.plt.show()
```

```
In [33]: heatmap(usr='42', rec='Recording172')
```

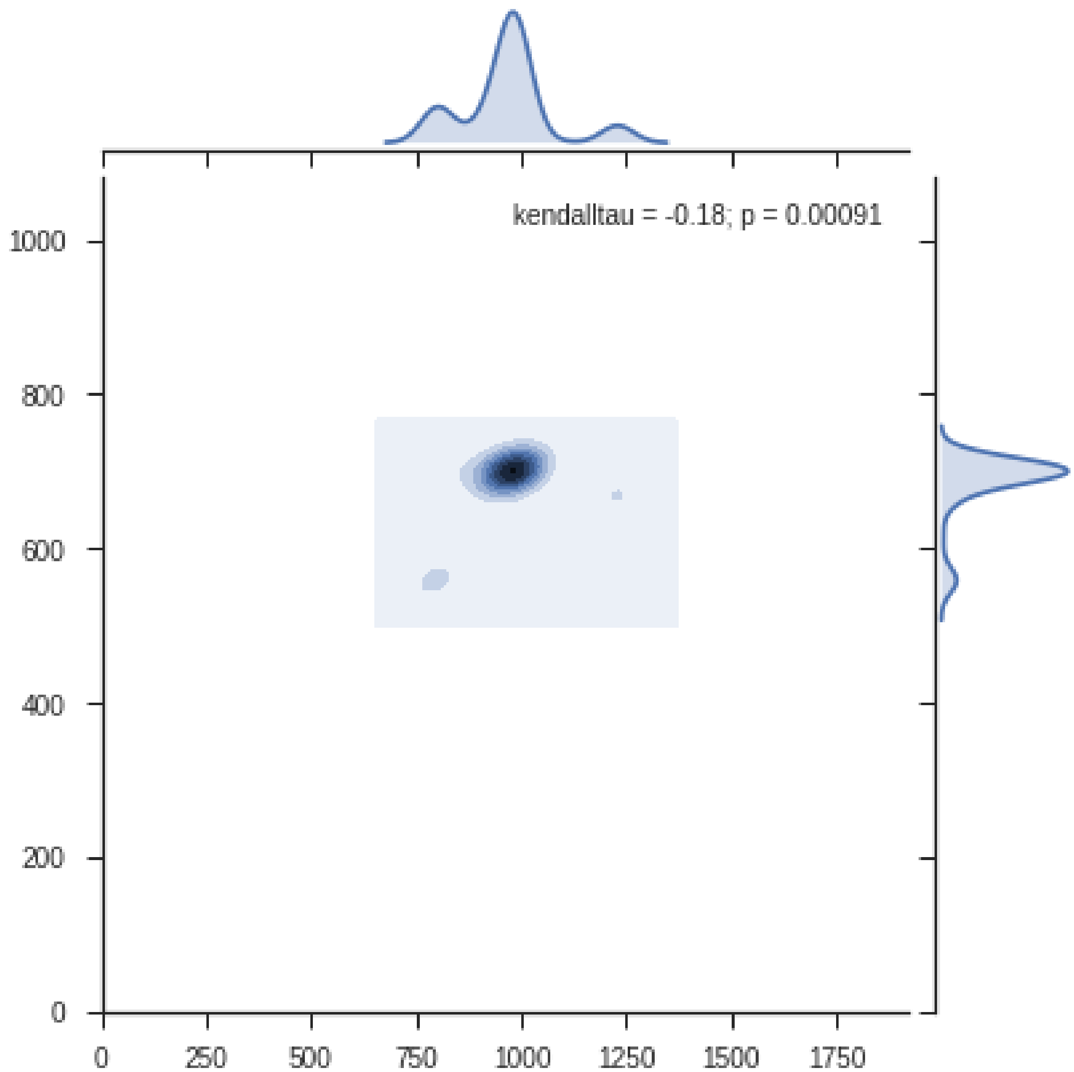
```
Plot from 0m0s to 2m32s


<matplotlib.figure.Figure at 0x7f91d60f75d0>
```




```
In [34]: # Example for one specific hazardous situation
         heatmap('38','Recording154',start=mins2ms(0,30,285),end=mins2ms(0,33,466))
```

```
Plot from 0m30s to 0m33s
```

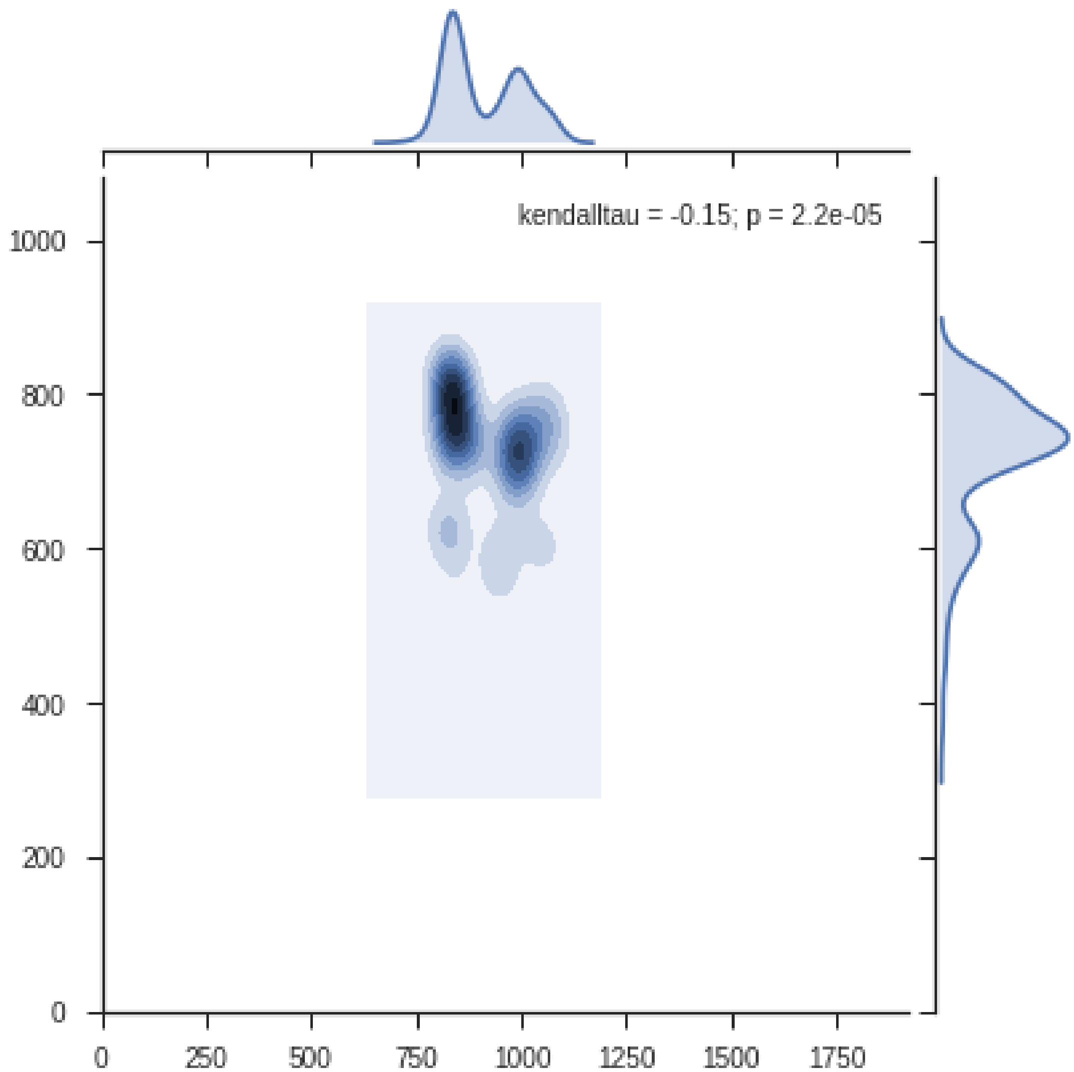
```
<matplotlib.figure.Figure at 0x7f91d6159d90>
```




```
In [35]: # Example for one specific hazardous situation
         heatmap('38','Recording154',start=mins2ms(2,47,99),end=mins2ms(2,54,580))
```

```
Plot from 2m47s to 2m54s
```

```
<matplotlib.figure.Figure at 0x7f91d5d69190>
```

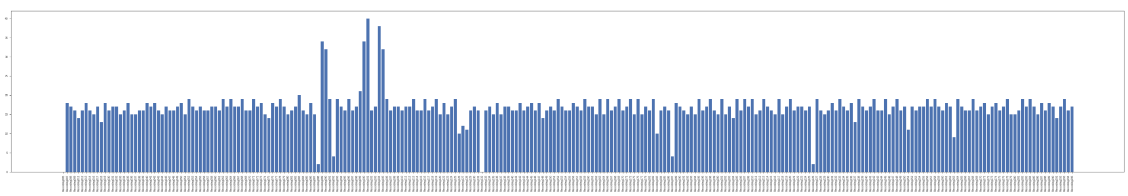


1.6 Checking for missing events in recordings

```
In [94]: # Get set of event names
         events_names = [ l[event_column] for l in tmp]
         events_names = list(set(events_names))

         # Interesting events
         log_event = 'Logged live Event'
         del_event = 'delete event'
         mis_event = 'Missing event'

         # Get number of interesting events
         def events_nb(data):
```

```
            count = 0
            for l in data:
                if l[event_column] == log_event or l[event_column] == mis_event:
                    count += 1
                if l[event_column] == del_event:
                    count -= 1
            return count

        # Print a summary of the number of logged event per recording
        summary = []
        for r in recordings:
                summary.append([r,events_nb([line for line in tmp if line[recording_name] == r]
```

In [99]:
```
        # Creating a dictionary
        D = {l[0]: l[1] for l in summary}
        D = collections.OrderedDict(sorted(D.items()))

        # Printing the results
        plt.figure(figsize=(60,10))
        plt.bar(range(len(D)), D.values(), align='center')
        plt.xticks(range(len(D)), D.keys(), rotation='vertical')
        plt.tight_layout()
        savefig('nb_events.pdf', bbox_inches='tight')
        plt.show()
```



In [101]:
```
         # TODO - do it with CSV files, with tryout type.
         errors = []
         for key, value in D.iteritems():
             if int(value) not in [16, 17, 19]:
                 errors.append([key, value])
         print str(len(errors)) + " recordings with wrong number of logged events. "
```

```
82 recordings with wrong number of logged events.
```

In [ ]:

In [ ]: