BCI practical course : Advanced stimulus presentation

Jason Farquhar (coordinator) <J.Farquhar@donders.ru.nl>
Loukianos Spyrou (Teaching Assistant) <l.spyrou@donders.ru.nl>

Learning Goals : Advanced Stimilus Presentation

- Understand:
  - why precise stimulus timing requirements mean we need to run the stimulus generation in a separate process and use more accurate stimulus presentation methods
- Know how to:
  - Use Psychtoolbox for more precise auditory/visual stimulus generation and timing

Today's Plan

- Review : Visual Speller Example. Solutions and discussion of problems
- Hands-on : Psychtoolbox based visual speller stimulus

Break

- Test run: 'DCC visitation presentation'

break

- Discussion: Possible projects

Hands on : Visual Matrix Speller

Experiment Task:

- Build a complete visual matrix speller based BCI experiment consisting of 3 blocks:
  - 1) Training/Calibration Block : where the user is presented with matrix speller stimuli and an instruction on which target to attend to
  - 2) Classifier Training Blocks : where the saved labelled data from the calibration block is used to train an ERP classifier
  - 3) Testing Block : where the trained classifier is used predict which symbol the user is attending to and at the end of the sequence this prediction is used to generate feedback

Discussion – Timing

- Timing issues due to:
  - Non-interruptable/single-thread nature of MATLAB
  - Slow matlab based drawing command
  - Variable length matlab based drawing commands
  - Lack of information about exactly when stimulus was presented (just start/end time of `drawnow` command)
  - Once function starts executing it blocks CPU until it's finished..

Solution:

- Use something designed for high speed, high precision stimulus presentation, e.g. Vision Egg (Python) or PyGame (Python) or Presentation etc.
- Psychtoolbox – a MATLAB/Octave (mex/OpenGL/PortAudio)

Hands on: Psychtoolbox

- Matlab's basic drawing commands are, both
  - slow (~70ms to execute) and
  - jittery (+/- 30ms execution time)
- Psychtoolbox (see www.psychtoolbox.org) allows to direct control of the video/audio hardware
  - Very fast (GPU accelerated) drawing (<2ms)
  - Very precise – locked to video hardware (<2ms)

Hands on : Psychtoolbox (PTB)

Task:

- Re-write your visual speller BCI from last week to use Psychtoolbox drawing functions
- Compare the timing performance of this version with the Matlab based version

Notes: System architecture

- Again we have a decision:
  1. Keep the drawing code in the experimental control process
     - Adv: simple to implement
     - Dis: less precise, lots other matlab happening
  2. Move the drawing into a new stimulus generation process
     - Adv: more precise timing (less other stuff going on, just draw & send events)
     - Dis: more complex

Key Functions : Psychtoolbox

- Note you can get comprensive help on using PTB functions from there documentation website
    - `http://docs.psychtoolbox.org/Psychtoolbox`
- Or, by using normal matlab help
    - `help Snd`
- Or, by using the functions builtin help, adding a '?' to the command
    - `Screen flip?`

Key Functions : Psychtoolbox

- `Screen`
  - All PTB drawing commands use this function with it's first argument specifying what exactly to do
  - `Screen` gives a list of subfunctions,
  - `Screen subfunction?` Gives help on subfunction
- Open a PTB window at `pos` with `bgColor` background
  - `wPtr= Screen('OpenWindow',num,bgColor,rect)`
- Update the display after drawing finished:
  - `screen('Flip',wPtr,when);`
  - If when==0 then wait until next refresh, otherwise wait until when is the system time
  - N.B. Screen only changes after a Flip command
- Texture creation and manipulation commands... see next.
- N.B. Add the Psychtoolbox functions to your MATLAB path using:
  - `run ../utilities/initPTBPaths`

Key Concepts : textures

Pyschtoolbox uses hardware based textures for very fast drawing.

- A texture is an image which is pre-loaded onto the graphics hardware using:
  - `textureID=Screen('MakeTexture',wPtr,image)`
  - `wPtr` is handle to the PTB drawing window
  - `image` is a [w x h x 3/4] image matrix
  - `texelID` is a handle to the created texture
- A texture can then be drawn to the screen rapidly using:
  - `Screen('DrawTextures',wPtr,texelID,srcR,destR,angle,filt,alpha,color)`
  - `srcR/destR` specify rectangles [left top right bottom] (ltbr) of the source image to draw at dest position on the screen
- Importantly: at drawing time the texture can be manipulated extreemly rapidly using the other arguments of 'DrawTextures', e.g.
  - position/size, rotation angle, color, transparancey, etc.
- N.B. The screen is only actually changed when Flip is called!

useful function: mkTextureGrid

PTB equivalent of initGrid to make a grid of textures:

```
[texs,srcRs,destRs]=mkTextureGrid(wPtr,symbs)
```

- `wPtr` – PTB window handle (as returned by Screen('OpenWindow',..))
- `symbs` – cell array of images or text to layout. The shape of Symbs gives the shape of the grid
- `texs` – handles to the textures
- `srcRs/destRs` source/destination rectangles (used in DrawTextures)
- N.B. Draw the textures using `DrawTextures`, e.g.:

```
Screen('DrawTextures',wPtr,texs,srcRs,destRs,[],[],[],
[255;255;255]*[0 0 0 1 1 1]);
```

  - Makes the first 3 textures **black** (with color [0 0 0]) and the last 3 white (with color [255 255 255]) by changing the color scaling , i.e. Drawn color= orginalColor .* drawColorScale
  - Note: in PTB all colors are in [rgb[a]] with a range from 0-255

Summary

- Using PTB (or other dedicated stimulus generation software) means we can get much better stimulus timing accuracy
- BUT Matlab based is simplier, less code / dependencies
  - And for many, e.g. Movement based, improved timing accuracy has no benefit
- So: think about what you need before you start!