Motor position control Aurora 309C using Spike2

Edwin D.H.M. Reuvers

Introduction

In this document, I will outline my software for controlling the Aurora 309C motor position using Spike2. The basic concept is straightforward: create a signal representing the motor position over time and instruct the software to output this signal to the DAC of the Power1401.

Please note that this document is not a quick help file to explain the final code of the program. Instead, the aim is to create an understanding of how the software works. Before moving on to the first step, I recommend reading the following items:

- To understand the control of the Aurora 309C, I recommend reading Chapter 4 (*General Operating Procedure*, pages 7-11) of the *Aurora 300C, 305C, 309C, 310C Manual*.
- To understand the Spike2 syntax, I recommend reading at least Chapters 1-3 of *The Spike2 Script Language*.
- To understand the configuration of Spike2, I recommend reading the Chapter *Sampling Data* (pages 15-18) of the *Spike2 Training Course Manual*.
- To understand Spike2 sequencer instructions, I recommend reading the chapter *Sampling, Control, and the Output Sequencer} (pages 36-46) of the* Spike2 Training Course Manual*.

A few important things to understand are:

- How length control and force control of the Aurora 309C work. Note that during length control, the maximum force is limited, and during force control, the maximum length is limited. It is very important to understand how this works; please see Chapter 4 (*General Operating Procedure*, pages 7-11) of the *Aurora 300C, 305C, 309C, 310C Manual*.
- Variables need to be defined in Spike2 before values can be assigned to them. Vector/-matrix sizes need to be defined in Spike2 before values can be assigned to these variables; the vector/matrix sizes cannot be changed afterward (as in MATLAB).

- Functions in Spike2 can only return a single number (i.e., a scalar). If you want to output a vector/matrix, you should declare this vector/matrix in your main script and then use this vector/matrix as input to the function. Variables declared in your main script have a global scope (for more information, see *The Spike2 Script Language*, pages 3-5), and therefore changing the values of the vector/matrix in your function will also change the value of the vector/matrix in the global scope (i.e., in your main script).
- In Spike2, variables ending with a % sign are integers, and variables ending with a $ sign are strings.
- The first index in Spike2 is 0. Therefore, if you want to access the first index of a variable, you should use `variablename[0]`.
- As far as I know, it is only possible to access variable values when debugging in Spike2. For instructions, see *The Spike2 Script Language*, pages 2-2 to 2-4. Alternatively, you can print your output to obtain the variable values (e.g., `PrintLog`).

Step 1: Create and output arbitrary waveform

Step 1A: Basic Script File

Below you can find the code of the script file (`Step1A_Script.s2s`) and the corresponding sequencer file (`Step1_SequencerFile.pls`) to output a sine wave to DAC0 with a certain frequency, amplitude, and offset. Note that this code only works while using the +/- 5V range of the Power1401. Please see Step 1B for an explanation. Let's go step-by-step through the script file. The sequencer file is self-explanatory (otherwise, see the Help section of Spike2).

Explanation of the Script File

- **Lines 2-5:** Here I declare the cycle frequency (`cf`), amplitude (`amp`), and offset (`lmotorAvg`) of the sine wave that I want to output. On the fifth line, I declare a variable `ncycles%` (an integer) which represents the number of sine waves that I want to output.
- **Line 7:** I declare the coefficient of the calibration factor. Here, I have used the standard value provided by the manufacturer for the Aurora 309C, but you should always verify this yourself.
- **Lines 10-12:** First, I declare the number of indices per second (`nstep%`). Second, I create a vector `nsamp%` that contains the number of indices of the signal we are going to output. Third, I create a time vector (`time`) and a vector for the motor position (`lmotor`; the signal we are going to output).
- **Line 15:** Here, I create the time vector, i.e., the vector increases by 1.0/nstep% for every index. As far as I know, Spike2 does not have a built-in function to create a time axis, so we use these commands (similar to `cumsum` in MATLAB).
- **Lines 18-22:** Spike2 does not have a predefined value of π, so we need to define π ourselves. Next, I assign the values to `lmotor` using a loop (I believe there is no way in Spike2 to directly fill a whole vector/matrix, so you should always use a loop).
- **Line 25:** Our output signal (`lmotor`) is in millimeters, but this should be converted to voltage. Hence, we need to divide `lmotor` by the calibration factor (which is in mm/V).
- **Lines 28-29:** With `PlayWaveAdd`, we assign the variable to `sampling configuration`. By doing so, we can later instruct the Power1401 to play this `waveform` (i.e., signal). Please see the Help section of Spike2 to understand all its inputs. With `PlayWaveCycles`, we change the number of cycles that this signal is being output (i.e., the number of repetitions of the output signal). After these two commands, your signal should be added to the `sampling configuration`. You can check this by going to **Sample > Sampling Configuration > Play Waveform**. Here you can check your key (first input of `PlayWaveAdd`; see below for explanation), label (second input of `PlayWaveAdd`; this is only a label for your reference), the DAC to which your signal is output, the number of indices per second (`Rate`), and the number of cycles.

- **Lines 32-33:** With `FileNew`, I open a new data file. Please see the Help section of Spike2 to understand both inputs. With `SampleStart`, I start sampling data.
- **Line 34:** With `SampleStatus()`, I check whether sampling has really started. We wait until sampling has started (then `SampleStatus()` returns 2; see the Help section of Spike2).
- **Line 35:** With `SampleKey`, I instruct Spike2 to perform sequencer instruction set “X” (hence, the motor should start to move). Please see below to understand the sequencer instructions sent to the Power1401.
- **Line 36:** With this line, we wait until the waveform created earlier in the script (“X”) has been played.
- **Lines 37-38:** With `SampleStop()`, I instruct Spike2 to stop sampling (i.e., we are now out of the while-loop and have sampled the desired amount of time). Then, I instruct it to save the data (a pop-up box will appear).

Listing 1: Code/Step1A_Script.s2s

```
1  ' User inputs
2  var cf := 1.0; ' [Hz] cycle frequency of sinusoidal motor position
3  var amp := 1.5; ' [mm] motor position amplitude
4  var lmotorAvg := 0.0; ' [mm] motor offset position
5  var ncycles% := 5; ' number of cycles
6
7  var coef := 1.25; ' [mm/V] relation between motor position output in millimeters and motor position output in voltage (i.e. 1 V equals 1.25 mm)
8
9  ' Initialize signal, time etc.
10 var nstep% := 1000; ' number of steps per second (i.e. dt = 1/nstep%)
11 var nsamp% := round(nstep%/cf); ' number of samples of our signal
12 var time[nsamp%], lmotor[nsamp%]; ' declare time and lmotor variable
13
14 ' Make time signal
15 ArrConst(time[], 1.0/nstep%); time[0] := 0; ArrIntgl(time[]);
16
17 ' Make motor length/position signal
18 var pi := Atan(1.0)*4;
19 var idx%;
20 for idx% := 0 to Len(lmotor[])-1 do
21     lmotor[idx%] := lmotorAvg+amp*Sin(time[idx%]*2*pi+0.5*pi);
22 next;
23
24 ' From motor position in millimeters to motor position in voltage
25 ArrDiv(lmotor,coef); ' From millimeters to voltage
26
27 ' Add to play waveform configuration
28 PlayWaveAdd("X", "SineWave", 0, nstep%, lmotor[]); '
29 PlayWaveCycles("X",ncycles%);
```

```

' Open new datafile and start sampling!
FileNew(0, 1); ' Open new datafile
SampleStart(0); ' Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (really) started
SampleKey("X");
while PlayWaveStatus$() = "X" do Yield() wend; ' Wait until waveform has been played
SampleStop(); ' Stop sampling
FileSave();
```

Listing 2: Code/Step1A_SequencerFile.pls

```
            SET      0.1,1,0          ; Run at 10 kHz (0.1 ms per sequencer step) and 5V range of Power1401
            HALT

; Play waveform X
AI:     'X  WAVEGO X                  ; Play waveform X
AE:         HALT
```

Step 1B: Bring Motor to Starting Position & Script for 5V and 10V Range

With the code presented in Step 1A, the output signal of the motor position instantaneously changes to the first value of the motor position signal (`lmotor`) when we start sampling. This might cause high acceleration to the motor position/muscle, which is undesirable (e.g., it might cause damage to the muscle). To prevent this, I have added code to slowly bring the motor to its starting position before sampling starts. Additionally, I have added code to ensure it works for both the +/- 5V and +/- 10V range of the Power1401. Let's have a look at the new script file (`Step1B_Script.s2s`) and the corresponding sequencer file.

Explanation of the Script File

- **Lines 28-34:** Sequencer instructions to the Power1401 are in bits. For some sequencer instructions, 16-bits are used, while 32-bits are used for other instructions (see Spike2 Help section: Output sequencer > Instructions > Variables). Hence, for 16-bit sequencer instructions, -2^{15} bits represent the lower bound of the range (i.e., either -5V or -10V) and 2^{15} bits represent the upper bound of the range (i.e., either 5V or 10V). As an example, to set the motor voltage to 2.5V while using a 5V range, the DAC should be set to $2^{15}/2$, while in the 10V range, the DAC should be set to $2^{15}/4$. The function PlayWaveAdd automatically converts the motor voltage to the number of bits and assumes that the range of the Power1401 is set to 5V (see the Help section of Spike2). Therefore, the amplitude should be divided by 2 to obtain the correct amplitude when operating in the 10V range. These lines determine the range of the Power1401 in use at the moment, and the range is assigned to the variable `DACscale`.

- **Lines 41-50:** Before executing the desired motion of the motor, the motor should first be brought slowly to its starting position. The motor position can only be changed when data is being sampled, so a new data file must be opened, and sampling must be started. Then, I use the RAMP sequencer instruction to bring the motor slowly to its starting position. The second input of this sequencer instruction is the motor position at the end of the ramp in 32-bits. Therefore, I convert the starting position of the motor to 32-bits on line 46 and assign this variable to no. 2. Note that in theory, we should multiply the starting position by $2^{31}/DACscale$, but on line 34, the motor starting position is already adjusted to a 5V range. Hence, the motor starting position should be multiplied by $2^{31}/5$. Lastly, I explicitly give an initial instruction to the sequencer to set DAC0 to the current motor voltage (see line 45 of the script and line 9 of the sequencer file). This is necessary because I experienced that the motor voltage quickly drops to 0 before ramping up when not doing so.

Listing 3: code/Step1B_Script.s2s

```
1  ' User inputs
2  var cf := 1.0; ' [Hz] cycle frequency of sinusoidal motor position
3  var amp := 1.5; ' [mm] motor position amplitude
4  var lmotorAvg := 0.0; ' [mm] motor offset position
5  var ncycles% := 5; ' number of cycles
6
7  var coef := 1.25; ' [mm/V] relation between motor position output in
       millimeters and motor position output in voltage (i.e. 1 V equals 1.25
       mm)
8
9  ' Initialize signal, time etc.
10 var nstep% := 1000; ' number of steps per second (i.e. dt = 1/nstep%)
11 var nsamp% := round(nstep%/cf); ' number of samples of our signal
12 var time[nsamp%], lmotor[nsamp%]; ' declare time and lmotor variable
13
14 ' Make time signal
15 ArrConst(time[], 1.0/nstep%); time[0] := 0; ArrIntgl(time[]);
16
17 ' Make motor length/position signal
18 var pi := Atan(1.0)*4;
19 var idx%;
20 for idx% := 0 to Len(lmotor[])-1 do
21     lmotor[idx%] := lmotorAvg+amp*Sin(cf*time[idx%]*2*pi+0.5*pi);
22 next;
23
24 ' From motor position in millimeters to motor position in voltage
```

```
ArrDiv(lmotor,coef); ' From millimeters to voltage

' Get DACscale an adjust motor position amplitude if necessary
var n% := 1, response%;
U1401Open(n%);
U1401Write(Print$("GAIN,M, 1;"));
U1401Read(response%); ' Get the result
U1401Close();
DACscale := response%/1000;
ArrDiv(lmotor,DACscale/5); ' Adjust motor position amplitude

' Add to play waveform configuration
PlayWaveAdd("X", "SineWave", 0, nstep%, lmotor[]); '
PlayWaveCycles("X",ncycles%);

' Bring motor to start position
FileNew(0, 1); ' Open a new datafile
SampleStart(0); Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (
    really) started
var VPre := ChanValue(1,0);
SampleSeqVar(1, Vpre*Pow(2,32)/(2*5));
SampleSeqVar(2, lmotor[0]*Pow(2,32)/(2*5));
SampleKey("I");
Yield(abs(Vpre-lmotor[0])/0.2+1); ' Keep changing the motor position until
     we reach the starting position
SampleStop(); ' Stop sampling
FileClose(-1,-1);  ' Close the file and do not save it

' Open new datafile and start sampling!
FileNew(0, 1); ' Open new datafile
SampleStart(0); ' Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (
    really) started
SampleKey("X");
while PlayWaveStatus$() = "X" do Yield() wend; ' Wait until waveform has
    been played
SampleStop(); ' Stop sampling
FileSave();
```

Listing 4: code/Step1B_SequencerFile.pls

```
                SET      0.1,1,0          ; Run at 10 kHz (0.1 ms per
                    sequencer step) and 5V range of Power1401
                HALT

; Play waveform X
AI:        'X   WAVEGO X                   ; Play waveform X
```

```
AE:          HALT

; Set motor to initial (I) length. For this we will slowly ramp.
IA:      'I  DAC     0,V1
             RAMP    0,V2,1.0/S(5)    ; Ramp up with 0.2 V/s
IRD:         WAITC   0,IRD            ; Wait until ramp is finished
             HALT
```

Step 1C: Clean-up script

The software to control the Aurora 309C motor position works perfectly fine now, both when operating in +/- 5V and +/-10 V. I prefer now to clean-up the script a little bit, and therefore I made some functions to make the motor position signal, to correct the motor position signal depending on which range we are operating and to bring the motor to its starting position. When doing so, the script file looks as follows (and the sequencer file is similar to as in Step 1B).

Listing 5: code/Step1C_Script.s2s

```
#include "Step1C_Script_Functions.s2s" ' Load functions

' User inputs
var cf := 1.0; ' [Hz] cycle frequency of sinusoidal motor position
var amp := 1.5; ' [mm] motor position amplitude
var lmotorAvg := 0.0; ' [mm] motor offset position
var ncycles% := 5; ' number of cycles

var coef := 1.25; ' [mm/V] relation between motor position output in millimeters and motor position output in voltage (i.e. 1 V equals 1.25 mm)

' Initialize signal, time etc.
var nstep% := 1000; ' number of steps per second (i.e. dt = 1/nstep%)
var nsamp% := round(nstep%/cf); ' number of samples of our signal
var time[nsamp%], lmotor[nsamp%]; ' declare time and lmotor variable

' Make time & motor position signal
ArrConst(time[], 1.0/nstep%); time[0] := 0; ArrIntgl(time[]);
MakeSineWave(lmotor[],time[],cf,amp,lmotorAvg);

' From motor position in millimeters to motor position in voltage
ArrDiv(lmotor,coef); ' From millimeters to voltage
CorrectSignal(lmotor); ' Correct motor position signal depending on range

' Add to play waveform configuration
PlayWaveAdd("X", "SineWave", 0, nstep%, lmotor[]); '
```

```
PlayWaveCycles("X",ncycles%);

' Open new datafile and start sampling!
BringToStartPosition(); ' Bring motor to starting position
FileNew(0, 1); ' Open new datafile
SampleStart(0); ' Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (really) started
SampleKey("X");
while PlayWaveStatus$() = "X" do Yield() wend; ' Wait until waveform has been played
SampleStop(); ' Stop sampling
FileSave();
```

Listing 6: code/Step1C_Functions.s2s

```
func MakeSineWave(lmotor[],time[],cf,amp,lmotorAvg)
' MakeSineWave makes a sine trajectory for the motor position over time.

' Inputs:
    ' time      = the time-axis [s]
    ' lmotor    = an empty array in which the motor position over time will be placed [mm]
    ' cf        = the cycle frequency (of the motor movement) [Hz]
    ' amp       = the amplitude (of the motor movement) [mm]
    ' lmotorAvg = the average motor position (i.e. at sin(-0.5*pi)) [mm]
' Outputs:
    ' lmotor    = the motor position over time [mm]

    var pi := Atan(1.0)*4;
    var idx%, tc;
    for idx%:=0 to Len(sig[])-1 do
        lmotor[idx%] := lmotorAvg+amp*Sin(cf*time[idx%]*2*pi+0.5*pi);
    next;
return 1;
end;

func CorrectSignal(lmotor[])
' CorrectSignal correct the signal that will be 'played' by PlayWaveAdd based on whether the DAC is in 5V or 10V mode.

    var n%:=1, Response%;

    U1401Open(n%);
    U1401Write(Print$("GAIN,M, 1;"));
    U1401Read(Response%); 'Get the result.
    U1401Close();

    DACscale := Response%/1000;
```

```
    ArrDiv(lmotor,DACscale/5); ' Adjust motor position amplitude
return 1;
end;

func BringToStartPosition()
' BringToStartPosition this function bring the motor to the initial
   position (the motor position at the start (t=0))

    FileNew(0, 1); ' Open a new datafile
    SampleStart(0); ' Start sampling
    while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (
       really) started
    var VPre := ChanValue(1,0);
    SampleSeqVar(1, Vpre*Pow(2,32)/(2*5));
    SampleSeqVar(2, lmotor[0]*Pow(2,32)/(2*5));
    SampleKey("I");
    Yield(abs(Vpre-lmotor[0])/0.2+1); ' Keep changing the motor position
       until we reach the starting position
    SampleStop(); ' Stop sampling
    FileClose(-1,-1);  ' Close the file and do not save it
return 1;
end;
```

Step 2: Stimulation

To stimulate the nerve of the muscle, I used a direct current stimulus isolator (Digitimer Limited, Hertfordshire, UK, model DS3). The DS3 was set to deliver a pulse when the input value changes from 0 to 1. Therefore, to stimulate the muscle from t=0.1 to t=0.3 with a frequency of 100 Hz, we need to provide the input signal to the DS3 as shown in Figure Figure 1}.

I chose the duration of the blocks (i.e., the time at which the signal equals 1) so that half of the time the signal equals 1 and the other half the signal equals 0. In theory, the duration of the block can be shorter or longer, as long as the signal returns to 0 before switching to 1 again.

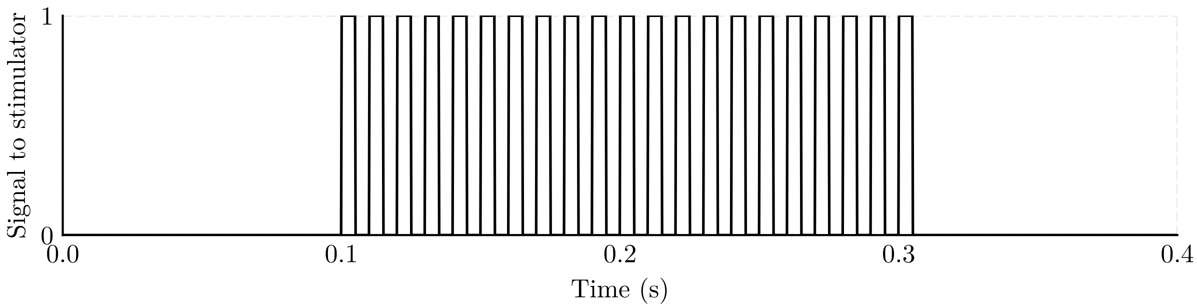

Figure 1: The signal to be sent to the DS3 stimulus isolator to stimulate the nerve from t=0.1 to t=0.3 at a frequency of 100 Hz.

Step 2A: A single burst of nerve stimulation

We can do this with the following script file (`Step2A\_Script.s2s`), functions file (`Step2A\_Functions.s2s`) and sequencer file (`Step2A\_SequencerFile.pls`).

Listing 7: Code/Step2A_Script.s2s

```
#include "Step2A_Script_Functions.s2s" ' Load functions

' User inputs - stimulation
var tStimOn := -0.04; ' [s] time of muscle stimulation onset
var tStimOff := 0.37; ' [s] time of muscle stimulation offset
var stimCF := 250.0; ' [Hz] muscle stimulation frequency

' Initialize signal, time etc.
var nstep% := 10000; ' number of steps per second (i.e. dt = 1/nstep%)
var nsamp% := round(nstep%/cf); ' number of samples of our signal
var time[nsamp%], stim[nsamp%]; ' declare time and stim variable

' Make time & motor position signal
```

```
ArrConst(time[], 1.0/nstep%); time[0] := 0; ArrIntgl(time[]); ' [s] time-axis
MakeStimulation(time,stim,tStimOn,tStimOff,stimCF,StimOnCycle[]);

' Make signal sent to both DAC outputs
CorrectSignal(stim[]); ' Correct motor position signal depending on DAC range

' Add to play waveform configuration
PlayWaveAdd("X", "SineWave", 1, nstep%, stim[]); '
PlayWaveCycles("X",1);

' Open new datafile and start sampling!
FileNew(0, 1); ' Open new datafile
SampleStart(0); ' Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (really) started
SampleKey("X");
while PlayWaveStatus$() = "X" do Yield() wend; ' Wait until waveform has been played
SampleStop(); ' Stop sampling
FileSave();

' Functions
func MakeStimulation(time[],stim[],tStimOn,tStimOff,stimCF,StimOnCycle[])
' MakeStimStimulation calculates the stimulation over time.

' Inputs:
    ' time      = the time-axis [s]
    ' stim      = an empty array in which the stimulation over time will be placed
    ' tStimOn   = time onset of stimulation [s]
    ' tStimOff  = time offset of stimulation [s]
    ' stimCF    = stimulation frequency [Hz]
' Outputs:
    ' stim      = the stimulation over time


    var tBlock := 1.0/stimCF; ' [s] duration between two stimulation pulses
    ArrConst(stim[], 0.0);
    var idx%, jdx%;
    for idx%:=0 to Len(time[])-1 do
        if time[idx%]>=tStimOn+jdx%/cf and time[idx%]<=tStimOff+jdx%/cf+0.5*tBlock then
            if ((time[idx%]-tStimOn) mod tBlock) <= tBlock/2 then
                stim[idx%] := 10;
            else
```

```
                stim[idx%] := 0;
            endif;
        endif;
    next;
return 1;
end;

func CorrectSignal(signal[][])
' CorrectSignal correct the signal that will be 'played' by PlayWaveAdd
    based on whether the DAC is in 5V or 10V mode.

' Inputs:
    ' signal    = the signal that will be corrected [V]
' Outputs:
    ' signal    = the corrected signal [V]

    var n%:=1, Response%;

    U1401Open(n%);
    U1401Write(Print$("GAIN,M, 1;"));
    U1401Read(Response%); 'Get the result.
    U1401Close();

    var DACscale := Response%/1000;
    ArrDiv(signal,DACscale/5); ' Adjust motor position amplitude
return 1;
end;
```

Listing 8: Code/Step2A_Functions.s2s

```
' Functions
func MakeStimulation(time[],stim[],tStimOn,tStimOff,stimCF,StimOnCycle[])
' MakeStimStimulation calculates the stimulation over time.

' Inputs:
    ' time      = the time-axis [s]
    ' stim      = an empty array in which the stimulation over time will
        be placed
    ' tStimOn   = time onset of stimulation [s]
    ' tStimOff  = time offset of stimulation [s]
    ' stimCF    = stimulation frequency [Hz]
' Outputs:
    ' stim      = the stimulation over time


    var tBlock := 1.0/stimCF; ' [s] duration between two stimulation
        pulses
    ArrConst(stim[], 0.0);
    var idx%, jdx%;
```

```
    for idx%:=0 to Len(time[])-1 do
        if time[idx%]>=tStimOn+jdx%/cf and time[idx%]<=tStimOff+jdx%/cf+0.5*tBlock then
            if ((time[idx%]-tStimOn) mod tBlock) <= tBlock/2 then
                stim[idx%] := 10;
            else
                stim[idx%] := 0;
            endif;
        endif;
    next;
return 1;
end;

func CorrectSignal(signal[][])
' CorrectSignal correct the signal that will be 'played' by PlayWaveAdd based on whether the DAC is in 5V or 10V mode.

' Inputs:
    ' signal    = the signal that will be corrected [V]
' Outputs:
    ' signal    = the corrected signal [V]

    var n%:=1, Response%;

    U1401Open(n%);
    U1401Write(Print$("GAIN,M, 1;"));
    U1401Read(Response%); 'Get the result.
    U1401Close();

    var DACscale := Response%/1000;
    ArrDiv(signal,DACscale/5); ' Adjust motor position amplitude
return 1;
end;
```

Step 2B: Nerve stimulation during different cycles

The last step is (1) to combine imposing motor motion with imposing nerve stimulation and (2) to impose nerve stimulation during different (stretch-shortening) cycles. For examples, when the motor position is changing with a sinusoidal motion (for five cycles), we want to impose nerve stimulation during the second, third and fourth cycle, but not during the first and fifth cycles. The problem here is that we cannot simply instruct the Power1401 to repeat the signal to DAC0 (motor position) and DAC1 (nerve stimulation) to repeat the signal of one cycles five times. Hence, we need to make a signal to DAC0 and DAC1 such that it is the entire time series (i.e. of all cycles). We can do this as follows:

Listing 9: Code/Step2B_Script.s2s

```
#include "Step2B_Script_Functions.s2s" ' Load functions

' User inputs - motor motion
var cf := 1.0; ' [Hz] cycle frequency of sinusoidal motor position
var amp := 2.5; ' [mm] motor position amplitude
var lmotorAvg := 0.0; ' [mm] motor offset position

' User inputs - stimulation
var tStimOn := -0.04; ' [s] time of muscle stimulation onset (relative to the time of start of one cycle)
var tStimOff := 0.37; ' [s] time of muscle stimulation offset (relative to the time of start of one cycle)
var stimCF := 250; ' [Hz] muscle stimulation frequency
var StimOnCycle[5];
StimOnCycle[0] := 0; StimOnCycle[1] := 1; StimOnCycle[2] := 1;
StimOnCycle[3] := 1; StimOnCycle[4] := 0; ' 1 indicates that muscle stimulation should be ON, and 0 indicates OFF (for every cycle)

' User inputs - calibration
var coef := 1.25; ' [mm/V] relation between motor position output in millimeters and motor position output in voltage (i.e. 1 V equals 1.25 mm)

' Initialize signal, time etc.
var ncycles% := Len(StimOnCycle); ' number of cycles
var nstep% := 10000; ' number of steps per second (i.e. dt = 1/nstep%)
var nsamp% := round(nstep%/cf*ncycles%); ' number of samples of our signal
var time[nsamp%], lmotor[nsamp%], stim[nsamp%], sig[nsamp%][2]; ' declare time and lmotor variable

' Make time & motor position signal
ArrConst(time[], 1.0/nstep%); time[0] := 0; ArrIntgl(time[]); ' [s] time-axis
MakeSineWave(time,lmotor,cf,amp,lmotorAvg);
MakeStimulation(time,stim,tStimOn,tStimOff,stimCF,StimOnCycle[]);
ArrDiv(lmotor,coef); ' From motor position in millimeters to motor position in voltage

' Make signal sent to both DAC outputs
ArrAdd(sig[][0],lmotor[]);
ArrAdd(sig[][1],stim[]);
CorrectSignal(sig[][]); ' Correct motor position signal depending on DAC range

' Add to play waveform configuration
var dacs%[2]; dacs%[0] := 0; dacs%[1] := 1;  'list of dacs
PlayWaveAdd("X", "SineWave", dacs%, nstep%, sig[][]); '
```

```
PlayWaveCycles("X",1);

' Open new datafile and start sampling!
BringToStartPosition(sig[][]); ' Bring motor to starting position
FileNew(0, 1); ' Open new datafile
SampleStart(0); ' Start sampling
while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (really) started
SampleKey("X");
while PlayWaveStatus$() = "X" do Yield() wend; ' Wait until waveform has been played
SampleStop(); ' Stop sampling
FileSave();
```

Listing 10: Code/Step2B_Functions.s2s

```
' Functions
func MakeStimulation(time[],stim[],tStimOn,tStimOff,stimCF,StimOnCycle[])
' MakeStimStimulation calculates the stimulation over time.

' Inputs:
    ' time      = the time-axis [s]
    ' stim      = an empty array in which the stimulation over time will be placed
    ' tStimOn   = time onset of stimulation [s]
    ' tStimOff  = time offset of stimulation [s]
    ' stimCF    = stimulation frequency [Hz]
' Outputs:
    ' stim      = the stimulation over time


    var tBlock := 1.0/stimCF; ' [s] duration between two stimulation pulses
    ArrConst(stim[], 0.0);
    var idx%, jdx%;
    for jdx%:=0 to Len(StimOnCycle[])-1 do
        for idx%:=0 to Len(time[])-1 do
            if time[idx%]>=tStimOn+jdx%/cf and time[idx%]<=tStimOff+jdx%/cf+0.5*tBlock then
                if ((time[idx%]-tStimOn) mod tBlock) <= tBlock/2 then
                    stim[idx%] := 10*StimOnCycle[jdx%];
                else
                    stim[idx%] := 0;
                endif;
            endif;
        next;
    next;
return 1;
end;
```

```

func CorrectSignal(signal[][])
' CorrectSignal correct the signal that will be 'played' by PlayWaveAdd
    based on whether the DAC is in 5V or 10V mode.

' Inputs:
    ' signal    = the signal that will be corrected [V]
' Outputs:
    ' signal    = the corrected signal [V]

    var n%:=1, Response%;

    U1401Open(n%);
    U1401Write(Print$("GAIN,M, 1;"));
    U1401Read(Response%); 'Get the result.
    U1401Close();

    var DACscale := Response%/1000;
    ArrDiv(signal,DACscale/5); ' Adjust motor position amplitude
return 1;
end;

func BringToStartPosition(sig[][])
' BringToStartPosition this function bring the motor to the initial
    position (the motor position at the start (t=0))

    FileNew(0, 1); ' Open a new datafile
    SampleStart(0); ' Start sampling
    Yield(1);
    while SampleStatus() <> 2 do Yield() wend; ' Wait until samples has (
        really) started
    var VPre := ChanValue(2,0);
    SampleSeqVar(1, Vpre*Pow(2,32)/(2*5));
    SampleSeqVar(2, sig[0][0]*Pow(2,32)/(2*5));
    SampleKey("I");
    Yield(abs(Vpre-sig[0][0])/0.1+1); ' Keep changing the motor position
        until we reach the starting position
    SampleStop(); ' Stop sampling
    FileClose(-1,-1);  ' Close the file and do not save it
return 1;
end;

func MakeSineWave(time[],lmotor[],cf,amp,lmotorAvg)
' MakeSineWave makes a sine trajectory for the motor position over time.

' Inputs:
    ' time      = the time-axis [s]
    ' lmotor    = an empty array in which the motor position over time
```

```
        will be placed [mm]
    ' cf        = the cycle frequency (of the motor movement) [Hz]
    ' amp       = the amplitude (of the motor movement) [mm]
    ' lmotorAvg = the average motor position (i.e. at sin(-0.5*pi)) [mm]
' Outputs:
    ' lmotor    = the motor position over time [mm]

    var pi := Atan(1.0)*4, idx%;
    for idx%:=0 to Len(time[])-1 do
        lmotor[idx%] := lmotorAvg+amp*Sin(cf*time[idx%]*2*pi+0.5*pi);
    next;
return 1;
end;
```

Step 3: Final notes

At this moment, it is hopefully clear how you can 'easily' make your own program in Spike2 to control the motor position and nerve stimulation. Based on the presented code, you can now alter the motor position by 'only' creating a different variable `lmotor[]`; which you can do by 'simply' editing the function MakeSineWave or making a new function yourself.

Future ideas for the presented program are to use a GUI such that the user can set the parameters depending on which motor/muscle motion is used. By then making different function for, for example, quick-release protocols, step-ramp protocols etc., the user had only to set the right parameters and the muscle motion position signal (variable: `lmotor[]`) will be automatically created and send to the Power1401.