**Dan Stolzberg’s ElectroPsychology**

(Online: files at Github.com as dstolz/epsych. We’ve forked as rosenlab/epsych. rosenlab:gerbil123

RZ6 does sound and behavior control

RZ5 does neurophysiology

Control this in OpenEx, with separate RPVds circuits for each device.

We’ve set up a startup.m file to automatically be run every time Matlab starts, which sets paths for ElectroPsych Toolbox. This is startup.m in C:\Users\LabUser\Documents\MATLAB\ - it adds our epsych path, then runs C:\Electropsych\epsych\epsych\_startup.m.

Components necessary to create a paradigm:

* Create **RPvds** .rcx file (TDT circuit file) – heart of the behavioral paradigm
* Create Matlab script as a Trial-Select Function. Initialize pump here and select trial types.
* optional: custom Matlab scripts to generate necessary output (e.g., Viemeister.m is the input used to modulate AM depth while maintaining RMS within the AversiveAMTest.rcx circuit)
* Use **ep\_ExperimentDesign.m** to create a .prot file – will open GUI.
  + Select **TDT module/device** (RZ6)
  + Associate an **RPvds** .rcx file; it will auto-locate all parameter tags in the circuit.
    - Use \* before parameter name so parameter won’t get sent to TDT devices.
  + Under Options, check “Operational Trigger” if this is for behavior (as opposed to passive neurophys)
  + Enter values for each parameter. These can be values, any matlab code that evaluates on a single line, or a matlab script.
  + Buddy up parameters to create specific trial types where certain parameters are matched with each other
    - e.g., buddy NoiseDur, ToneDelay, and ToneDur to create SAFE and WARN trials
  + Within Trial-Select Function, point to **Matlab trial-choosing script**
    - Type name of matlab script, excluding the path and the .m file extension.
    - How does the .prot file know the path?
      * The .prot file created using the Experiment Design utility handles this in two ways:

1. Using in conjunction with OpenEx: The module names in the Experiment Design utility must match the module names in OpenEx Workspace program. Epsych uses these module names to ensure parameters are updated on the appropriate module.
2. Using without OpenEx: The Experiment Design utility associates an RPvds file with each module as it is created.
   * + If none chosen, uses DefaultTrialSelectionFcn.m
     + C:\Electropsych\epsych\runtime\trial\_selection\: TrialFcn\_Aversive.m or TrialFcn\_AversiveAnimalTrig.m
     + Initiates pump by calling TrialFcn\_PumpControl and sets syringe diameter
     + Chooses certain number of safes then warn
   * Add any parameter tags that communicate with the Trial-Select Function
     + e.g. \*MIN\_NOGOS and \*MAX\_NOGOS (\* is so tags aren’t sent to TDT devices)
   * Add calibration files: Next to Tone Freq, click Calib and add the calibration file generated by **ep\_CalibrationUtil**
   * Can add .wav files – check the “.wav” box, gives a pop-up
     + Add them in order and note the correspondence of the indices of ordering (importing) them to their names – this is their File ID for referencing them. Useful to write this in “Protocol Notes” since it’s not saved to data file.
     + Under menu Settings, can check “Include wavfile buffers” to hold them in memory

* Use **ep\_DisplayPrefs.m**: Generates an .epdp bitmask-containing file.
  + This contains values from a bitmask table to enter into the state machine (an input to the ep\_StimDetect macro at the heart of the RPvds circuit).
  + Also load the .epdp file by clicking +DisplayPrefs within the RunExpt GUI.
* Create a **Matlab GUI interface** that can be default (boxfig) or custom
  + start with BoxFig then modify it (I think)
  + parameter tags in circuit with ! (bang) beginning their names can appear in BoxFig GUI
  + all BoxFig GUIs are in C:\Electropsych\epysch\boxfigs
* Use **ep\_CalibrationUtil.m** for calibration.
  + Settings → Signal Amplitude – choose voltage (maybe 1V) as the amplitude used to generate signals
  + Set Hardware to GB (gigabyte interface) and RZ6 for both input and output. Choose a sampling rate to match that you’ll use in your paradigm (e.g. 100kHz).
  + Set up B&K Nexus amplifier and mic, with the output sent to RZ6 Analog In1. Don’t use any gain – if you do, levels will be off by that amount.
  + Place microphone into pistonphone. Note frequency and amplitude.
    - Enter that amplitude into “dB SPL” box under “Calibration” window on GUI, on left.
    - Enter that frequency: menu Settings → Reference Frequency
  + Turn on pistonphone. Press button “Ref. Piston”. This will generate a mV RMS associated with a the dB SPL you just typed in. That mV RMS will appear under “Calibration” (e.g., 881.4mV = 94dB SPL), overwriting the number that appeared there by default.
  + Look at the sine wave. It should be < +/- 10V in amplitude (TDT can only record +/-10V). Change the mic sensitivity on the B&K Nexus and/or the amplification on the RZ6 Analog Input until this is the case.
  + Now you’re ready to generate your calibration curve (separate curves for Tone and Noise).
  + Set Stimulus to Tone (or Noise). Actually click Tone or Noise to set the parameters.
    - For Tone, use Matlab syntax to enter frequency range you’d like to calibrate (e.g., 200:100:20000)
    - For noise, enter Highpass and Lowpass (e.g., 3400 and 4600 for 4kHz center freq @ 30% BW).
    - If it’s noise, you want the noise to match that you’ll be using in your paradigm. Edit the C:\gits\epsych\calibration\CalUtil\_RPvds\STACQ\_FiltNoise\_Calibration\_RZ6.rcx file to have equivalent biquad for sharpness, etc. as that in your experimental rcx file.
  + Under “Calibration” in the “Norm=” box, enter the loudest dB SPL level that you’d like your system to be able to achieve. (Don’t worry, this will NOT mean the speaker will present those loud levels at the moment.)
  + Click Run. A graph will appear with a line at the loudest level you’ve requested, and points indicating how loud your system can present each frequency.
  + Save calibration file (with .cal extension) when prompted. (You can point to this in the .prot file, created using ep\_ExperimentDesign.)
  + Check calibration levels independently, of course, using freestanding calibration rig
* **OpenEx** **integration** (optional):
  + Use the behavioral RPvds file for RZ6 control – but how does this access digital ports on the RZ5? Check.
    - The RPvds file running on the RZ6 can’t access the RZ5 digital ports directly. I believe we used a short BNC cable to connect an RZ6 digital port to an RZ5 digital port.
  + Modify a generic RZ5 circuit for neurophys (Neural\_16ch\_basic.rcx), specify our preamplifier, add storage of relevant timestamps (tone, trial, etc).
  + Make new project (use OpenProject, wrapper for OpenEx components). Then add components:
    - WorkBench, Controller, Scope, RPvds
  + On WorkBench, add RPvds file to device and name device.
    - Your .rcx circuit can be created and stored where you like. But when you create a new project in OpenProject and link to the circuit, a copy will be created wherever OpenProject creates your project! Careful when editing.
  + In Setup, disable:
    - Sweep Loop, Condition, Stimulation Timing, Acquisition Timer (might be a diff word here)
  + File → Data Tank Management – at setup, select Tank. New block each time you hit run.
    - Neuro option: one tank per electrode site, one block per data type\
    - Behav option: one tank per animal, one block per session
  + Use Stream\_Store to store licking constantly at fs=993.4924 Hz (in RZ6 behavior circuit); neural data is stored at 24.414 kHz (in RZ5 neural circuit).
  + Run using ep\_RunExpt
    - Create data tank
    - Register tank to make asterisk appear (what does this mean?)
      * Registering a tank simply means that it will appear in the tank list the next time you open the program. This is optional (but convenient) and a tank can be manually located using the open folder button

To run a paradigm if you’re not using OpenEx:

* From Matlab, type **ep\_RunExpt**. (Opens RunExpt GUI.)
* Open or create .config file (one .config file per gerbil).
  + Open an animal’s .config file OR
  + Create: Click on +Subject to add one animal. (One subject / box. Multiple boxes run several animals at once; we have one box).
* Locate experimental protocol .prot file (created by ep\_ExperimentDesign)
  + For animal-triggered conditioned avoidance task: C:\Electropsych\Conditioned\_Avoidance\FwdMask\_AnimalTrig.prot
  + Can highlight subject, then click Edit Protocol to see associated .prot file
  + Can highlight subject, then click View Trials to see trial types.
* Click on +DisplayPrefs to associate a .epdp file (bitmask for state machine, created by **ep\_DisplayPrefs**)
  + For animal-triggered conditioned avoidance task: FwdMaskAversive.epdp
* Under Settings in menu
  + Define Saving Function (as e.g. ep\_SaveDataFcn)
  + Define box figure (our customized GUI interface: e.g. ForwardMaskAversiveGUI.m)
* Saves data as .mat with timestamp (default spot is in Documents folder)

To run a paradigm if you’re using OpenEx (records lick and neural data):

* Open the project (.wsp file) that you created using OpenProject
* From Matlab, type **ep\_RunExpt**
* Open or create .config file (one .config file per gerbil)
  + Open an animal’s .config file OR
  + Click on +Subject to add one animal. (One subject / box. Multiple boxes run several animals at once; we have one box).
* Locate experimental protocol .prot file (created by ep\_ExperimentDesign)
  + For animal-triggered conditioned avoidance task: C:\Electropsych\Conditioned\_Avoidance\FwdMask\_AnimalTrig.prot
  + Can highlight subject, then click Edit Protocol to see associated .prot file
  + Can highlight subject, then click View Trials to see trial types.
* Click on +DisplayPrefs to associate a .epdp file (bitmask for state machine, created by **ep\_DisplayPrefs**)
  + For animal-triggered conditioned avoidance task: FwdMaskAversive.epdp
* Under Settings in menu
  + Define Saving Function (as e.g. ep\_SaveDataFcn)
  + Define box figure (our customized GUI interface: e.g. ForwardMaskAversiveGUI.m)
* Saves data as .mat with timestamp (default spot is in Documents folder)
* Saves lick/neural data in TDT Tank format (best to store in E:\TANKS\)
  + Lick data is stored at fs=993.4924Hz; neural data is stored at 24.414kHz (how is this set up? How is LICK data assigned and appearing as such?)
    - The TDT macros which store streaming data can be downsampled by double clicking the macro and editing the parameters.

Creating TDT RPVds .rcx paradigm file

* Setup sampling rate you want (100kHz)
* Add core sweep control for RZ6 (DS has alternative macros with less cycle usage if necessary)
* Add RZ6-control (to control digital in/outs)
* Add RZ6\_AudioOut
* In browser within RPVds, browse to Epsych macros and add them so they become visible.
* Add custom macro, e.g. ep\_StimDetect
  + ep\_StimDetect is a macro with timing circuitry, customized for our needs. Double-click for description.
  + Set response window delay and response window duration
  + Output RespWindow is high when it’s within the response window.
  + Input = TTL to determine when trial starts
  + IMPORTANT: In clock (under Setup), match RZ6 sampling rate setup.
  + Trialoffset is variable. It’s based on when animal responds OR end of response window.
* For calibration: go to SignalCal macros, add name that matches the tag of whatever is being calibrated (using **ep\_CalibrationUtil**)

Specific to AM Aversive (C:\epsych\Aversive\AversiveAM\RCOCircuits\AversiveAMTest.rcx)

* **Optical Lick** A4 (BitIn) →DeBounce → custom macro ep\_StimDetect
* Schmitt →BitOut **Pump** A2 (C.2 = M2)
* Schmit → BitOut **Shocker** A6 (C.5 = M32)
* IR optical lick → Peterbox input3 → Peterbox TTL out 3 → patchpanel A4 (RZ6\_control byte C.3 input)
* Pump into patchpanel A2 (RZ6\_control byte C.1 output = 22 = M2)
* Shocker into patchpanel A6 (RZ6\_control byte C.5 output = 25 = M32)

State Machine to determine animal response:

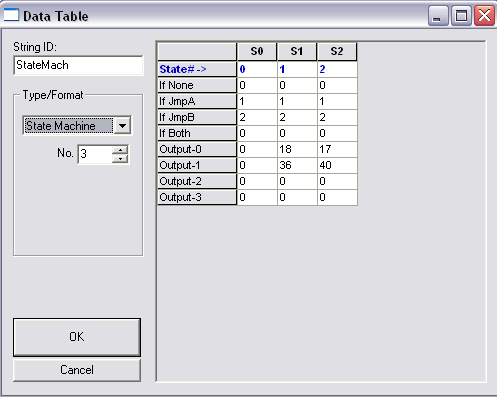
The SMData Input on the ep\_StimDetect macro takes a state machine input. This provides bitmasks to uniquely address different outcomes: hit, miss, CR, FA.

* Add a Data Table, and make it a State Machine by choosing this type within it.
* Rows are SAFE or WARN
* Columns are responses (on or off spout)
* [I think for 2AFC, columns are time periods: 1) before trial, 2) response window, 3) after trial.]
* This is output from the macro as RespCode. Can break it out using ToBits.

In Matlab script ep\_DisplayPrefs.m: create State Machine. Code how animal responds on each trial. Add Functions: Shock, Hit, Miss, FA, CR, Warn, Safe. Choose them as appropriate in bitmask table.



Then put the values into the State Machine input to SMData. Whatever number you enter in JmpA row or JmpB row will direct you to that column (i.e., 1 goes to S1).



- CLARIFY: how do you send info into JmpA and JmpB to distinguish them as warn or safe?

Warn and Safe are “trial types” and are selected as Output-0 or Output-1 rows. In the macro, a state machine will activate JmpA if there is a response during the response window and JmpB if there is no response during the response window. As you can see in the screenshot here, we’ve defined JmpA to pick 1 for S1 column and JmpB to pick 2 for S2 column. Therefore, depending on the current trial type (output row), and the response (or no response) during the response window, the state machine will select the corresponding numeric value (bitmask).

This is a very simplistic use of the state machine. The state machine can be used in much more complex ways if your experimental design requires it (would likely require modification of the macro circuitry).

Our existing paradigms

|  |  |  |  |
| --- | --- | --- | --- |
| **Stimulus** | **Paradigm type** | **Experiment Type** | **Status** |
| FwdMask | AnimalTrig, intermittent rhythmic stimuli | aversive water | Ready to go? |
| FwdMask | Automatic, intermittent rhythmic stimuli | aversive water | .config probably needs to be re-done for OpenEx integration |
| FwdMask | Animal triggers LED TrialSignal -> single stimulus | aversive water | .rcx circuit working. Needs checking and integration with GUI and OpenEx. |
| .wav | AnimalTrig, intermittent rhythmic stimuli | aversive water | needs proper .wav files, .rcx check, integration with GUI and OpenEx |
| FwdMask |  | shuttlebox | While waiting for relays, start working on TDT RPvds circuit. |

**TO DO**:

* AnimalTrigFwdMask:
* To our control GUI (ForwardMaskAversiveGUI.m):
  1. Check calibration
  2. Add display info for amount water dispensed and time in cage
  3. Change timing of safes and ITIs – 1.5s ITI, #safes 1-3
  4. Parafilm connector to water tube
  5. Add display info about level or duration to the GUI as needed
  6. Add manual spout override.
  7. Implement diff type of pause while on spout: safe sounds but no safes.
  8. ~~Add button to change main parameter(s) tone dur and tone dB~~
  9. ~~Program up training & DL, link to buttons on main GUI.~~
  10. ~~Add edit box to change water rate~~
* Figure out the digital I/Os for the RZ5 – what are they for, and are all needed for our AnimalTrigFwdmask? Note any that we can disconnect (ie., any that are used only for the LEDSignalFwdMask) in order to have one extra for a manual spout override.
* LEDSignalFwdMask
  + Get LED-signalled circuit to work
  + Create .prot file and test it
* Where is struct TRIALS stored and how do we access it?
* How is NextTrialID sent from TrialFcn back to RunExperiment?
* Shuttlebox (physical):
  + Look at withdrawing spout at MedAssociates, or implement withdrawal with a servo motor
  + Build shuttlebox control box
  + Modify shuttlebox for better sound field
* Shuttlebox (programming):
  + Program up a paradigm with no OpenEx component.

Matlab tips:

* ep\_ (hit tab) will show a list of all scripts that begin with “ep\_” – useful to find essential GUIs
* when creating our own GUI, want access to globals.
  + RUNTIME global has data. Big struct with RUNTIME.TRIALS.DATA etc
  + AX global – control used by TDT to access equipment during runtime
* Useful functions: TDT2ddt, TDT2plx, TDT2mat
  + data = TDT2mat('E:\TANKS\Test', 'Block-7');
  + lick = data.streams.LICK.data;
* [data.Tone\_Dur] makes a vector from a struct field!  
  ind = [data.Tone\_Dur] = 30  
  [data(ind).ResponseCode]
* load(‘C:\file.prot’, ‘-mat’) – reads a file that is a .mat file even though its extension doesn’t match

RPVds tips:

* colors for links are indicators. Green=logic. Bright blue=floating point. Dark green=integer.
* Access RZ6 digital I/O using RZ6\_Control macro. RZ6 with 8 digital I/Os = Byte C.
  + Byte-C.0 = A1 = bit1 = 20
* **Schmitt** trigger (under Logical) detects and holds open rising edge of logical pulse for a given time. Schmitt vs Schmitt2 is in ms or samples.
* **Biquad** is a filter. Requires coefficient input, which you can create using coefficient generators, e.g. **BufCoef1**. BufCoef1 is simpler than BufCoef (only one component LP or HP or BP). FC is frequency center. There are 6dB/octave attenuation per biquad. Sharper falloff with more biquads. Send coefficient output of **BufCoef1** into Biquad.
* **DeBounce** (under Counters/Logic): looks at x samples at a time and ignores transients. Signal must be up for at least x consecutive samples.
* **HopTo** and **HopFrom** to avoid spaghetti code.
* **Cos2Gate** will gate on and off with cos2 ramp at begin AND end. The duration is sent into it using a **Schmitt** trigger. The Cos2Gate rise/fall is set within it. This function adds the duration of ONE of these rise/falls. So, add that fall time to the Schmitt duration (which sets signal duration by setting how long the signal is gated on).
* Basic tone presentation: ResponseWindow (or appropriate output from ep\_StimDetect) →Edge\_Detect →Schmitt(300ms) → Cos2Gate 🡪 RZ6\_AudioOut  
   also, Tone goes into the gate →
* Create in RPVds, e.g., RespWinDelay, RespWinDur, Freq
* To quick-communicate with **BoxFig**, use ! (bang) within a parameter tag – this will make it show up with ep\_BoxFig when you’re running the experiment.
* For communicating with ep\_ExperimentDesign, within circuit use \* before parameter name so parameter won’t get sent to TDT devices.

RZ6 Digital IO: A4 = Byte-C.3 = 23 = 8

|  |  |  |  |
| --- | --- | --- | --- |
| 20 | 1 | A1 | Byte-C.0 |
| 21 | 2 | A2 | Byte-C.1 |
| 22 | 4 | A3 | Byte-C.2 |
| 23 | 8 | A4 | Byte-C.3 |
| 24 | 16 | A5 | Byte-C.4 |
| 25 | 32 | A6 | Byte-C.5 |
| 26 | 64 | A7 | Byte-C.6 |
| 27 | 128 | A8 | Byte-C.7 |

General tips:

* For **wav files**, use Audacity to ~match Fs. Format as mono (1-channel), 16 bit.

OpenEx tips:

* In RPvds, need CoreSweep Control. Use modified version that Dan made: **CoreSweepControl\_NoStore** – synchronizes across devices!
* OpenDev and ActiveX use different syntax in Matlab. Need to check with IF statement (really? what does this mean?)
* OpenProject: wrapper for OpenEx components
* Use tdt2mat in matlab to read data tanks.
* In whatever paradigm we made work with OpenEx, we’re storing OpenEx trial info using RZ5 circuit. Need to check whether there’s any delay that we need to account for. To do this, store the info using RZ6 and compare.