Technical Report: Testing and operating the QuaeroSys Piezostimulator

MRC Cognition and Brain Sciences Unit

June 2015

Gary Chandler (gary.chandler@mrc-cbu.cam.ac.uk)
David Hayes (david.hayes@mrc-cbu.cam.ac.uk)
Mark Townsend (mark.townsend@mrc-cbu.cam.ac.uk)
Andrew Thwaites (acgt2@cam.ac.uk)

Revisions:
June 2015: Original draft
September 2019: Correction to note that QuaeroSys are still trading.

Disclaimer

The tests described here are intended to check CBSU equipment, and are not intended as a substitute for QuaeroSys users running tests on their own equipment. The CBSU cannot take responsibility for any damage caused to others' equipment as a result of readers using this document.

Table of Contents

Background

This document outlines the testing carried out on a QuaeroSys piezostimulator for use with EMEG setup at the Medical Research Council's Cognition and Brain Sciences Unit in mid-2015. QuaeroSys trade from Germany, and their website is www.quaerosys.com. Three of their piezostimulators are known to have been purchased by UK Departments: Cardiff, Birmingham and York. The following tests were carried out on the Cardiff stimulator, with kind permission of Dr David McGonigle.

The stimulator consists of a box with USB ports, trigger ports, power socket, control hardware, and braille-device ports ('the controller') and (up to 5) finger-sized blocks, which hold the pins and piezo-rods that drive the movement of the pins ('the braille-devices'*).* We will refer to 'piezostimulator' or 'stimulator' to refer to all these devices as a group.

We used Matlab [1] with Psychtoolbox [2] to present the stimulus, calling methods of QuaeroSys' provided `.dlls` [3]. Fairly thorough documentation can be found in the piezostimulators' .pdf manual [4], although it contains no example scripts for sinusoids.

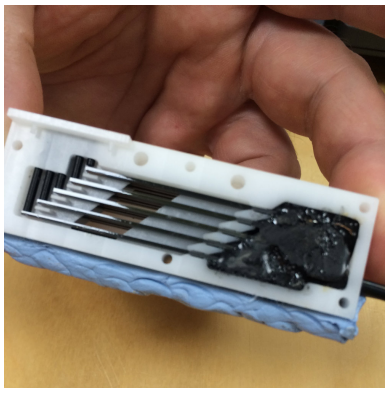


Fig one. The inside of one of the braille-devices of the stimulator, showing the piezoelectric-rods that move the pins.

In general, we found the piezostimulator to be much more accurate than expected. It has clear millisecond and height accuracy (see *test sections*) for sinusoidal type movements. Each pin is controllable with a high degree of accuracy, in a way that is fairly simple to code, provided one frames the commands in a suitable wrapper script. The device also provides methods to be able to tell if the device is dropping frames.

Although the device does work in MEG with little magnetic artifact (if shielded properly, see 'testing' below), when used with MEG *and* EEG cap/HPI coils, the MEG picks up too much noise. MEG alone, or EEG alone, is fine.

Operating

Overview

We wished to pass any arbitrary sinusoidal wave (sampled at 1000Hz) through the pins on the braille-devices. The manual only has explicit information about how to code rectangular waves, but sinusoidal waves can be coded using the updated-DAC-voltage method hinted at in the introductory pages.

The stimulator devices, when running sinusoidal waves, are fairly quiet, issuing only a slight hum. However, a square wave (the easiest movement to code using the stimulators methods), results in *very* loud clicking and violent movements of the whole device. This is not a design error, but the inevitable result of the piezoelectric-rods trying to move the pins from 0mm to 1.5mm under half-a-millisecond. It is the acceleration and deceleration of this action that causes such behavior, and, if you do need a square wave for your experiment, it worth asking if this superfast speed is necessary (there are ways to slow the movement down to say, acceleration over two milliseconds, which makes the sound/vibration a lot quieter, and is, for all intents and purposes, an imperceptible difference to the participant).

Setup

The manual can be followed quite closely to initiate setup. However, there are a couple of potential stumbling blocks for the new user. The first is that modern stimulus computers are likely to be 64bit Windows, whereas the provided .dll (stimlib0.dll) is 32bit Windows. This means that a) a 32bit version of the presentation software (Matlab, Presentation, BrainStim etc) has to be used to call the commands, and b) the stimlib0.dll should NOT be placed in `/system32/` as suggested in the manual – if it is, it will not be found (Windows does not allow access to 32bit .dlls in `/system32/`). Instead, it should be placed on the desktop, or similar, and then linked to as normal in the presentation script (see appendix for example script.)

The final point to remember is that each stimulator has a license key associated to it. The stimulus computer will *not be able to communicate with the controller unless this stimulator key is entered as part of the* `initStimulator` *command* (see example in appendix). If you are reading this document, and you have a stimulator, please take a minute to write the license key on the back of the controller for future users - the stimulator is useless without it.

Running

When you first turn on the controller, we have found that you need to run a couple of test commands first, or you can't run long sets of commands. Best to try something like:

```
loadlibrary('C:\Users\at03\Desktop\StimLibV2-0.3.3-
Win\stimlib0.dll','C:\Users\at03\Documents\MATLAB\stimlibrel.h','alia
s','stimlib');
calllib('stimlib', 'initStimulator','M041F77IA1Y194777825MCH0BA3ABC')
calllib('stimlib', 'setDAC',0,0);
calllib('stimlib', 'setDAC',1,4000);
calllib('stimlib', 'setProperty','local_buffer_size',1000000);
%-----
calllib('stimlib', 'setPinBlock10',0,1,0,0,1,0,1,0,1,0,0,0);
calllib('stimlib', 'setPinBlock10',1,1,0,0,0,1,0,1,0,1,0,0);
calllib('stimlib', 'wait',1,2);
calllib('stimlib', 'setPinBlock10',0,1,0,0,1,0,1,0,1,0,0,0);
calllib('stimlib', 'setPinBlock10',1,1,0,0,0,1,0,1,0,1,0,0);
calllib('stimlib', 'startStimulation');
calllib('stimlib', 'closeStimulator');
```

Which moves the pins up and down once. You can then proceed as normal.

When we talk about 'controller commands' in this document, we refer to the commands that start `calllib('stimlib', ……. )`. It is these that are sent to the controller. To make the text a bit easier to read we will abbreviate commands like `calllib('stimlib', 'closeStimulator')` to `'closeStimulator'`.

The wait command

The timing is controlled by the `wait` command, which links the timing to the internal clock (see 'tests' section for tests on the accuracy of this clock). The wait command needs a minimum of 1ms between each call, otherwise it struggles (see 'Clock speed and the number of Braille-Devices' below to see caveats to this). As the controller is clocked at 2Hz, 1ms equals 2 cycles, ie:

```
calllib('stimlib', 'wait',1,2); % equal to one millisecond.
```

Stimulating sinusoids

Rectangular waves are fairly straightforward, and discussed in the manual. An example would be using the controller commands as follows:

```
loadlibrary('C:\Users\at03\Desktop\StimLibV2-0.3.3-
Win\stimlib0.dll','C:\Users\at03\Documents\MATLAB\stimlibrel.h','alia
s','stimlib');
calllib('stimlib', 'initStimulator',
'M041F77IA1Y194777825MCH0BA3ABC')

% ensure 2 DAC settings to the correct heights
calllib('stimlib', 'setDAC',0,0);
calllib('stimlib', 'setDAC',1,4000);
```

```
calllib('stimlib', 'setProperty','local_buffer_size',1000000);

%-----

for t=1:15

    % set the middle six pins to 'DAC1'
    calllib('stimlib', 'setPinBlock10',0,1,0,0,1,0,1,0,1,0,0,0);
    calllib('stimlib', 'setPinBlock10',1,1,0,0,0,1,0,1,0,1,0,0);

    % wait 2000 cycles (one second)
    calllib('stimlib', 'wait',1,2000);

    % set the middle six pins to 'DAC0'
    calllib('stimlib', 'setPinBlock10',0,1,0,0,0,0,0,0,0,0,0,0);
    calllib('stimlib', 'setPinBlock10',1,1,0,0,0,0,0,0,0,0,0,0);

    % wait 2000 cycles (one second)
    calllib('stimlib', 'wait',1,2000);


end

% ---- start simulation
calllib('stimlib', 'startStimulation');
calllib('stimlib', 'closeStimulator');
```

This would create a list of controller commands that would make a square wave for 30 seconds, changing height every second.

Creating a sinusoid follows the same principle, but is a little more difficult as the manual references how to do it only obliquely. We want to use a for loop to create each subsequent command, but for a 1000 Hz sampled signal, we need 1000 commands every second. An example of how to do this is in our appendix: we give examples of how to creating a sinusoid of any frequency, and how to read in in a wave from an external .mat file.

One of the most important commands is the `'remote_buffer_underruns'` command, which should be called after the `'closestimulator'` command.

```
%report how error-prone the result was (should be '0')
remote_buffer_underruns = calllib('stimlib',
'getProperty','remote_buffer_underruns')
```

Equivalent to Psychtoolbox's 'frames lost' function, it tells the user how many commands the controller failed to send. This number should be zero – if not, it means that the controller got some of the timings/heights wrong during your experiment.

Triggers

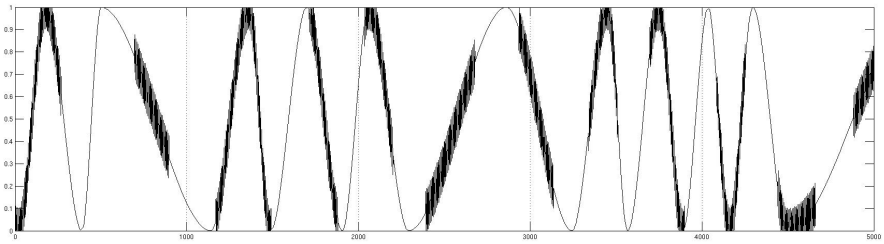
Triggers can be emitted from the control box at any particular point by using the necessary controller commands. See the appendix example of create a trigger every second during a 30 second sinusoid.

Controller commands can also wait for external triggers (for instance start vibrating when a button box is pressed) but we have not investigated these. Command

Testing

Overview

The stimulator was tested for the similarity of the intended pin movement (an arbitrary continuous signal from Matlab) to the actual movement of the pins. In the tests, the input was set as a signal of 30 second duration, sampled at 1000Hz. The stimulator can run at higher sample rates, but as the stimulator controller is clocked at 2000Hz and struggles to present signals using the voltage-altered-DAC method at this rate (see next entry), 1000Hz seemed like a good upper case limit. As a result, the input was a 30,000 (ie. 30x1000) array, with each element being a value between 0 and 4095 (the minimum and maximum 'extents' of the pins). For most testing purposes the signal was a 3hz sinusoid (or 25Hz), but it could have been anything. A number of things were checked for the accuracy of this stimulus. Due to the abstract nature of this particular stimulus, it is hoped that this will cover, or partially cover, most test cases.



Example 5-second height 'signal' to be sent to the braille pins (pseudo-sinusoidal contour with vibration overlaid), sample rate 1000Hz.

Clock speed and the number of Braille-Devices

The controller is clocked at 2000Hz. After testing, it was found that, when using a single braille-device, 1000Hz is the maximum sample rate that should be attempted – higher than this and the controller will struggle to run each command at the correct time. Unfortunately, the higher the number of braille devices, the lower this sample rate has to be, due to the increased processing load on the controller. Testing reveals that, for each additional braille device, an extra millisecond is needed to process the commands.

Number of braille devices	Minimum sample rate period (ms)	Maximum frequency (given the Nyquist rate)
1	1	500Hz
2	2	250Hz
3	3	166Hz
4	4	125Hz

In reality, the maximum frequency attempted should be a little lower that that advertised here to avoid undersampling.

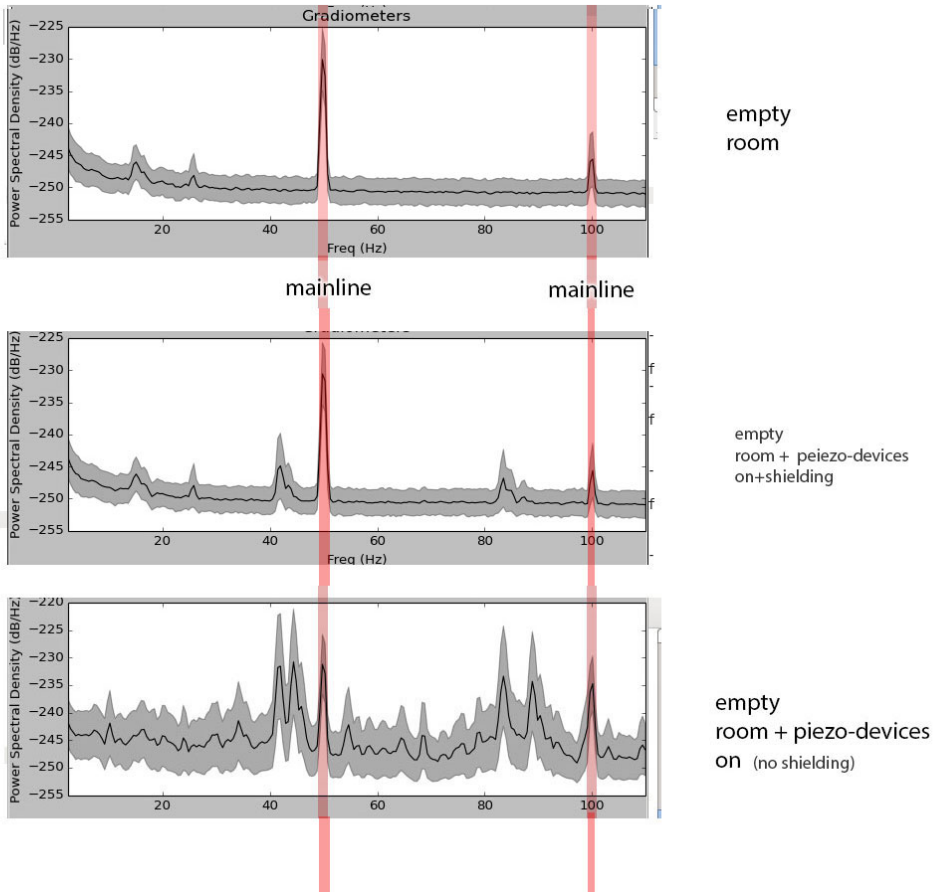
This means you cannot run a sine wave at (for instance) 150Hz through four braille-devices simultaneously.

Check latency

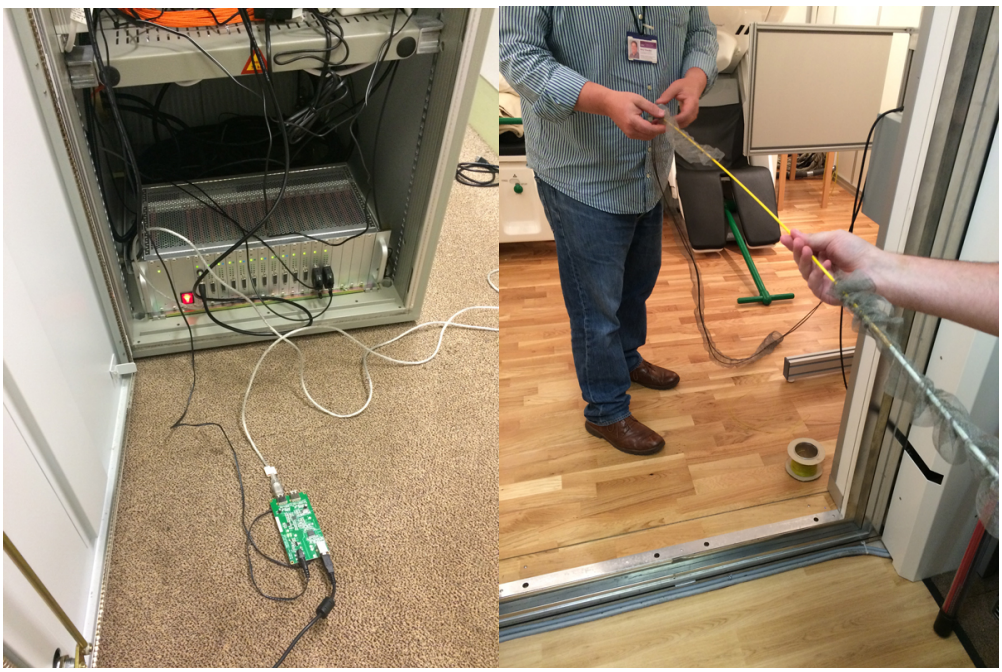
Unlike the visual or auditory setups, the latencies for the stimulator appear to be almost instantaneous relative to its internal clock.

Check EMEG compatibility

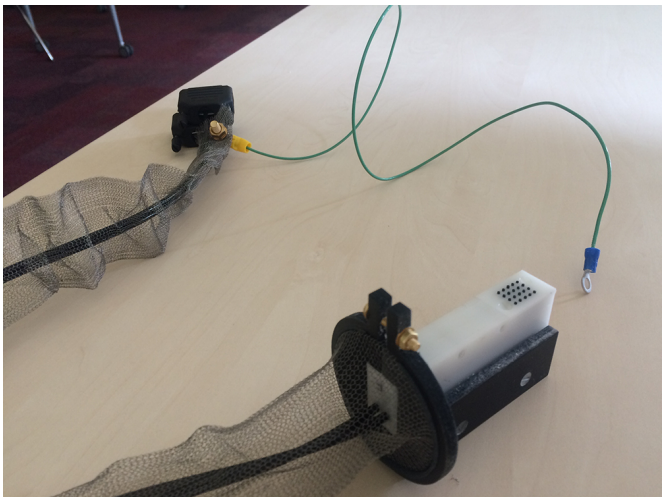
The QuaeroSys website claims the devices themselves are MEG compatible. We tested this in the CBUs MEG setup. We use a Neuromag Elekta MEG machine, held in a shielded room. We placed the control box outside the shielded room, threading the devices into the room to roughly hand-height for the participant.



We recorded an empty room recording with the devices running at 3Hz (top) and without (bottom). First, there is very, very little noise due to the sinusoidal running of the braille devices (but see later figure). Noise is clearly induced, however, from some other source. This seems to be due to the communication between the controller buffer and the stimulus computer, through contamination via the controller back-plate. For instance, if communication between the two is reduced (for instance by using a lower sampling rate) the latter part of the experiment will be noise free. However, this is not a suitable solution for most users, and attempts were made to remove this artifact without reducing the sample rates of the sinusoids. Optical USB filters (below) failed, but the addition of grounded Faraday shielding around the wiring (eg. 'RS components', stock number 7720290, EMC mesh cable wrap monel (50mm x 10m) MPN:KWM-1-5006) makes a clear difference (middle Power Spectral Density diagram above), although they can't mask the noise completely (see minor spikes of 42Hz and multiples).

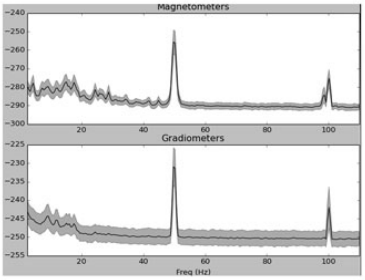


Left: an attempt to reduce communication noise using an optical filter on the input USB (no improvement); Right: threading Faraday shielding over the braille-device wiring (large improvement).

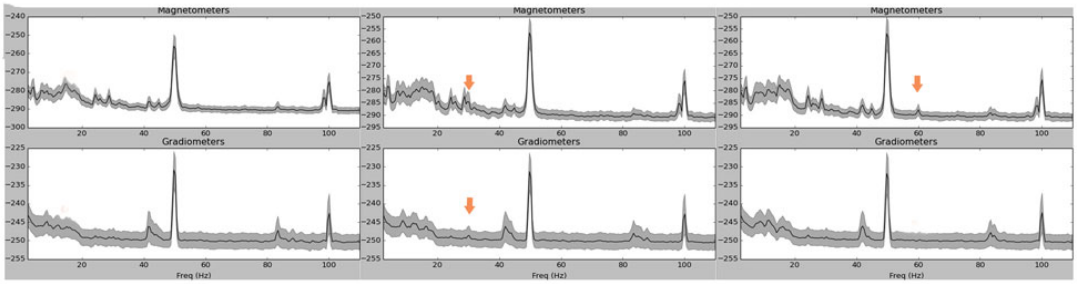


Above: Final setup for Faraday shielding over the braille-device wires, using 3D printed clamp on device end (please email mark.townsend@mrc-cbu.cam.ac.uk for file), and shielding grounding wire at D-type connector end.

We then did one final test to check the pins themselves were not generating magnetic noise, by sending 15Hz, 30Hz and 60Hz sine waves through all four braille devices to see if these frequencies turn up in in the recorded signal. These frequencies were observed - but so slightly that it only noticeable when comparing with the empty room, and not always in both magnetometers and gradiometers. For 15Hz it is not noticeable at all:



empty room



15Hz 30Hz 60Hz

Problems when running with HPI coils and EEG cap

Unfortunately, using EEG caps or HPI coils can amplify the reduced 'buffer' noise from the shielded braille device wires by acting as aerials. The resultant MEG noise (EEG remains unaffected) can be reduced by moving the braille devices away from the helmet, moving the EEG/HPI wires away from the devices. We did get usable data out of these recordings, even with the noise, but the resultant noise is comparable to the no-shielding-and-no-EEG-and-no-HPI setup.

Testing of internal clock

The stimulator has an internal clock, running at 2000Hz (roughly every 0.5ms). All its commands are run relative to this, using the 'wait()' command. We tested the accuracy of this internal clock by asking the stimulator to give out a trigger every 1000ms:

Trigger	EMEG Timestamp (seconds.ms)
1	26.865
2	27.865
3	28.865
4	29.865
5	30.865
6	31.865
7	32.864
8	33.864
9	34.864
10	35.864
11	36.864
12	37.864
13	38.863

14	39.863
15	40.863
16	41.863
17	42.863
18	43.863
19	44.862
20	45.862
21	46.862
22	47.862
23	48.862
24	49.862
25	50.862
26	51.861
27	52.861
28	53.861
29	54.861
30	55.861

This clock, of course, is here being checked against the clock of the recording computer – which might itself be wrong. Regardless of where the error is, the important point is that they *are* subtly different; for our setup the controller is slower, loosing a millisecond every 6000 or 7000 milliseconds. This means that, if you are sending a movement commands to the pins over, say, 5 minutes, it would be a bad idea to have a single trigger as the first command, and then to epoch your trials over the subsequent five minutes based on this initial trigger, as the movement of the braille-pins at the four-minute mark would be roughly 40ms out. Instead, it is best to insert short controller-internal out-triggers into the braille-device control commands throughout the stimulus run. This way, the 'drift' of the clock will never be far enough away from a trigger to interfere with your experiment. See appendix for an example of how to do this.

Testing of height

NOTE: The testing of heights using a setup of optical lasers was not carried out.

The pins occasionally stick. However, this is not due to the rod positions. The pins move upwards with the piezoelectric rods, but retract downward only by their own weight. Unfortunately they are very light, which means that a small amount of added friction (due to dust or some other source) can render them remaining in the upright position. For most experiments, this is not a problem – as the weight of the participant's finger (or whatever) will be easily enough to overcome this friction. We have not investigated ways to clean pins – but alcohol cleaning of those pins that show sign of sticking may improve things.

Memory buffer limit

You will need to ensure that enough memory is set aside on the stimulus computer to hold the commands that can't be sent to the controller's buffer to

begin with (the controller ('remote') buffer is not very big). As commands are run, these locally buffered commands will be fed through the USB to the controller buffer. It is trial and error as to whether you have set enough buffer, but the command cannot be sent if the buffer is not big enough, so just keep raising the size:

```
calllib('stimlib', 'setProperty','local_buffer_size',1000000);
```

until it all fits.

References

[1] The MathWorks, Inc., Natick, Massachusetts, United States.
[2] Brainard, D. H. (1997) The Psychophysics Toolbox, *Spatial Vision 10*:433-436; Pelli, D. G. (1997) The VideoToolbox software for visual psychophysics: Transforming numbers into movies, *Spatial Vision 10*:437-442; Kleiner M, Brainard D, Pelli D, 2007, "What's new in Psychtoolbox-3?" Perception 36 ECVP Supplement.
[3] https://github.com/svengijsen/StimulGL/tree/master/Source/Plugins/PiezoStimDevice/QuaeroSys
[4] Quarosys Medical Devices (2010)

Appendix

Example sinusoid code

This does *not* include a solution to initiate the controller with a couple of commands before the buffer will work (see 'running' above).

```
% Do NOT ask for the stimlib0.dll in system32 as suggested in the
manual -
% it will not find it, unless you have done this:
% http://superuser.com/questions/510986/how-to-know-why-system-cant-
find-dll
loadlibrary('C:\Users\at03\Desktop\StimLibV2-0.3.3-
Win\stimlib0.dll','C:\Users\at03\Documents\MATLAB\stimlibrel.h','alia
s','stimlib');

% Initiate the Stimulator - check the flag is 'OK'
% [NOTE: you will have to change the 30-digit licence key to your
own, as
% each stimulator has a unique liscence]
calllib('stimlib', 'initStimulator',
'M041F77IA1Y194777825MCH0BA3ABC')

% ensure all 8 DAC settings to 0 - not necessary but we do it anyway
calllib('stimlib', 'setDAC',0,0);
calllib('stimlib', 'setDAC',1,0);
calllib('stimlib', 'setDAC',2,0);
```

```
calllib('stimlib', 'setDAC',3,0);
calllib('stimlib', 'setDAC',4,0);
calllib('stimlib', 'setDAC',5,0);
calllib('stimlib', 'setDAC',6,0);
calllib('stimlib', 'setDAC',7,0);

calllib('stimlib', 'setProperty','local_buffer_size',1000000);

%-----
%You could import your own wave here (height needs to be between 1
and zero)...
%importedSignals = open('C:\Users\at03\Desktop\stimulisig.mat')

%leftindex=round(importedSignals.stimulisig.LHIndexFinger.*4000);
%rightindex=round(importedSignals.stimulisig.RHIndexFinger.*4000);
%duration = size(leftindex);

%-----
% Or create a sinusoid....
f=25; %in Hz - you only need to change this to get a different
sinisoid.
Fs=1000;
Ts=1/Fs;
secs = 30;
n=0:(secs*Fs);
x=sin(2*pi*f*n*Ts);
leftindex=round(((x+1)./2).*4000);
plot([0:Ts:1], leftindex(1:1001));
%-----

for t=1:30000 % for 30 seconds
    %-------examples:-----------
    %calllib('stimlib', 'setPinBlock10',slot,trigger,level 1 level 2
level 3 etc);
    %calllib('stimlib', 'wait',1,2);
    %---------------------------


    if (t==1 || t/1000 == floor(t/1000)) % this inserts a trigger
every second
        calllib('stimlib','triggerOut', 16, 1);
    end
    calllib('stimlib', 'wait',1,2); %2 = 1ms as it is clocked at 2KHz

    % set the voltage of 'DAC1' to produce the required height
    calllib('stimlib', 'setDAC',1,leftindex(t));

    % set the middle six pins to 'DAC1'
    calllib('stimlib', 'setPinBlock10',0,1,0,0,1,0,1,0,1,0,0,0);
    calllib('stimlib', 'setPinBlock10',1,1,0,0,0,1,0,1,0,1,0,0);

end


% check all fits in the local buffer
local_buffer_size = calllib('stimlib',
'getProperty','local_buffer_size')
local_buffer_status = calllib('stimlib',
'getProperty','local_buffer_status')
```

```
% startSimulation
calllib('stimlib', 'startStimulation');
calllib('stimlib', 'closeStimulator'); % this is the close for INIT!
NOT START!

%report how error-prone the result was (should be '0')
remote_buffer_underruns = calllib('stimlib',
'getProperty','remote_buffer_underruns')
```