Institute for Cognitive Systems
Technische Universität München
Prof. Gordon Cheng

Biosignal Processing and Modeling
Tutorial – LSL Streams: Acquiring, Plotting and Filtering

Course Instructors: Alireza Malekmohammadi `alireza.malekmohammadi@tum.de`,
Nicolas Berberich, `n.berberich@tum.de`
Teaching Assistant: Karahan Yilmazer, `karahan.yilmazer@tum.de`

Summer Semester 2023

Introduction

Hopefully, you went through the cumbersome setup procedure and now the exciting part starts! In this tutorial you will learn how to

- create mock LSL streams
- visualize LSL streams
- filter incoming EEG data in real-time

Info: These tutorials are designed keeping in mind that the class comprises of people from different backgrounds. Do not be overwhelmed, the topics will be covered in as much detail as required to enable you to start using the technologies presented in the tutorials.

1 LSL Inlet and Outlet

As we have learned in the previous tutorial, Lab Streaming Layer (LSL) is a great tool to synchronize different real-time data streams under one framework. Now we will see how we can manage LSL streams in Python. For this, switch to the folder **`T1-lsl-streams`**.

Here you will find different scripts. As the names suggest, `lsl_outlet.py` creates an LSL outlet which streams data out. You can run this code cell by cell by pressing Shift+Enter in VS Code.

While an outlet streams data, an inlet captures the data streamed by an outlet. In the script `lsl_inlet.py`, you will see how you can connect an inlet to an outlet, extract information about the stream and receive data and corresponding time stamps. For this script to work, you will need an active outlet streaming data. You can either set this up using the `lsl_outlet.py` script or using the external Python library called liesl to create a mock LSL outlet.

If you want to use your own outlet, simply hit Shift+Enter in the outlet script and until you run the cell containing the line `outlet.push_sample(...)` or the cell below containing `outlet.push_chunk(...)`. As long as you don't close the interactive window on the right or kill your outlet you will be able to access the data you stream from your inlet script. For this, uncomment the line `inlet_name = 'test_outlet'`. Now, Shift+Enter your way through until you start pulling samples from the stream.

If you want to go with the second option, you will have to create a mock LSL outlet by opening up a terminal (Terminal –> New Terminal in VS Code) and typing in the following command:

Create a Mock LSL Stream Outlet

```
liesl mock --type EEG
```

Now, if you run the `lsl_inlet.py` script cell by cell, you will see the stream information and data from this mock outlet.

The two other scripts `lsl_print_sample.py` and `lsl_save_to_csv.py` do as their names suggest: the first one prints the received data into the terminal and the second one saves the data into a CSV file. These scripts are provided for your understanding. Do not use them to record data as this might cause time synchronization problems in your recording. As mentioned in the previous tutorial, if you want to record LSL streams, the right way to do this is through LabRecorder.

2 Real-Time Plotting of a Mock LSL Stream

Receiving data and seeing its value is great but visualizing it would be better. Now, you will create a liesl mock LSL stream (if you haven't done already). For this refer to the line in "Create a Mock LSL Stream Outlet" code box above.

This will create an LSL stream with 8 channels. The terminal you used for creating the mock stream will be occupied as long as it is running. So, open up a new terminal and type in the following line to pick up this stream and plot the incoming signal:

Plot Incoming LSL Stream

```
python -m pylsl.examples.ReceiveAndPlot
```

At this point, you should be seeing a plot of the incoming data. As you can see, real-time plotting of LSL streams is as easy as that. However, distinguishing between different channels is rather difficult due to the monochrome plotting. A slightly modified version of this plotting script can be found in the folder for Tutorial 1. The script is called `plot_lsl_stream.py`.

You can run this script to get the same plotting but now with different colors and subplots for each channel. Furthermore, you can change set four new parameters for some customization:

- `ylims`: y-axis limits for all the streams
- `single_stream_name`: name of a specific stream to plot
- `stream_str`: substring to look for in the available stream names –> only the streams that involve this name will be plotted
- `lim_name`: name of the stream to limit the y-axis of

Warning:
Play around with the parameters `plot_duration`, `update_interval` and `pull_interval` to see their effect on the plotting. You could also look through the code and try adding your own customizations for better plotting.

3 Streaming EEG Data Through LSL

Now that we know how LSL streams look like on an abstract level, we can transfer this knowledge to stream EEG data from the Unicorn Hybrid Black. If you still have not set up the Unicorn LSL plugin to stream data from the Unicorn headset, refer to the first tutorial.

Make sure that you killed the terminal that is creating the mock LSL stream. After that start streaming real EEG data from the Unicorn and try plotting it again with `plot_lsl_stream.py`.

Depending on the environment you are recording in, you will most likely see a strong line noise in the signal, just like in Figure 1.

The source of this high frequency line noise is the electrical fluctuations in the power line. The European standard for AC power plugs is 50 Hz, which is why we are seeing a 50 Hz sine wave superimposed on our signal. It is worth noting that depending on the country, you could also get a 60 Hz line noise.

Another thing we can see in Figure 1 is the clear distortion in all channels at the middle of the plot which is caused by a strong eye blink. This should illustrate why eye blinks are seen as artifacts and that they should be avoided as much as possible during EEG recordings.

However, we can also exploit their significance in EEG signals to design a system that reliably detects eye blinks and uses them as control signals but more on that in the next tutorial.

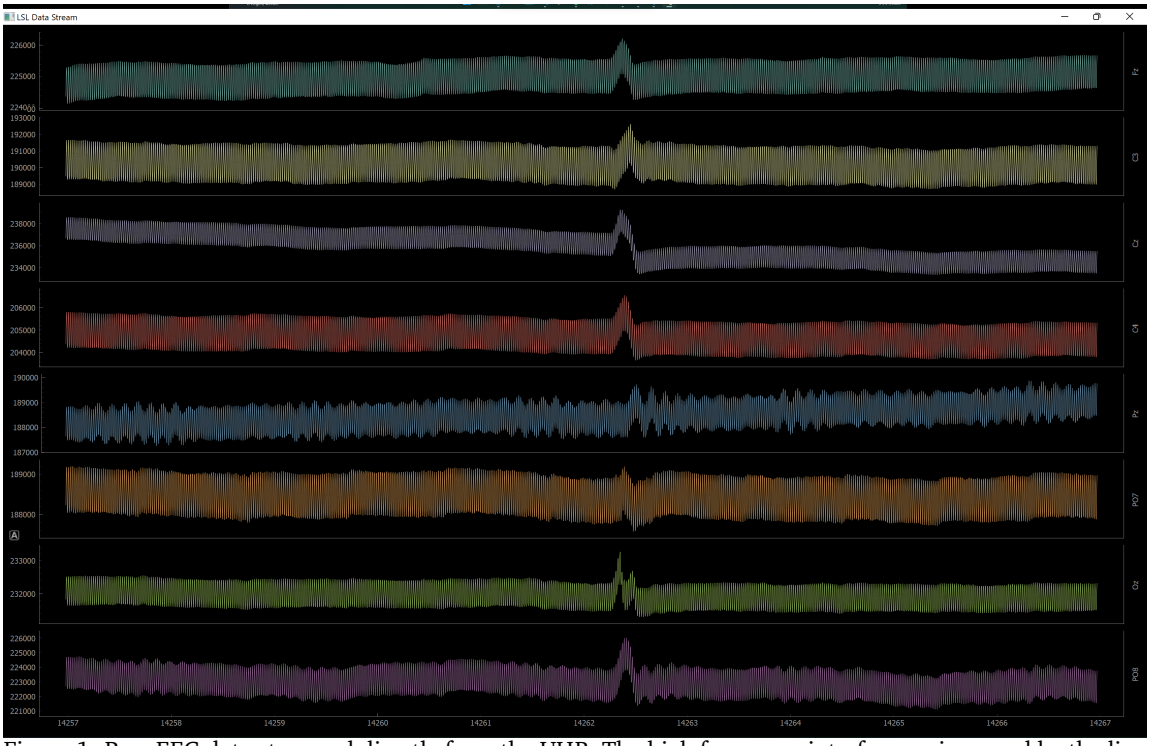

Figure 1: Raw EEG data streamed directly from the UHB. The high frequency interference is caused by the line noise.

4 Real-Time Filtering

Now that we have seen an interference caused by a specific frequency, it should be clearer why we need spectral filtering. However, as you have learned, spectral filtering is not only used for improving the signal-to-noise ratio (SNR), but also to extract EEG features.

In any case, we will be needing a real-time filtering pipeline that accepts raw EEG samples and returns the filtered signal. Fortunately, `real_time_lsl_filtering.py` does exactly that.

While UHB is still streaming data, run `real_time_lsl_filtering.py`. This script will filter the incoming EEG data using an IIR band-pass filter and stream the filtered signal as a new LSL stream. Then, in a new window, run the `plot_lsl_stream.py` script again to visualize both the raw and filtered EEG data, just like in Figure 2.

This script is provided as a template for your real-time filtering/processing pipeline. It will be your task to find the best processing steps for your own project. The script is full of comments to help you understand the code. However, feel free to ask any question you might have.

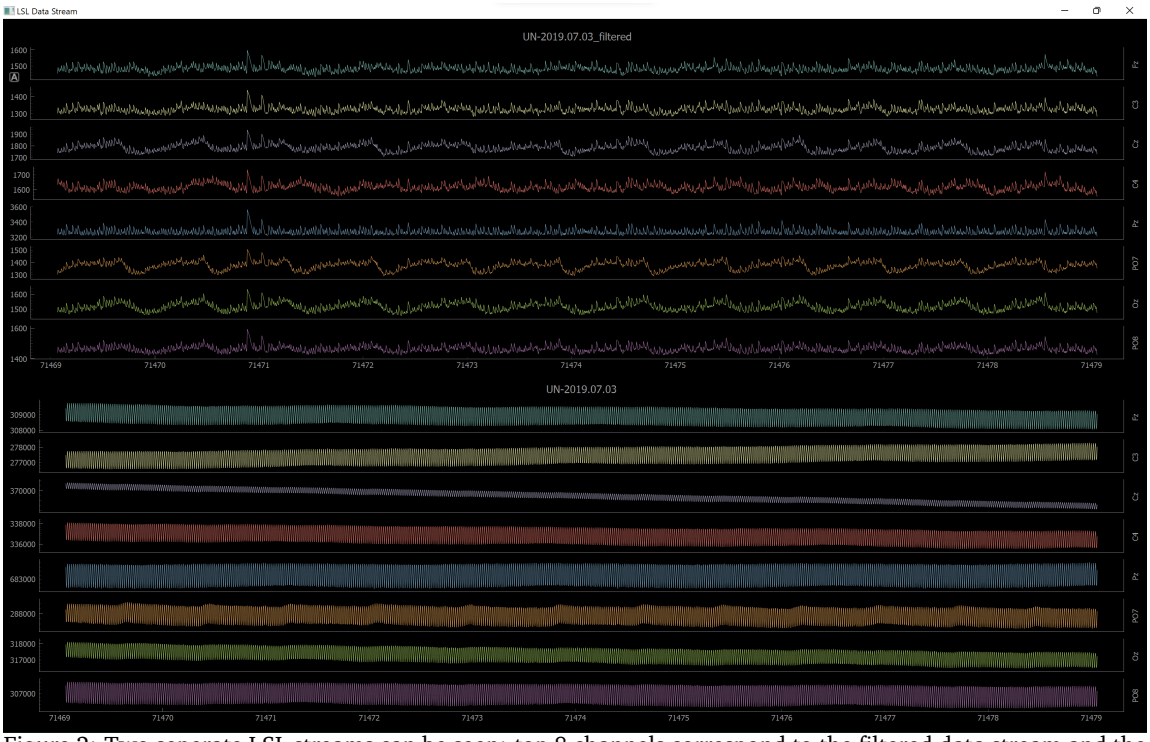

Figure 2: Two seperate LSL streams can be seen: top 8 channels correspond to the filtered data stream and the bottom 8 to the unfiltered (raw) data stream. It can be seen that adequate spectral filtering effectively removes the 50 Hz line noise.