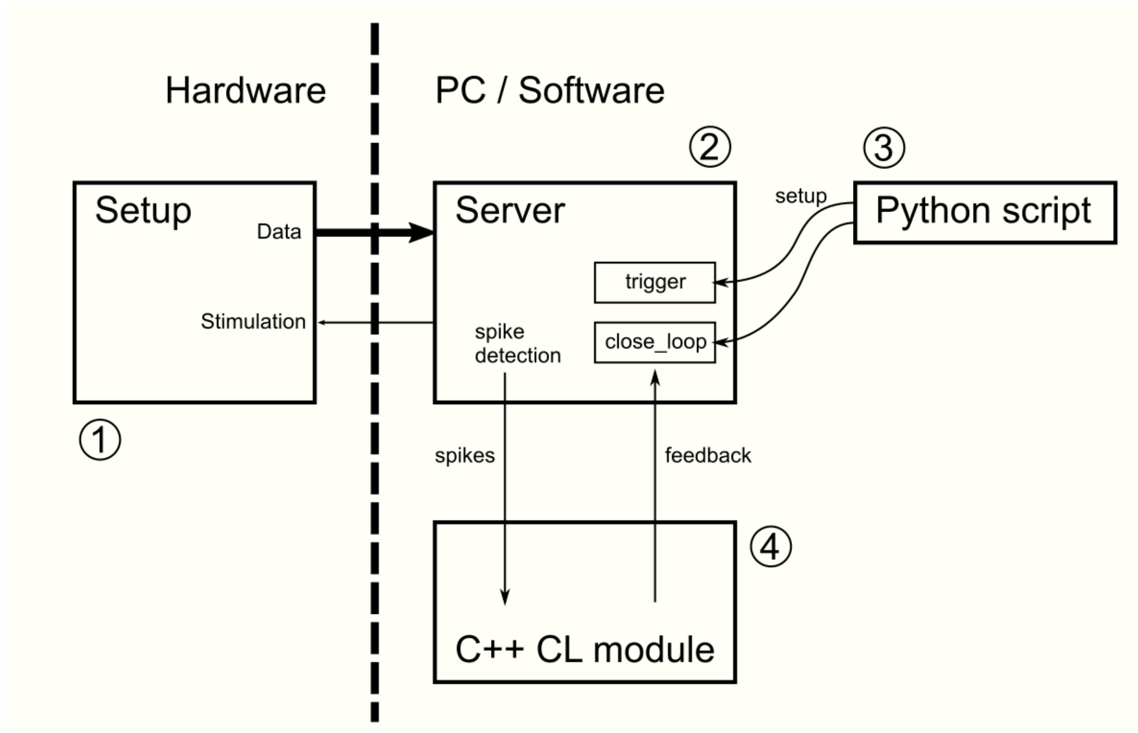
Information about CloseLoop stimulation

Introduction

This document together with the encompassing scripts and source code explains how to prepare MaxLab Live for close-loop electrical stimulation experiments. Please make sure to use the latest version of the software. Run the 'maintenancetool' in the MaxLab directory to update to the latest version.

System Overview

See below figure for a diagram of the involved components. The different components are explained below.



1. “Setup” refers to the hardware components: MaxOne chip, recording unit and MaxOneHub.
2. The server software is running on the recording PC. This software performs band-pass filtering and spike detection on the MEA data and streams the data and spikes downstream to attached clients. For example to the scope for visualization and to the close-loop C++ stimulation module.
3. A Python script is used to setup the stimulation environment, such as downloading the electrode array configuration, connecting the electrodes with stimulation units and preparing two stimulation sequences ('trigger' and 'close_loop')
4. Finally, the C++ stimulation module receives spike events (time points) form the server, checks whether the spikes occurred on the desired channel (trigger channel) and if yes, sends a stimulation pulse to the server. The stimulation pulse in this example is called 'close_loop'.

The concept of this setup is to use Python to do all the preparation and setup work. Including preparing a stimulation pulse (maxlab.Sequence) called 'close_loop'. This way, the

C++ code can be kept simple and all it needs to do is receiving data and spike-times through the network socket and decide whether it wants to send a stimulation pulse by requesting the server to send out the sequence called 'close_loop'.

To simplify testing, the example works without real cells on the dish. Instead of an actual neuron, a sequence called 'trigger' is used to simulate spike events on the electrode where the C++ code is expecting the spikes (trigger channel).

Compiling

To link the C++ program, one needs the C++ development files for zmq.

On RedHat/CentOS run below command:

```
sudo yum install cppzmq-devel
```

To compile the C++ code, us the Makefile or do:

```
g++ close_loop.cpp -lzmq -o closeLoop
```

Running

Once the binary is generated, use the following steps to run the experiment:

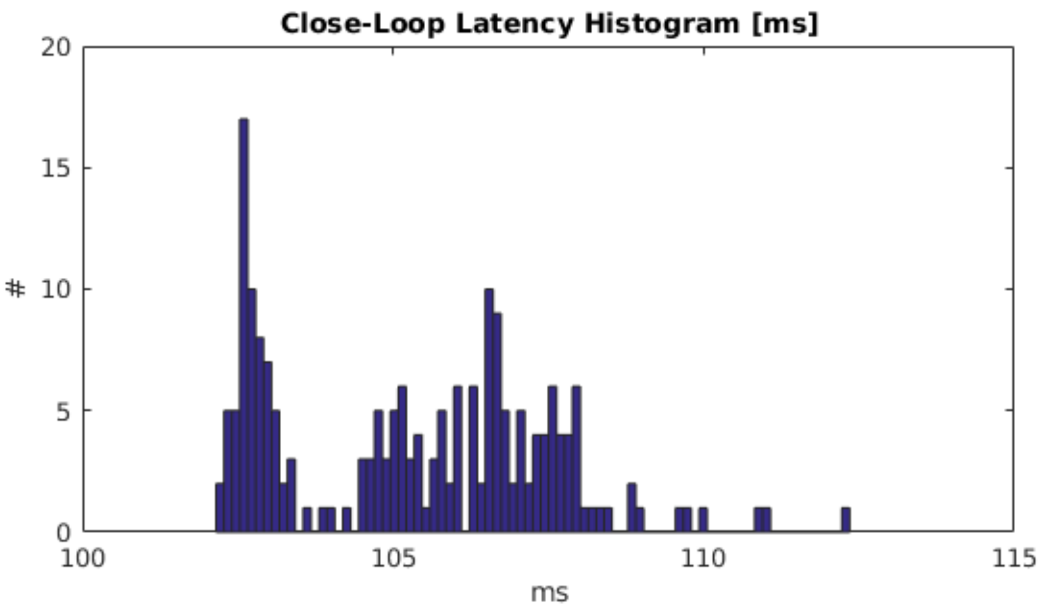
1. run the generated "closeLoop" binary on the same machine as you run the server. Give the number of the trigger channel as the single command line argument.
2. Run the python script and wait for the setup procedure to finish and wait for the 200 trigger stimulation pulses to run. They are used to simulate a spike event on the trigger channel.
3. The C++ module now detects events on the trigger channel and will immediately after detection of an event send a stimulation pulse

Improvements

One crucial thing with the close-loop stimulation is to get the refractory period correct. If not, the artifact due to the stimulation pulse will look like a spike on the trigger channel and trigger another close-loop stimulation, ending up in an infinitely repeating stimulation sequence.

Latency and Jitter

The figure below plots the close-loop feedback latency and jitter for stimulation. In the figure one can see the average delay is around 105 ms and jitter is around +/- 5 ms.



Further optimizations

Potential further optimizations are:

- reducing the computational load on the recording computer
- decrease latency of the band-pass filtering module
- etc.