Eye Sample Noise Reduction Filters

Purposes:

I. Reduction of high frequency noise from eye position data.
II. Decreasing velocity and accelleration noise in the eye signal to aid in improved Eye Event Detection (Saccades, Fixations, Blinks).

Notes:

I. The requirements of the experimental paradigm being used, and the need for real-time eye data signals, will often influence which Filtering alogorithms can be used for a particular study.
II. There is usually a trade-off between the number of data points required by a filtering algorithm / the computational complexity in performing the filter and the quality of the filtered output.
III. An ideal filter would:
    A. require few sample points per calculation, or require only previously collected data points when filting the current point.
    B. be computationally inexpensive (i.e. fast to calculate)
    C. maintain real changes in eye signal positions, throughout the full biological oculomotor frequency range, while removing artificial signal fluctutions and artifacts.
IV. I'm not aware of any filter that is /ideal/.

Example 1: Using Various Window Based Filters. SciPy.signal to the rescue....

Good Real-Time Filters

In [9]:
```
Code TBC
```

```
  File "<ipython-input-9-d697675ba635>", line 1
    Code TBC
           ^
SyntaxError: invalid syntax
```

Example 2: Savitzky Golay Smoothing Filter

Good *Post Hoc* Filter

Quoting from wikipedia:

The Savitzky–Golay smoothing filter is a filter that essentially performs a local polynomial regression (of degree k) on a series of values (of at least k+1 points which are treated as being equally spaced in the series) to determine the smoothed value for each point. The main advantage of this approach is that it tends to preserve features of the distribution such as relative maxima, minima and width, which are usually 'flattened' by other adjacent averaging techniques (like moving averages, for example).

This filtering algorithm was first described in 1964 by Abraham Savitzky and Marcel J. E. Golay.

Savitzky and Golay's paper is one of the most widely cited papers in the journal Analytical Chemistry and is classed by that journal as one of its "10 seminal papers" saying "it can be argued that the dawn of the computer-controlled analytical instrument can be traced to this article"

Python Implementation written by Dr. Thomas Haslwanter

SciPy / Numpy implementation from the SciPy Cookbook

In [2]:
```
# SciPy / Numpy implementation from the SciPy Cookbook; recipe implemented by Thomas H
# http://www.scipy.org/Cookbook/SavitzkyGolay

import numpy as np
import matplotlib.pyplot as plt

def savitzky_golay_1D(y, window_size, order, deriv=0, rate=1):
    r"""Smooth (and optionally differentiate) data with a Savitzky-Golay filter.
    The Savitzky-Golay filter removes high frequency noise from data.
    It has the advantage of preserving the original shape and
    features of the signal better than other types of filtering
    approaches, such as moving averages techniques.

    Parameters
    ----------
    y : array_like, shape (N,)
        the values of the time history of the signal.
    window_size : int
        the length of the window. Must be an odd integer number.
    order : int
        the order of the polynomial used in the filtering.
        Must be less then `window_size` - 1.
    deriv: int
        the order of the derivative to compute (default = 0 means only smoothing)

    Returns
    -------
    ys : ndarray, shape (N)
        the smoothed signal (or it's n-th derivative).

    Notes
    -----
    The Savitzky-Golay is a type of low-pass filter, particularly
    suited for smoothing noisy data. The main idea behind this
    approach is to make for each point a least-square fit with a
    polynomial of high order over a odd-sized window centered at
    the point.

    References
    ----------
    .. [1] A. Savitzky, M. J. E. Golay, Smoothing and Differentiation of
       Data by Simplified Least Squares Procedures. Analytical
       Chemistry, 1964, 36 (8), pp 1627-1639.
    .. [2] Numerical Recipes 3rd Edition: The Art of Scientific Computing
       W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery
       Cambridge University Press ISBN-13: 9780521880688
    """
    import numpy as np
    from math import factorial

    try:
        window_size = np.abs(np.int(window_size))
        order = np.abs(np.int(order))
    except ValueError, msg:
        raise ValueError("window_size and order have to be of type int")
    if window_size % 2 != 1 or window_size < 1:
        raise TypeError("window_size size must be a positive odd number")
    if window_size < order + 2:
        raise TypeError("window_size is too small for the polynomials order")
    order_range = range(order+1)
    half_window = (window_size -1) // 2
```

Testing with Eye Data (binocular, 120 Hz, remote head-free eye tracker)

Loading the 'Sample' data....

In [3]:
```
import psychopy.iohub
from psychopy.iohub.datastore.util import displayDataFileSelectionDialog, ExperimentDa
from psychopy.iohub import EventConstants

import matplotlib.pyplot as plt
import matplotlib.transforms as mtransforms

import os

orginal_plt_width=None
orginal_plt_height=None
plt.rcParams['figure.figsize'] =6,4
if orginal_plt_width is None:
    orginal_plt_width,orginal_plt_height=plt.rcParams['figure.figsize']
    plt.rcParams['figure.figsize'] =orginal_plt_width*4,orginal_plt_height*2.5

def loadSampleData():
    # Open an ioDataStore HDF5 file.
    #
    data_file_path= displayDataFileSelectionDialog(psychopy.iohub.module_directory(sav
    if data_file_path is None:
        sys.exit(0)
    dpath,dfile=os.path.split(data_file_path)

    # Create an instance of the ExperimentDataAccessUtility class
    # for the selected DataStore file. This allows us to access data
    # in the file based on Device Event names and attributes.
    #
    dataAccessUtil=ExperimentDataAccessUtility(dpath,dfile, experimentCode=None,sessio

    # Retrieve a subset of the BINOCULAR_EYE_SAMPLE event attributes, for events that
    # between each time period defined by the TRIAL_START and TRIAL_END trial variable
    # in the trial_conditions data table.
    #
    session_trial_sample_data=dataAccessUtil.getEventAttributeValues(EventConstants.BI
                                ['time','left_gaze_x','left_gaze_y','right_gaze_x','le
                                 'right_gaze_y','right_pupil_measure1','status'],
                                conditionVariablesFilter=None,
                                startConditions={'time':('>=','@TRIAL_START@')},
                                endConditions={'time':('<=','@TRIAL_END@')})


    dataAccessUtil.close()
    return session_trial_sample_data
```

Filling in Gaps in Data Stream so the Full Trial Duration can be Filtered

In [4]:
```
def handleTemporalGaps(tsamples):
    notmasked_contiguous=np.ma.extras.notmasked_contiguous
    marray=np.ma.array
    status=tsamples.status
    left_pupil_measure1=tsamples.left_pupil_measure1
    right_pupil_measure1=tsamples.right_pupil_measure1
    left_gaze_x=tsamples.left_gaze_x
    left_gaze_y=tsamples.left_gaze_y
    right_gaze_x=tsamples.right_gaze_x
    right_gaze_y=tsamples.right_gaze_y

    missing_left_indexes=status//10>=2
    missing_right_indexes=status%10>=2
    left_pupil_measure1[missing_left_indexes]=0
    right_pupil_measure1[missing_right_indexes]=0
    left_valid_data_periods=notmasked_contiguous(marray(left_pupil_measure1, mask=miss
    right_valid_data_periods=notmasked_contiguous(marray(right_pupil_measure1, mask=mi

    #Left eye fill in for filtering continuity
    left_gaze_x[0:left_valid_data_periods[0].start]=left_gaze_x[left_valid_data_period
    left_gaze_y[0:left_valid_data_periods[0].start]=left_gaze_y[left_valid_data_period
    last_slice_end=left_valid_data_periods[0].stop
    for data_slice in left_valid_data_periods[1:]:
        invalid_1=last_slice_end
        invalid_2=data_slice.start
        valcount=(invalid_2-invalid_1)
        # left_x
        startval=left_gaze_x[invalid_1-1]
        endval=left_gaze_x[invalid_2]
        left_gaze_x[invalid_1:invalid_2]=np.arange(startval,endval,(endval-startval)/v
        # left_y
        startval=left_gaze_y[invalid_1-1]
        endval=left_gaze_y[invalid_2]
        left_gaze_y[invalid_1:invalid_2]=np.arange(startval,endval,(endval-startval)/v
        last_slice_end= data_slice.stop

    #Right eye fill in for filtering continuity
    right_gaze_x[0:right_valid_data_periods[0].start]=right_gaze_x[right_valid_data_pe
    right_gaze_y[0:right_valid_data_periods[0].start]=right_gaze_y[right_valid_data_pe
    last_slice_end=right_valid_data_periods[0].stop
    for data_slice in right_valid_data_periods[1:]:
        invalid_1=last_slice_end
        invalid_2=data_slice.start
        valcount=(invalid_2-invalid_1)
        # left_x
        startval=right_gaze_x[invalid_1-1]
        endval=right_gaze_x[invalid_2]
        right_gaze_x[invalid_1:invalid_2]=np.arange(startval,endval,(endval-startval)/
        # left_y
        startval=right_gaze_y[invalid_1-1]
        endval=right_gaze_y[invalid_2]
        right_gaze_y[invalid_1:invalid_2]=np.arange(startval,endval,(endval-startval)/
        last_slice_end= data_slice.stop

    return missing_left_indexes,missing_right_indexes
```

Applying the Savitzky Golay Smoothing Filter

In [5]:
```
def filterSamplesFromTrial(trial_id,tsamples):
        poly_order=4
        # msec sampling rate of eye tracker
        inter_sample_interval=1000.0/120.0
        # desired filtering window size in msec
        temporal_window=200.0
        # window size in sample count
        window_size=int(temporal_window/inter_sample_interval)
        if window_size%2==0:
           window_size+=1
        #print 'Window Size: ', window_size, window_size*inter_sample_interval

        if poly_order>window_size-1:
            print "Window size must be increased, or order of polynomial fit must be d
            return None

        time = tsamples.time
        left_x=tsamples.left_gaze_x
        left_y=tsamples.left_gaze_y
        left_pupil=tsamples.left_pupil_measure1
        right_x=tsamples.right_gaze_x
        right_y=tsamples.right_gaze_y
        right_pupil=tsamples.right_pupil_measure1

        left_x_sg = savitzky_golay_1D(left_x, window_size=window_size, order=poly_orde
        left_y_sg = savitzky_golay_1D(left_y, window_size=window_size, order=poly_orde
        right_x_sg = savitzky_golay_1D(right_x, window_size=window_size, order=poly_or
        right_y_sg = savitzky_golay_1D(right_y, window_size=window_size, order=poly_or


        return time,(left_x,left_y,left_pupil,right_x,right_y,right_pupil),(left_x_sg,
```

Plotting Filtered vs Unfiltered Sample Traces

In [7]:
```
def plotFilterResults(trial_id,time,unfiltered_data,filtered_data):
    plt.clf()

    left_x,left_y,left_pupil,right_x,right_y,right_pupil=unfiltered_data
    left_x_sg,left_y_sg,right_x_sg,right_y_sg=filtered_data

    fig = plt.figure()
    ax2 = fig.add_subplot(212)
    ax1 = fig.add_subplot(211,sharex=ax2)
    ax1.plot(time,left_x,label='Unfiltered Horizontal Position')
    ax1.plot(time,left_y,label='Unfiltered Vertical Position')
    ax1.plot(time,left_x_sg,label='Filtered Horizontal Position')
    ax1.plot(time,left_y_sg,label='Filtered Vertical Position')
    ax2.plot(time,right_x,label='Unfiltered Horizontal Position')
    ax2.plot(time,right_y,label='Unfiltered Vertical Position')
    ax2.plot(time,right_x_sg,label='Filtered Horizontal Position')
    ax2.plot(time,right_y_sg,label='Filtered Vertical Position')
    ax2.set_xlabel('Time')
    ax1.set_ylabel('Position (pixels)')
    ax2.set_ylabel('Position (pixels)')
    ax1.set_title("Left Eye Position Traces: Trial %d"%(trial_id,))
    ax2.set_title("Right Eye Position Traces: Trial %d"%(trial_id,))
    tmin=time.min()//1
    tmax=time.max()//1+1
    #trange=tmax-tmin
    plt.xticks(np.arange(tmin,tmax,0.5),rotation='vertical')

    trans1 = mtransforms.blended_transform_factory(ax1.transData, ax1.transAxes)
    trans2 = mtransforms.blended_transform_factory(ax2.transData, ax2.transAxes)
    ax1.fill_between(time, 0, 1, where=left_pupil==0, facecolor='DarkRed', alpha=0.5,
    ax2.fill_between(time, 0, 1, where=right_pupil==0, facecolor='DarkRed', alpha=0.5,

    plt.legend()
    plt.show()

if __name__ == '__main__':
    all_sample_data=loadSampleData()

    for t,tsamples in enumerate(all_sample_data):
        missing_left_indexes,missing_right_indexes=handleTemporalGaps(tsamples)

        time,(left_x,left_y,left_pupil,right_x,right_y,right_pupil),(left_x_sg,left_y_

        left_x[missing_left_indexes]=np.NaN
        left_y[missing_left_indexes]=np.NaN
        right_x[missing_right_indexes]=np.NaN
        right_y[missing_right_indexes]=np.NaN
        left_x_sg[missing_left_indexes]=np.NaN
        left_y_sg[missing_left_indexes]=np.NaN
        right_x_sg[missing_right_indexes]=np.NaN
        right_y_sg[missing_right_indexes]=np.NaN

        plotFilterResults(t+1,time,(left_x,left_y,left_pupil,right_x,right_y,right_pup

        del time
        del left_x
        del left_y
        del right_x
        del right_y
        del left_x_sg
        del left_y_sg
```

That's It!