Introduction

In this assignment, we will be looking at some of the basic 1-D signal transforms we've seen in class. The goal of this assignment is to get a practical feel for what these transforms tell us about the data. We will be focusing on the Short-Time Fourier Transform (STFT) and the Haar transform, and applying these to some real-world signals.

Submission details

- This assignment can be done individually or in pairs (though we strongly encourage you to work in pairs).
- Programs should be in Python. We prefer Python version 2, but we will allow version 3 also. Please state the Python version at the beginning of your writeup.
- The example Python code uses the SciPy library (http://www.scipy.org), which provides Matlab-like arrays with similar functionality. It also has some handy functions for plotting graphs of signals, which the Python standard library lacks. These operations tend to make signal processing algorithms much easier to code. You are not required to use this library, but it could make your life easier.
- To install (email TA if you run into trouble)
    - Ubuntu Linux:
      sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose
    - On Windows or Mac, you may have to install a Python distribution containing SciPy. We recommend the Anaconda distribution, but other distributions are also possible.
- For submission, package up your code as a zip file. Include your written answers as a pdf or word file named writeup.[pdf|docx]. There's a bunch of graphs we want you to plot, include these in your writeup, and please label them so we know which figures go with which sub-problem.
- Send the zip file to the TA (see next bullet). Add the course number to the subject of the mail.
- If you have any questions about this assignment, please contact the TA (stinger@tx.technion.ac.il). Add the course number to the subject of the mail.

Task 1: Haar Wavelet Transform

Recall the Haar transform from class, which computes averages and differences of larger and larger windows of the signal. We want to use the Haar transform to analyze some real-world signals, so your first task is to implement it.

1. Write a function **haar(x)** which takes a signal x, given as a sequence of floating-point numbers, and returns the Haar transformed signal. Recall that the Haar transform only works on a signal which has a length of 2^n. For this part of the assignment, you can assume that x has a length which is a power of 2. The transformed result X should have the same length as x.
2. Write a function **inverse_haar(X)** which reverses the Haar transformation. You can also assume that X has a length which is a power of 2.

Task 2: Smoothing accelerometer data

We are going to use the Haar transform to analyze the signal from a smartphone accelerometer as the user performs various actions. We provided you with a few example data.

The example code in accel_example.py loads an accelerometer reading from a file, and graphs the signal from one direction of motion. You should use this code as a starting point.

A. Haar transformed data

First, some basic applications of the Haar transform to this data.

1. Use the function you wrote in Task 1 to transform the data. Recall that Haar requires input which has a length of 2^n, so you should first trim the input to the largest 2^n-sized subsequence.
2. Graph the result using **graph_accel()**.

B. Haar transforms and "edges"

One particular reason for using the Haar transform over other wavelet transforms is that the basic operation it uses is taking pairwise differences. Pairwise differences will be large when the signal is changing rapidly, and small when the signal is constant, so we can use the Haar transform to find rapid changes in the signal, or "steps".

First, look at just the second half (i.e., the last 2^{n-1} entries) of the Haar-transformed signal. These correspond to pairwise differences of individual entries of the original signal. These are typically referred to as the first-order differences, so we'll denote this by X^1.

Write a function to find the 5 biggest positive entries of the pairwise differences X^1, and the 5 most negative entries. Briefly (one or two sentences) describe what is happening to the original signal at the locations corresponding to these points.

C. Smoothing

Loosely speaking, the second half of the Haar transformed signal captures rapid changes in the original signal, while the earlier entries capture more slowly varying changes. This suggests that we can get a smoothed version of the input by ignoring the information in the second half of the Haar-transformed signal.

In particular, we already have a way of inverting the Haar-transform, the function **inverse_haar(X)** that you wrote earlier. So, we can just throw away the information corresponding to rapid changes, and take the inverse transform of the result. Let's see what happens when we do this.

1. First, take the original signal x and compute its Haar-transform X. Call **inverse_haar(X)** and graph the result. Compare the result to the original signal (hint: they should be identical).
2. Start with the Haar-transformed signal X, and truncate X to the first 2^(n-1) entries. Take the inverse Haar-transform of this truncated X, and graph the result.
3. Do the same thing with the first 2^(n-k) entries, for k = 2, 3 and 4, graphing each result. Briefly describe (1 to 2 sentences) what happens as k increases.

Task 3: Aligning Audio Signals

To make this a concrete assignment you are actually going to try and solve a real problem. Aligning audio signals is an important challenge in Signal Processing, often used for audio identification and synchronization. In this task we provide you with a relatively easy case of audio alignment. The underlying assumption is that all audio tracks have a mutual phrase that could be used for temporal alignment. In our case it will be the phrase "early in the morning".

You will try to align the audio tracks using two different methods that we've learned in class. You may also suggest a different method of your own. If you chose to do that, please correspond with the TA and me before you start to make sure the implementation of your idea is reasonable in scope (See section C). We provide 8 audio tracks, and you will also record one of your own.

A. Using STFT

We've provided you with code to compute the STFT of a signal, and to render the STFT as a spectrogram. Recall that the STFT is an array X where X[t,f] is the response of frequency f at the window starting at time t. There is also a handy function in the scipy library for loading .wav files into scipy arrays: scipy.io.wavfile.read.

The usage of these functions is demonstrated in **audio_example.py**, which you should feel free to use as a starting point for your solution.

Since the full STFT is too large, we'll try to find some interesting features of the transformed signal, and see if we can use those for aligning the tracks. Like Shazam, we're going to be looking at the peaks of the transformed signal. A peak is a time-frequency pair (t; f) which has bigger value X[t,f] than any other (t0; f0) which is nearby. By nearby, we mean all the other points in a K-by-K grid around (t; f).

1. Choose 4 tracks out of the provided 8, and set one of them as your reference track. You can choose yourself which tracks you use, but write their names next to the plots.
2. Write a function which computes all the peaks of a transformed signal X which are the biggest values in a 20 x 20 surrounding grid (K=20).
3. Using the function **plot_peaks()** in the example code, plot the peaks and spectrogram of the tracks.
4. We have provided for each of the tracks the start and end points of the mutual phrase. Mark the start and end points of the phrase on each peaks-map by using the second and third parameters of the **plot_peaks()** function.
5. Write a few sentences about the similarity between the peaks maps of the four tracks you chose. Do you think they could be aligned?
6. Build a 1-D array y[t] for each audio signal, where y[t]= 0 if there is no peak, and **y[t]=peak_value** if there is one. If there is more than one, choose one of them.
7. Take the 3 pairs of tracks and use cross-correlation to find the best alignment between the corresponding 1D arrays. How successful was this?
8. Re-record one of the three tracks that you chose, and repeat items 3-7. What are the results?

B. Using Haar wavelets

Next thing we'll do is try the same thing but this time using Haar wavelets instead of Fourier transform.

1. Write a function **short_time_haar(x)**. This function is analogous to the STFT, in that we break up the signal x into windows. However, take note of the difference in the nature of the data arising from these two transforms and write your function accordingly. We'll arbitrarily choose a window size of 4096 and a shift of 2048. So, your windows should be x[0:4096], x[2048:2048+4096], x[4096:4096+4096],….
   Trim the signal x to the largest sub-signal that supports this window size.
   On each window, compute the Haar transform to get a 2D array, where X[t,k] holds the k'th Haar coefficient of the window at time t.
2. For the same 4 tracks as in A, plot the Short-time Haar transform, using **plot_transform**.
3. Explain what's going on. Write a few sentences (2-4) explaining why the STFT might be better for analyzing audio signals than the Haar transform, (or maybe you think otherwise?).
4. Repeat items 2-3 with your recorded track from A.8. and compare the results.

C. Using your own solution

If you have your own idea for aligning the tracks let us know. If we confirm it, you could try it out instead of the Haar wavelets Task 3.B.