Pupil dilatation

June 21, 2017

1 Eye tracking - Pupil Dilatation Analysis

This part focuses on working over pupil dilatation on the 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 numpy as np
        import matplotlib.pyplot as plt
        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

    # Returns list of recordings for one participant in data
    def get_recordings_per_user(data, user):
        records = list(set([l[recording_name] for l in data[1:] if str(l[participant]) == st
        return records
```

```
In [4]: def pupil_analysis(d, user, recording, show=True, save=False):

            # Creating the graph
            f, axarr = plt.subplots(3, sharex=True)

            # Getting the diameter values
            d_user = [line for line in d if line[participant] == user and line[recording_name] =

            if len(d_user) is 0:
                print("No data for " + str(recording) + " for " + str(participant))
                return

            diam = [[x[time_column], x[pupil_diam_left].replace(',', '.'), x[pupil_diam_right].r
                    for x in d_user[1:]]
            events = [x[time_column] for x in d_user if x[event_column] == "Logged live Event"]

            # 0: Getting the recording time values and pupil diameter
            diam_x = [float(x[0]) for x in diam if is_number(x[1]) and is_number(x[2])]
            diam_y = [(float(x[1]) + float(x[2]))/float(2) for x in diam if is_number(x[1]) and

            # 0: Printing the values of pupil diameter
            axarr[0].set_title('Pupil diameter over time')
            axarr[0].plot(diam_x, diam_y, color='blue')

            # 0: Printing the mean value of pupil
            mean = np.array(diam_y).mean()
            axarr[0].plot((min(diam_x), max(diam_x)), (mean, mean), color='red')
```

```python
# 0: Printing the events on the map
for event in events:
    axarr[0].plot(event, mean, marker='o', color='green')

# 0: Smoothing the curve
def smooth(y, box_pts):
    box = np.ones(box_pts) / box_pts
    y_smooth = np.convolve(y, box, mode='same')
    return y_smooth
# Deleting the start and the end where errors are introduced
smooth_diam_y = smooth(diam_y, 5)
axarr[0].plot(diam_x, smooth_diam_y, color= 'green')

# 1: Pupil gradient over time
axarr[1].set_title('Pupil gradient over time')
grad_diam_y = np.gradient(np.array(diam_y))
axarr[1].plot(diam_x, grad_diam_y * 100, color='grey')
grad_diam_y_bis = np.gradient(np.array(smooth_diam_y))
axarr[1].plot(diam_x, grad_diam_y_bis * 100, color='blue')
for event in events:
    axarr[1].plot(event, min(grad_diam_y), marker='o', color='green')
# 1: Printing the mean value of pupil
mean = np.array(smooth_diam_y).mean()
axarr[0].plot((min(diam_x), max(diam_x)), (mean, mean), color='red')

# TODO dynamically compute threshold
def get_pics(x, y, threshold):
    pics_x = []
    for index in range(0, len(y)-1):
        if y[index] > threshold or y[index] < -threshold:
            pics_x.append(x[index])
    return pics_x

pics_x = get_pics(diam_x, grad_diam_y, 0.15)
pics_x_bis = get_pics(diam_x, grad_diam_y_bis, 0.05)

axarr[2].set_title("Gradient optimum and events distribution")
for event in events:
    axarr[2].plot(event, 0, marker='o', color='green')
# TODO add lines for event positions
for pic in pics_x:
    axarr[2].plot(pic, 1, marker='x', color='red')
for pic in pics_x_bis:
    axarr[2].plot(pic, 2, marker='x', color='blue')
axarr[2].plot(0, -2)
axarr[2].plot(0, 3)
```

```
        # IDEA: check if the gradient optimum arrives between the start and end of events
        """
        If they do, it means it has reaction to stimuli otherwise means he is distracted and
        cannot be used. Compute percentage of gradient pics among event times.
        """

        # Printing
        f.tight_layout()
        f.set_size_inches(20, 18, forward=True)
        plt.ylabel('Pupil diameter (mm)')
        plt.xlabel('Time (ms)')

        # Save pictures
        if save == True:
            file_name = 'pupil_'+ str(user) + '_' + str(recording)
            savefig(file_name + '.png', bbox_inches='tight')
            savefig(file_name + '.pdf', bbox_inches='tight')

        if show == True:
            plt.show()
```

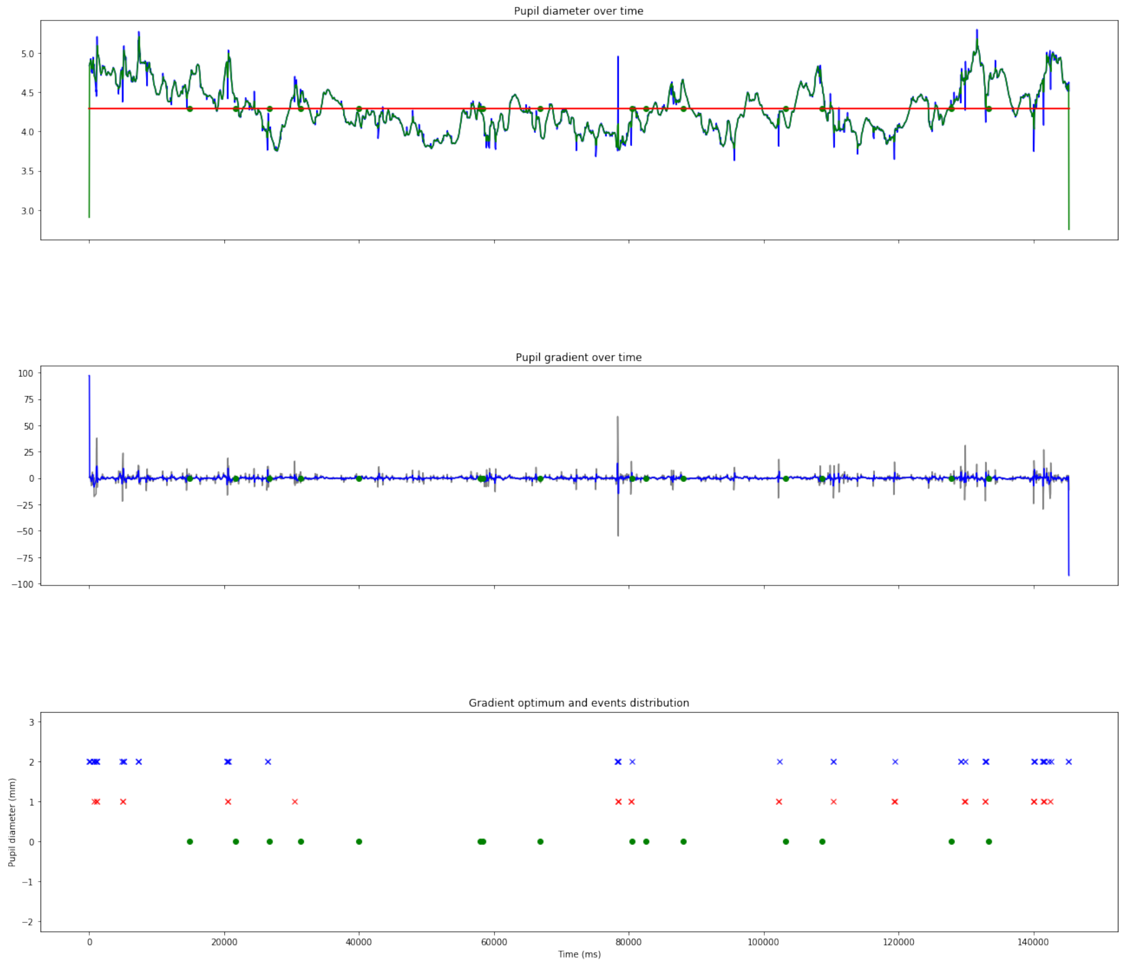
```
In [5]: # Reading the eye tracking data from 3 different files
        if True:
            tmp = load("/media/sf_EyeTracking/data/short_1_35_unicode.txt")
            tmp.extend(load("/media/sf_EyeTracking/data/short_36_51_unicode.txt"))
            tmp.extend(load("/media/sf_EyeTracking/data/short_52_71_unicode.txt"))

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

```
Data contains 2155045 rows.
```

The following grap is split in three subgraphs: - the first one shows the variation of size of pupil diameter over time in blue, a smoothen version of the curve is displayed in green (trying to cover then big variations that might be due to eye-blinking), finally a line showing the average pupil size is displayed in red. The dots in red on the average line represents the event recorded during the live recording, - the second subplots shows the gradient of the pupil diameter over time (normal one in grey, soften one in blue). The green points still show the events. - the third subplot represents the localation over time of the optimums found on the gradients and the events recorded live. The idea is to find some correlation between the number of optimums (showing then variations of the pupil size) around the live recorded events.

```
In [6]: pupil_analysis(tmp, "43", "Recording177")
```



In [29]:
```
# Generate all the graphs
# users = get_users(tmp)
# [pupil_analysis(tmp, user, record, False, True) for user in users for record in  get_
```

Out[29]:
```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
```

```
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None,
None]
```