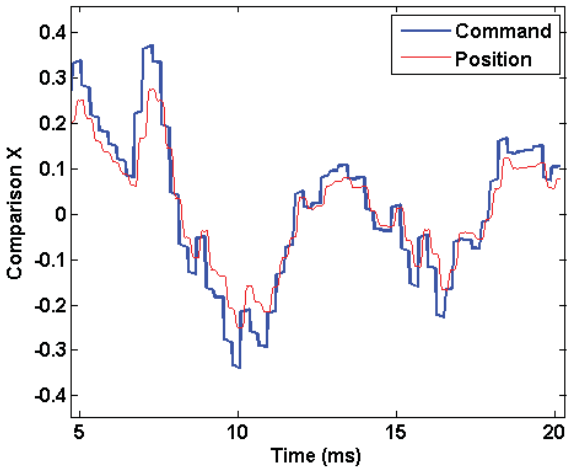
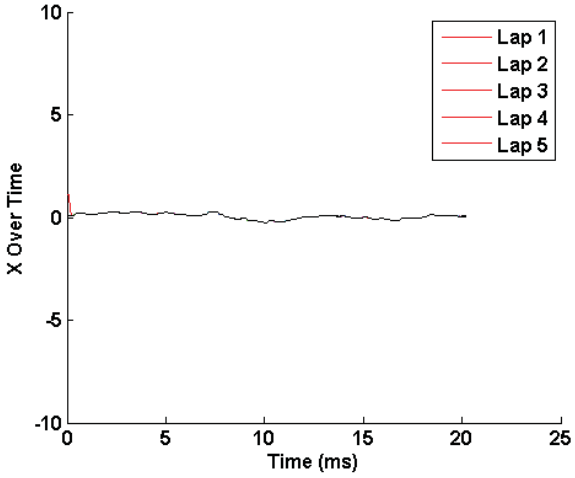
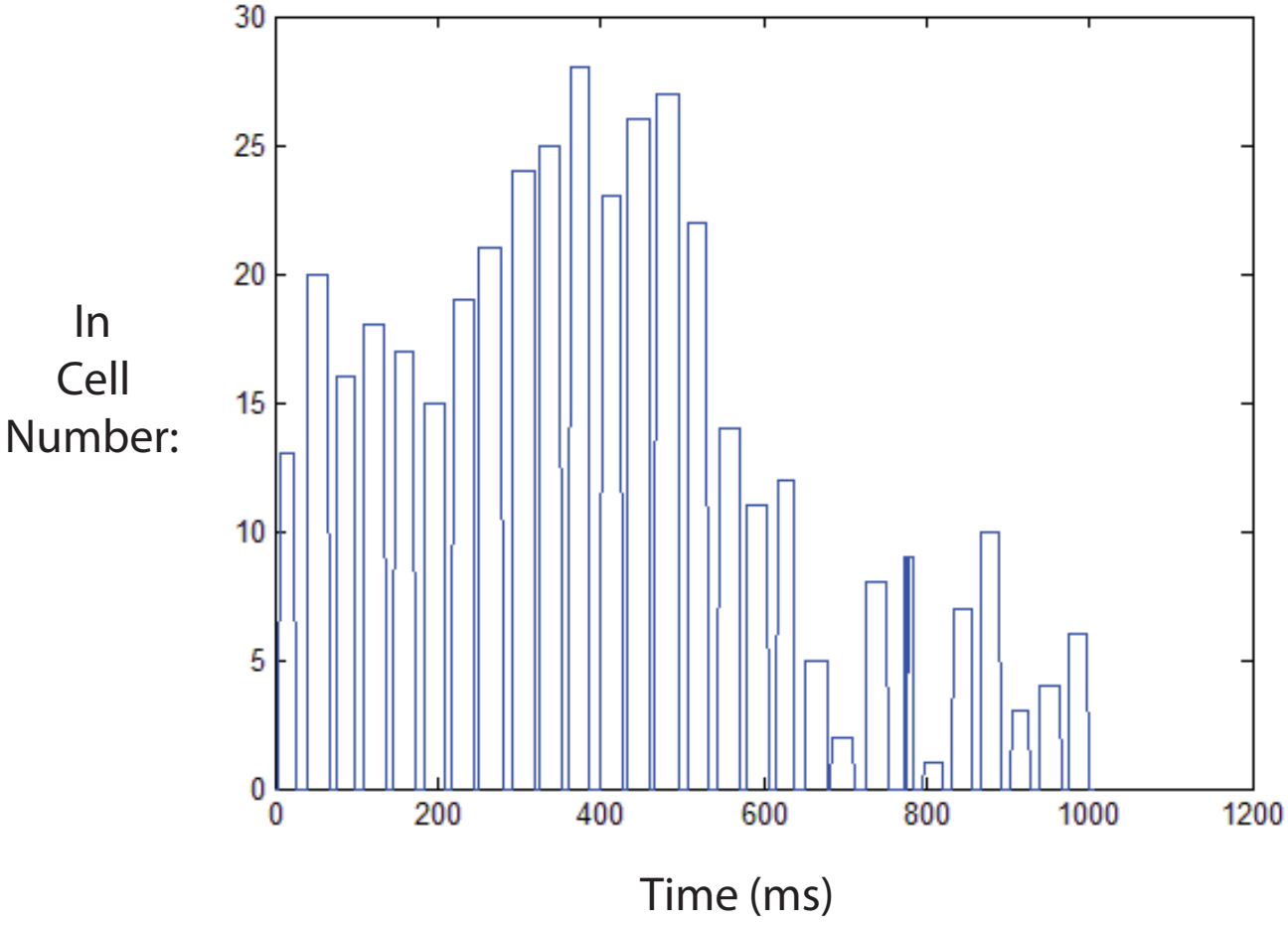
A



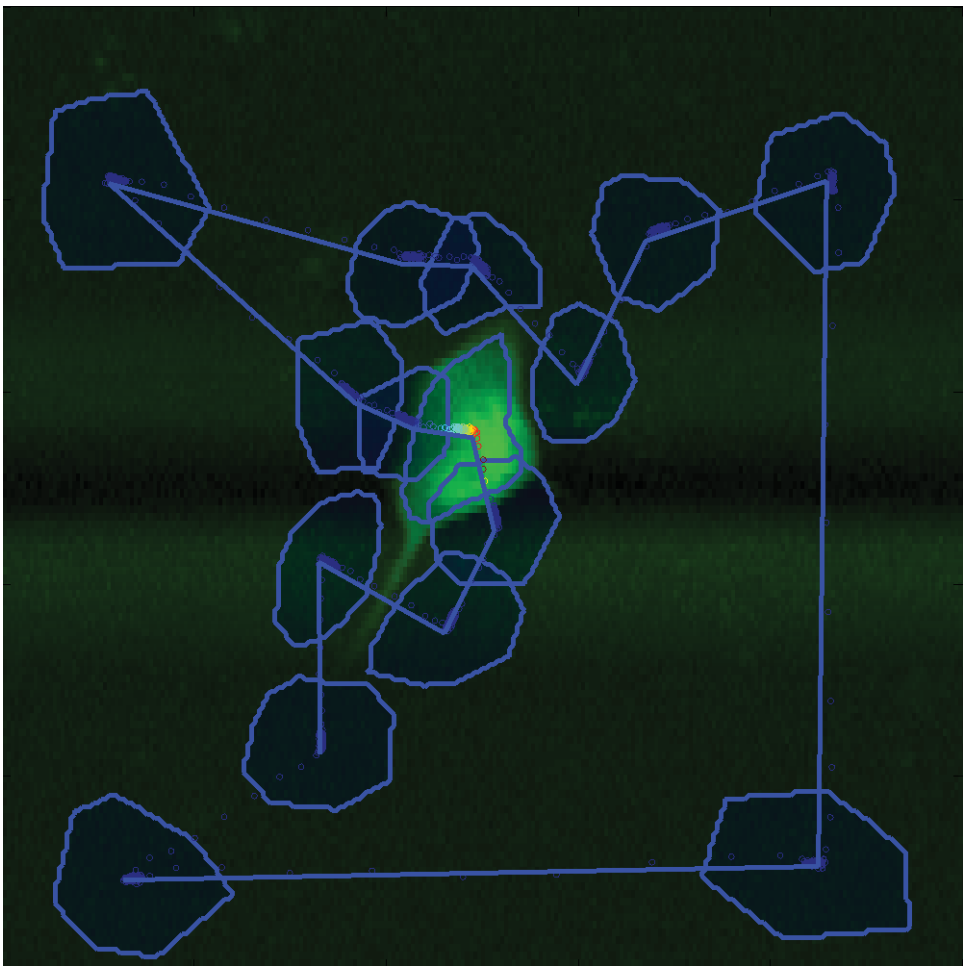
B



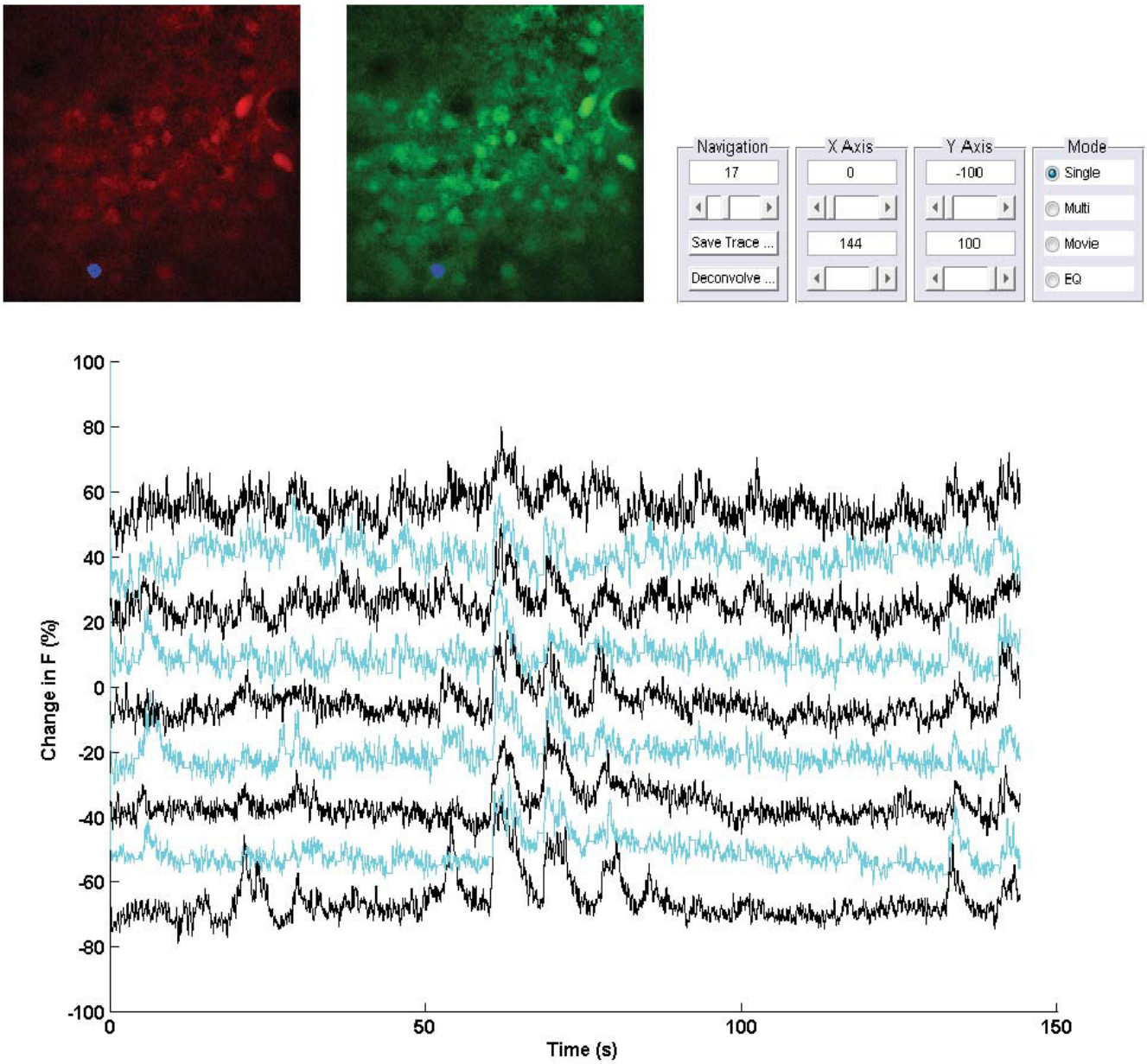
Supplementary Data 1. Mirror positions on some microscopes will lag command signals by a significant, variable amount that is repeatable and predictable. **A.** Comparison of timing of command signals sent (blue) and actual mirror positions recorded (red) with Y scaling added to mirror position for easier discernment of the X shift. As is shown, the actual position "lags" the command signal by a significant and variable amount. A simple X correction offset would not be sufficient to align the two. **B.** At the same time, the pattern of offset to a repeated command waveform (multiple laps) is also repeatable – it is consistently skewed in the same way from one lap to the next.



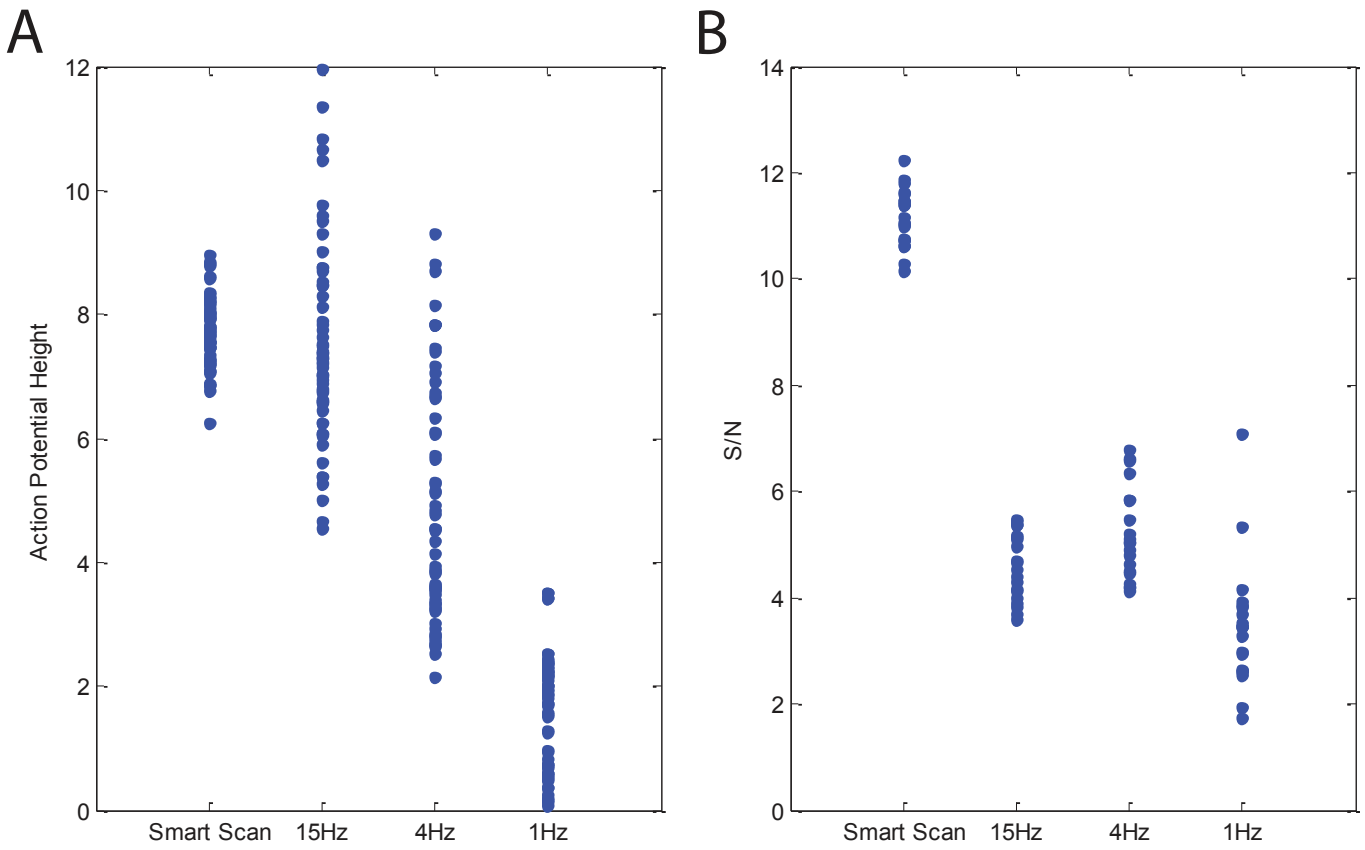
Supplementary Data 2. Landing on the cells with the help of mirror feedback. The mirror feedback in X and Y reveals the exact location of the mirrors at any moment of recorded fluorescence. Thus, at different points in the lap, the mirror position can be known to be in one cell or another. Here we plot which cell number the X,Y position is in at each sampled point of the lap. Mirrors spend a variable effective time in each cell, but most cells still receive significant dwell time over the course of the lap (a cell or two may be missed). By knowing the actual positions of the mirrors, it is possible to attribute fluorescence correctly back to the cells that emitted them.



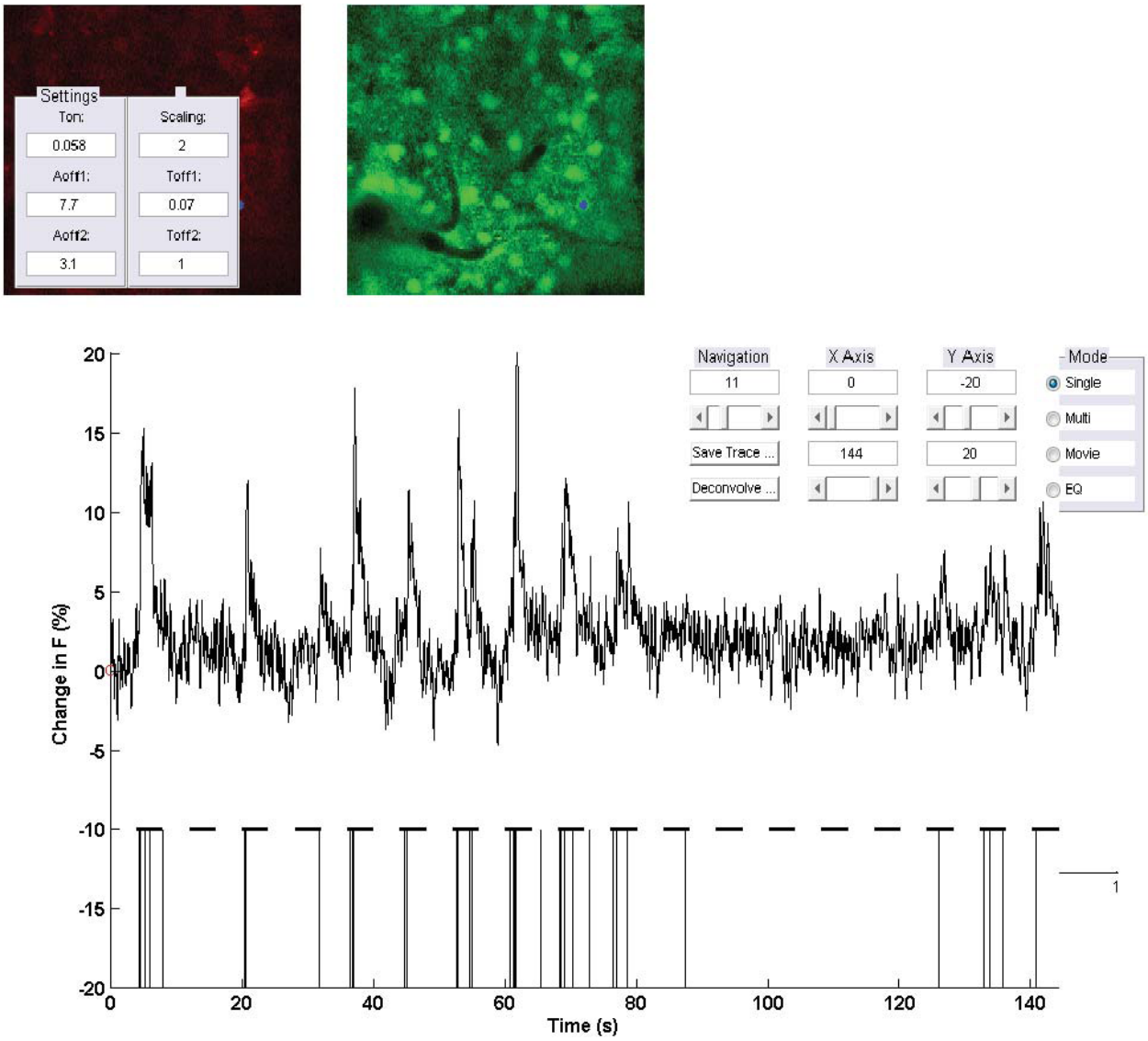
Supplementary Data 3. By knowing the actual position of the mirrors at the point when fluorescence is detected, it is possible to attribute the intensity of the signal back to the cell bodies that emitted them. Here we show a cell in an acute slice being imaged by the targeted path along with a number of surrounding fake regions of interest defined using the software. The scanpath trail is colored as a heat map according to the fluorescence intensity encountered at that location. Here, the “hot” intensity is restricted to the cell body, with “cold” intensity in the fake regions of interest. This is a good method for calibrating whether cell targeting is working correctly.



Supplementary Data 4. With careful calibration and proper attribution of fluorescence to spatial locations, delivering sensory stimuli across repeated trials in vivo will result in calcium traces collected from a given cell with a consistent selectivity for those sensory stimuli. Here we show multiple 144 s trials recorded from the same cell of interest. Light blue traces are trials recorded during optical activation of interneurons. These trials are interspersed with control trials (black) with no optical activation. In all trials, strong responses occur at the same points in the trial, because different orientations of a visual stimulus are delivered in the same pattern for each trial and the recorded cell is tuned to specific orientations of the stimulus.



Supplementary Data 5. **A.** Analysis of measured fluorescence signals from single action potentials. A signal source was generated by convolving spike trains with a kernel with a peak height of 7.7% [22] and adding Gaussian noise. This was repeated 20 times with pseudorandom noise. These signals were then sampled with parameters mimicking our smart scan protocol (50 Hz sampling of 50 cells), or raster scanning, using an example region of interest for a cell at 3x optical zoom. The plot presents the measured height of the AP-induced fluorescence signal (in % change) for each sampling mode. **B.** Analysis of signal to noise with different scanning protocols. Using the same data from (**A**), we computed the signal to noise as [average AP height] / SD(noise).



Supplementary Data 6. Proper calibration of the system will also allow for clean calcium transients that can be deconvolved to inferred spike trains automatically using the software. Here, an optically recorded trace (144 s) from a single cell is presented. Below the trace, there are horizontal lines depicting presentation of a visual stimulus, and then below that vertical lines depicting the estimated times at which the cell fired action potentials, as predicted by the deconvolution algorithm.

Supplementary Figure 1. Platform for high-speed, targeted neuronal activation during concurrent two-photon imaging, electrophysiology and sensory stimulation

Modification of the light path allowed the joining of an imaging beam (2P) and an excitation beam (473 nm) driven by independent random-access galvanometers. These pairs of galvanometers were brought under custom hardware and software control that allowed for the arbitrary targeting of specific cells within the network during concurrent high-speed targeted imaging. **A.** The point spread function of the stimulation beam was assessed by burning spots into a fluorescent substrate over long durations (minutes), which should provide an upper bound on the maximal area experiencing energy transfer during illumination with the targeted stimulation beam. Through the focus and aperture of our optics we were able to get the stimulation beam spread to a scale comparable to that of the 2P beam. **B.** The Network Visor software was written to synchronize the position and timing of the photoactivation beam with that of the high-speed imaging, electrophysiology, and visual stimulation. **C.** Measurement of the functional impact of the beam physiologically. ChR2 was expressed in neurons in acute slices (300 μm) (including those deep in the slice), the neuron was visualized under 2P illumination and patched, and the targeted beam was "stepped" towards the neuron until an action potential was detected. Scale bar: 100 μm. **D.** We found that the beam was on target with eliciting action potentials at the cell's location, and that the action potential disappeared when the beam was stepped beyond ~50 um from the cell. **E.** Quantification of the experiment in C-D, showing the rapid falloff as the beam left the vicinity of the cell. Reprinted with permission from [24].

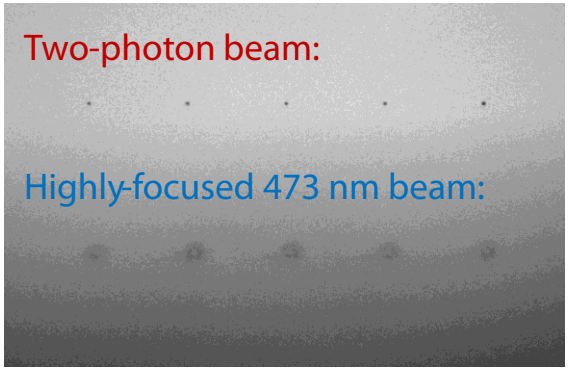
Supplementary Figure 2. Further calibration of targeted cell stimulation for optical activation in vivo

In slice experiments, the functional radius of the targeted activation to produce ***any*** detectable depolarization under voltage clamp was measured at different beam intensities, in order to calibrate a beam intensity that was effective but localized. **A.** DIC image of a patched Thy1-ChR2 cell used for the calibration. Scale bar: 40 μm. **B.** Laser pulses of 10 ms in duration were delivered using different laser intensities by applying different-sized analog pulses (2.5V, 4V, and 8V) to the laser driver of a 200 mW laser. As light intensity increased, so did the size of the depolarization current observed. Scale bars: 200 ms, 100 pA. **C.** Radii of minimal detectable depolarization in voltage clamp for different laser powers, depicting the area where shooting the beam effectively depolarized the patched neuron with any detectable voltage transient, no matter how small. Increasing laser intensity increased the size of the area. Scale bar: 40 μm. **D.** Quantification of the effective beam radius for detectable neuronal depolarization at different laser powers. **E.** Demonstration of the spatial acuity of the laser in real-time. Periodic laser stimulation pulses are delivered at 6V amplitude, depicted as blue dots, to create depolarization in the patched cell. The mirrors are first pointed directly at the cell, and regularly spaced depolarization transients result (start of trace). Then, at the location “Flip to Nearby Position Now”, a mirror transient is observed as the mirrors are directed to a nearby position in the same field of view, and the depolarization transients are immediately abolished. Finally, at the location “Flip Back to Target Now”, a second mirror transient is observed, and the depolarization pulses resume as the laser is again targeted at the cell. Scale bars: 500 ms, 100 pA. Reprinted with permission from [24].

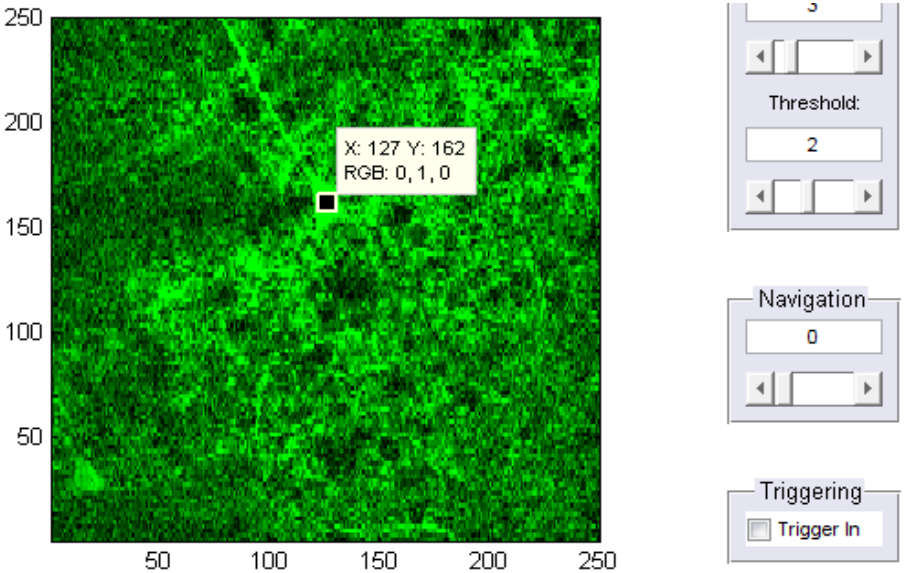
Supplementary Figure 3. Specificity of activation of neurons in the z-plane
A. We mapped the ability of focal blue laser stimulation to elicit spikes both inside and outside the plane of focus that included the ChR2+ neuron's cell body, at 16 μm resolution. The spike probability from locations 100 μm above the neuron was virtually 0 except for one location directly above the cell body (turquoise dot in top grid), which elicited spikes with very low probability. Therefore, in order to elicit spikes, even at low probability, from outside of the z plane of the ChR2+ neuron's cell body, the focal stimulation must be directly above the soma. **B-C.** Cells were selected for targeting based on careful screening of the area for sparse ChR2 expression, in both XY and in depth by taking Z stacks. These experiments were also intentionally done in mice with sparse virally expressed ChR2 expression. Given this sparse expression in our experiments, where neurons at different depths were rarely if ever in the same xy location, it is unlikely that we activated neurons from depths above or below the neuron of interest. Here we show the xy planes from z-stacks that contained mCherry-ChR2+ neurons from a representative PV-Cre mouse (B), where 315 μm of depth were examined at 3 μm increments and a SOM-Cre mouse (C), where 250 μm of depth were examined at 5 μm increments. The cross hairs in each plane indicate the focus of the orthogonal XZ and YZ planes shown to the right and below each image, and show that no other mCherry-ChR2+ neurons were directly above or below any of these neurons. Reprinted with permission from [24].

FIG. S-1

A

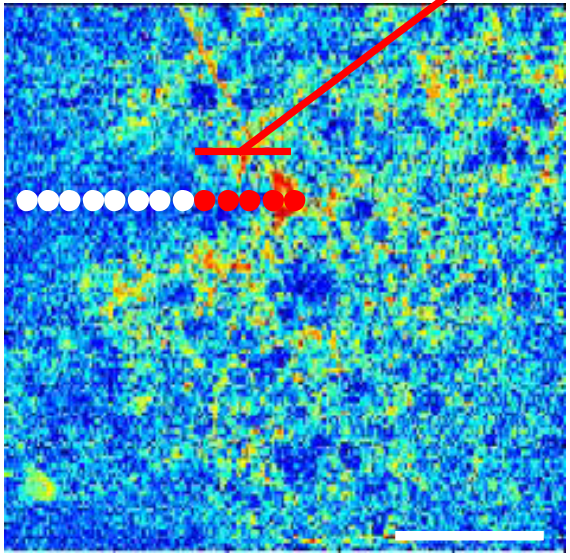


B

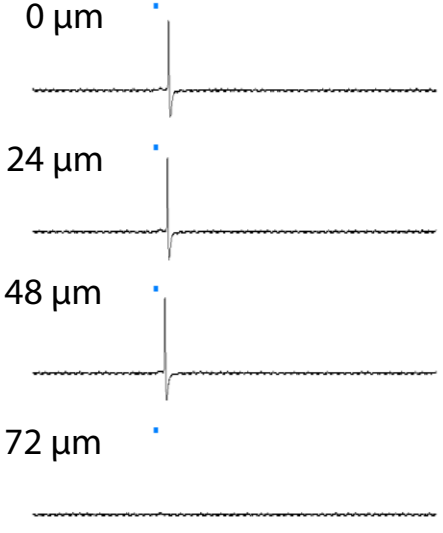


Effective Radius: ~ 50 µm

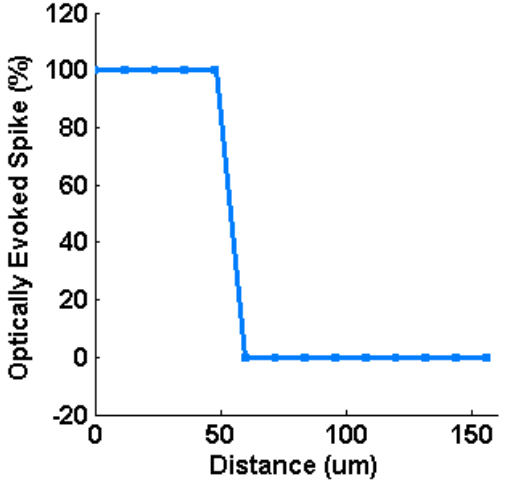
C



D



E

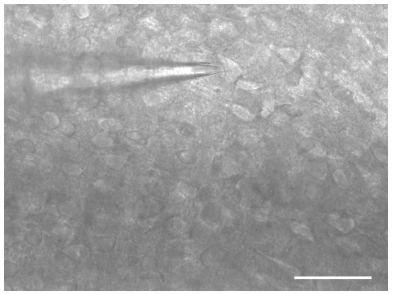


Supplementary Figure 1. Platform for high-speed, targeted neuronal activation during concurrent two-photon imaging, electrophysiology and sensory stimulation

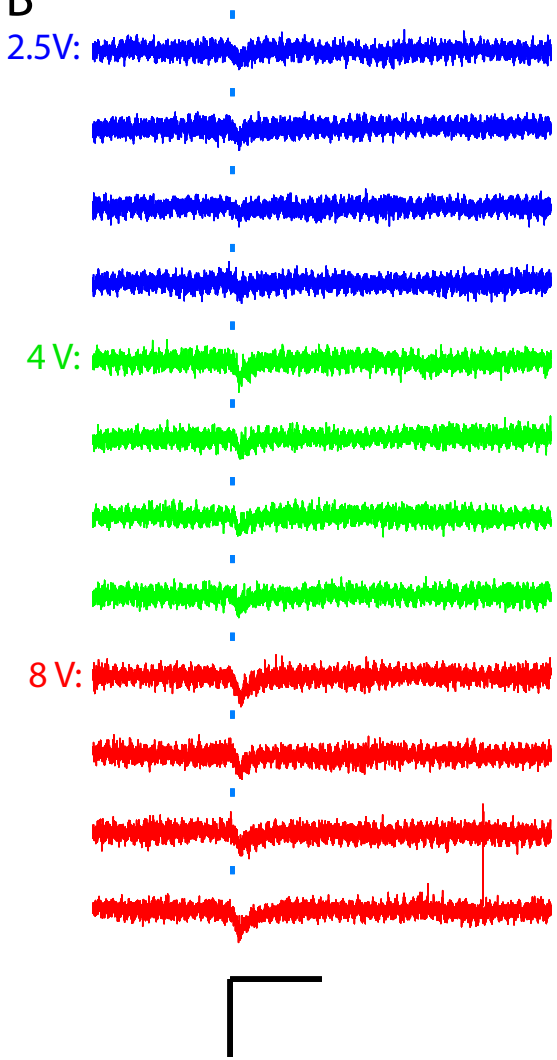
Modification of the light path allowed the joining of an imaging beam (2P) and an excitation beam (473 nm) driven by independent random-access galvanometers. These pairs of galvanometers were brought under custom hardware and software control that allowed for the arbitrary targeting of specific cells within the network during concurrent high-speed targeted imaging. **A.** The point spread function of the stimulation beam was assessed by burning spots into a fluorescent substrate over long durations (minutes), which should provide an upper bound on the maximal area experiencing energy transfer during illumination with the targeted stimulation beam. Through the focus and aperture of our optics we were able to get the stimulation beam spread to a scale comparable to that of the 2P beam. **B.** The Network Visor software was written to synchronize the position and timing of the photoactivation beam with that of the high-speed imaging, electrophysiology, and visual stimulation. **C.** Measurement of the functional impact of the beam physiologically. ChR2 was expressed in neurons in acute slices (300 μm) (including those deep in the slice), the neuron was visualized under 2P illumination and patched, and the targeted beam was "stepped" towards the neuron until an action potential was detected. Scale bar: 100 μm. **D.** We found that the beam was on target with eliciting action potentials at the cell's location, and that the action potential disappeared when the beam was stepped beyond ~50 um from the cell. **E.** Quantification of the experiment in C-D, showing the rapid falloff as the beam left the vicinity of the cell. Reprinted with permission from [24].

FIG. S-2

A

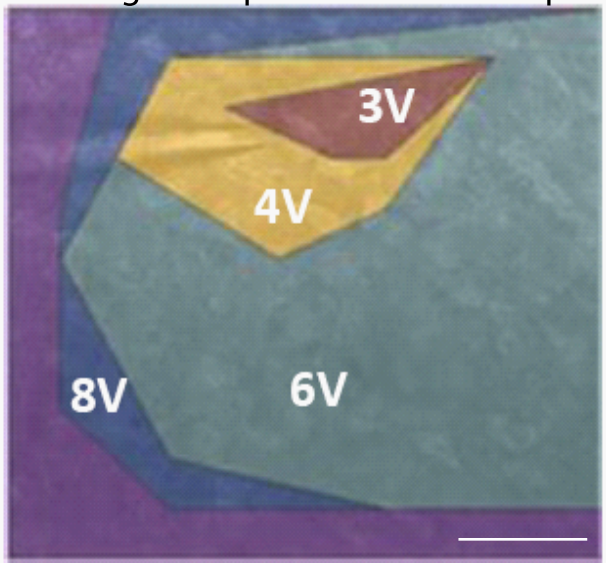


B



C

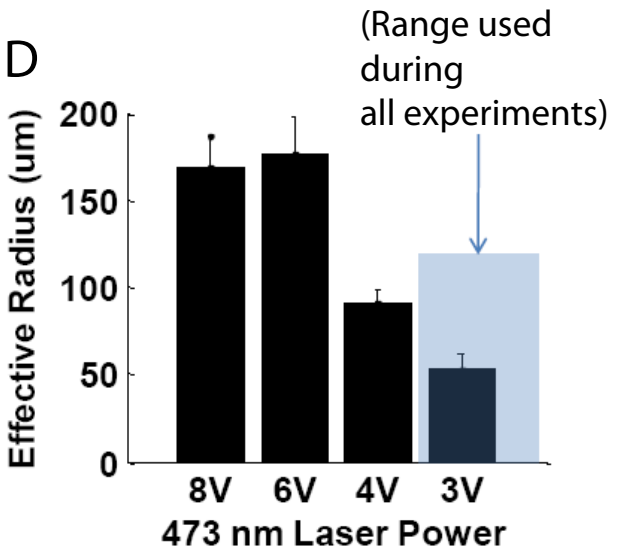
Radii of minimal detectable depolarization in voltage clamp for different laser powers:



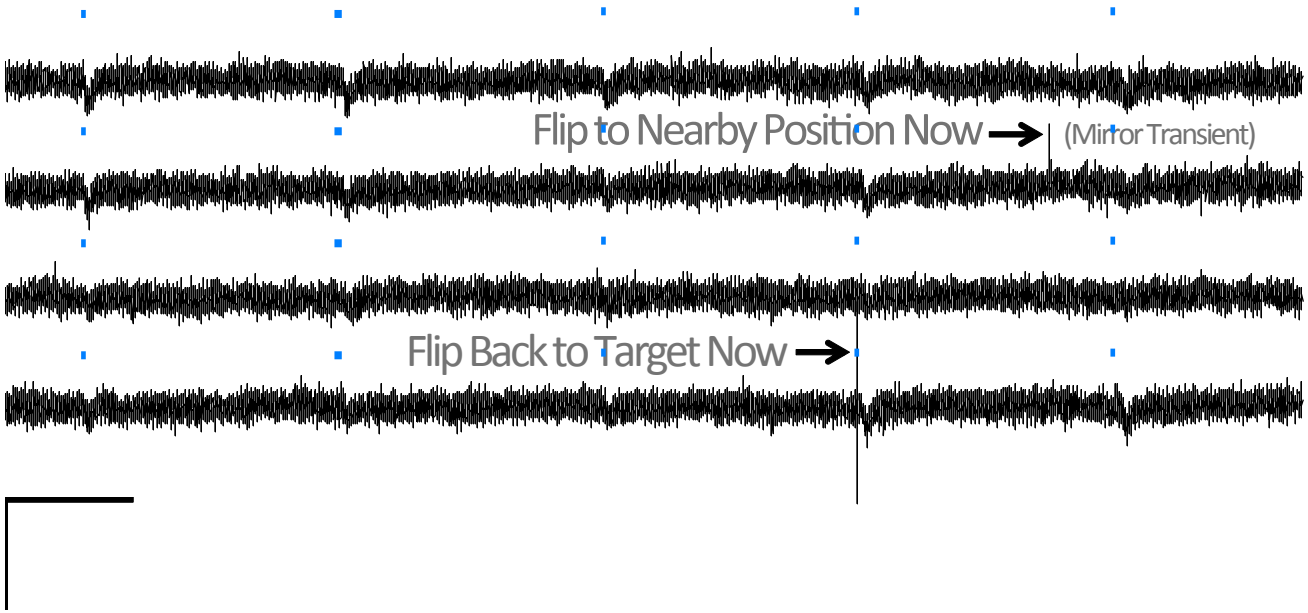
Amplitude driving 200 mW laser:

3V 4V 6V 8V

D



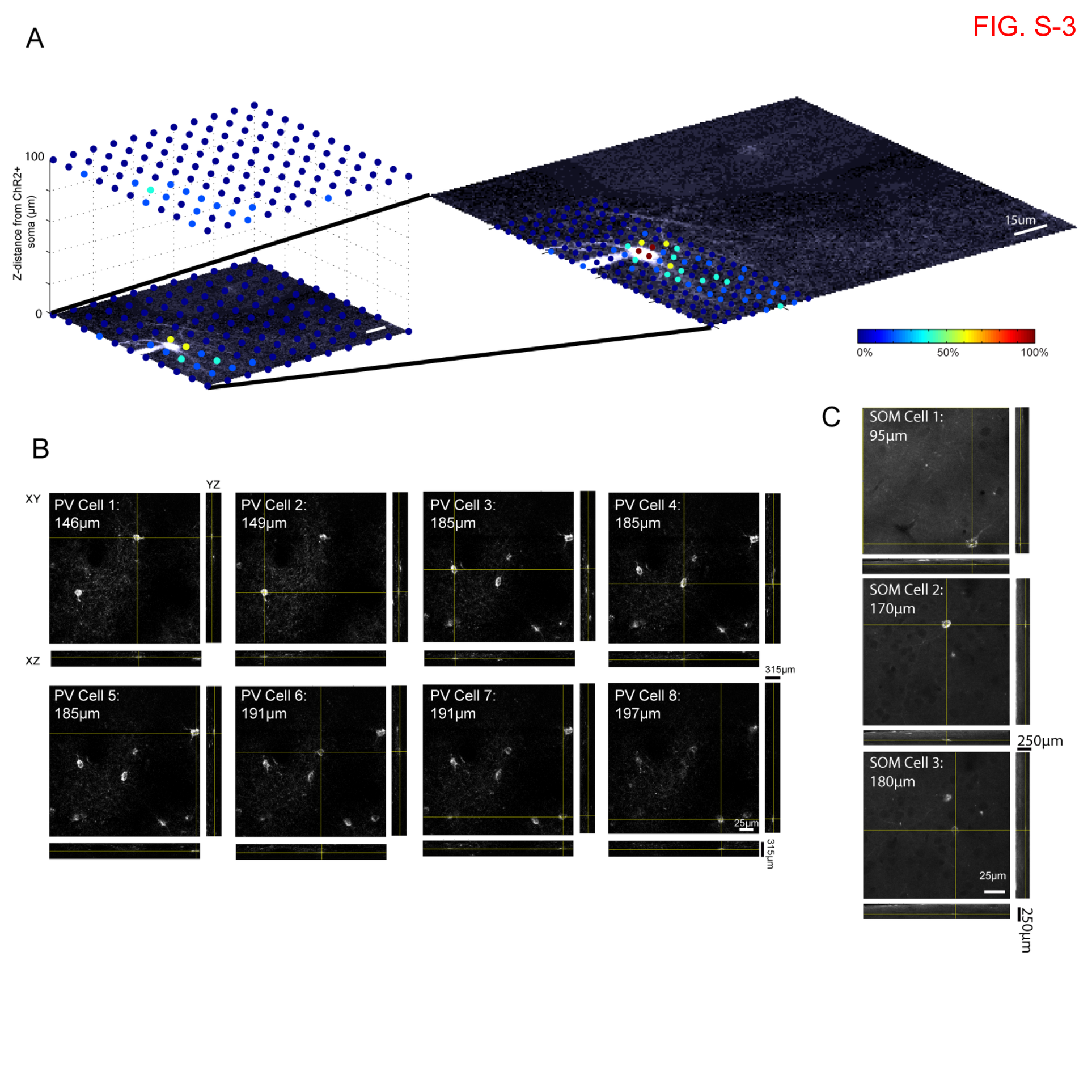
E



Supplementary Figure 2. Further calibration of targeted cell stimulation for optical activation in vivo

In slice experiments, the functional radius of the targeted activation to produce ***any*** detectable depolarization under voltage clamp was measured at different beam intensities, in order to calibrate a beam intensity that was effective but localized. **A.** DIC image of a patched Thy1-ChR2 cell used for the calibration. Scale bar: 40 μm. **B.** Laser pulses of 10 ms in duration were delivered using different laser intensities by applying different-sized analog pulses (2.5V, 4V, and 8V) to the laser driver of a 200 mW laser. As light intensity increased, so did the size of the depolarization current observed. Scale bars: 200 ms, 100 pA. **C.** Radii of minimal detectable depolarization in voltage clamp for different laser powers, depicting the area where shooting the beam effectively depolarized the patched neuron with any detectable voltage transient, no matter how small. Increasing laser intensity increased the size of the area. Scale bar: 40 μm. **D.** Quantification of the effective beam radius for detectable neuronal depolarization at different laser powers. **E.** Demonstration of the spatial acuity of the laser in real-time. Periodic laser stimulation pulses are delivered at 6V amplitude, depicted as blue dots, to create depolarization in the patched cell. The mirrors are first pointed directly at the cell, and regularly spaced depolarization transients result (start of trace). Then, at the location “Flip to Nearby Position Now”, a mirror transient is observed as the mirrors are directed to a nearby position in the same field of view, and the depolarization transients are immediately abolished. Finally, at the location “Flip Back to Target Now”, a second mirror transient is observed, and the depolarization pulses resume as the laser is again targeted at the cell. Scale bars: 500 ms, 100 pA. Reprinted with permission from [24].

FIG. S-3


A
100
0
Z-distance from ChR2+ soma (µm)
15um
0%
50%
100%
B
XY
YZ
XZ
PV Cell 1: 146µm
PV Cell 2: 149µm
PV Cell 3: 185µm
PV Cell 4: 185µm
PV Cell 5: 185µm
PV Cell 6: 191µm
PV Cell 7: 191µm
PV Cell 8: 197µm
315µm
25µm
315µm
C
SOM Cell 1: 95µm
SOM Cell 2: 170µm
SOM Cell 3: 180µm
250µm
25µm
250µm

Supplementary Figure 3. Specificity of activation of neurons in the z-plane
A. We mapped the ability of focal blue laser stimulation to elicit spikes both inside and outside the plane of focus that included the ChR2+ neuron's cell body, at 16 μm resolution. The spike probability from locations 100 μm above the neuron was virtually 0 except for one location directly above the cell body (turquoise dot in top grid), which elicited spikes with very low probability. Therefore, in order to elicit spikes, even at low probability, from outside of the z plane of the ChR2+ neuron's cell body, the focal stimulation must be directly above the soma. **B-C.** Cells were selected for targeting based on careful screening of the area for sparse ChR2 expression, in both XY and in depth by taking Z stacks. These experiments were also intentionally done in mice with sparse virally expressed ChR2 expression. Given this sparse expression in our experiments, where neurons at different depths were rarely if ever in the same xy location, it is unlikely that we activated neurons from depths above or below the neuron of interest. Here we show the xy planes from z-stacks that contained mCherry-ChR2+ neurons from a representative PV-Cre mouse (B), where 315 μm of depth were examined at 3 μm increments and a SOM-Cre mouse (C), where 250 μm of depth were examined at 5 μm increments. The cross hairs in each plane indicate the focus of the orthogonal XZ and YZ planes shown to the right and below each image, and show that no other mCherry-ChR2+ neurons were directly above or below any of these neurons. Reprinted with permission from [24].

Controlling a Prairie Microscope with ScanImage Software

Nathan Wilson / Sur Laboratory

Overview:

ScanImage from HHMI Janelia Farm is a powerful software package for doing 2P imaging and learning about the associated principles. Prairie Technologies makes 2P imaging systems that are increasingly used. In working with Prairie microscopes and hacking them for custom control it became apparent that the same configuration could enable any custom software to control the Prairie setup. The steps below are meant to illustrate this and help you on your way in using ScanImage or other software packages with a commercial Prairie microscope. This is instructional / use at your own risk, but we hope you find it helpful in your own work.

About the Integration:

The essence of the integration is to set up a second computer that can "borrow" the Prairie scanners and run alongside the normal Prairie setup. The goal is to not remove any part of the existing Prairie setup, but rather to add an extra control channel to the setup.

Basically what you will do is add a second "auxiliary" line of controls to the mirrors via the galvo box, and then add a toggle command signal that can switch the galvo box to listen to your second computer instead of the original Prairie computer. The Prairie computer remains connected to the galvo box as well, using the original inputs.

Finally, you will attach in (via BNC splitter) a control line from your second computer to the existing existing shutter command line from the Prairie computer. When you scan using your second, custom computer, it will send a "high" signal to the shutter to open it, send a "high" signal to the galvo toggle to tell it to listen to its X/Y mirror commands instead of the Prairie computer's, and then it will do the scanning. When it is done scanning it will send low signals to close the shutter and release the galvos. You could then immediately take a scan with the Prairie computer. Pockels and PMTs on the Prairie are controlled either manually using neither computer, or from the Prairie computer and software – no control lines need to intercept these from the custom computer, though in principle it is of course possible.

Driving a Prairie with ScanImage:

Details of this integration may over time stop making sense after this publication. However, the general principles should remain sound, and we may update our respective websites with new information as ScanImage or Prairie evolves.

1. Perform the steps in the Procedure, ***III. Steps 9-21: Connecting computer to two-photon imaging microscope*** For your Prairie microscope, making the connections described in Figure 2 and Table 2. You will now have a second "custom" computer that can control the Prairie using Visor, etc.

2. Install ScanImage from the Janelia Farm website.

3. In the "Machine Data File Configuration", replicate the relevant settings for PMT inputs, mirror commands, shutter control, and trigger control that are set in the Visor software.

Running ScanImage and Visor at the same time:

Finally, you can also run ScanImage and Visor at the same time – check Visor code that is instantiated during scan start and scan stop under the "Sutter" microscope configuration to see how Visor can gently "borrow" the data acquisition cards from ScanImage, and then "give back" the cards at the end of the scan.

Overview of the Software

Here we present a brief overview of the principles that we use for targeted scanning, followed by detailed documentation on the software and how it works for imaging and controlling data cards more broadly. We hope you find this useful for conducting circuit neurophysiology in time/space, and developing future computer interfaces.

Once you understand these principles in Matlab (perhaps even the graphical interface libraries can be useful to you) they are readily applicable to any computer-brain effector or measurement method that you might create.

Additional documentation for more elaborate tricks may be added online over time.

Algorithmic Principles Used

Cell segmentation. Our software implements a fast cell segmentation method developed in our lab and optimized for neurons and astrocytes. The basic premise is to apply a "watershed filter" that raises the "water level" on the background until only the "peak plateaus" (cells) of pre-specified radii are "above water" as determined by comparing a noisy 2-dimensional Gaussian and filling in the "convex hull" areas of a common intensity level. In practice in our software, the user has control over two parameters that let them specify the typical "cell radius" for the software to expect (which uses the user-specified "zoom" and "lens magnification" to determine the actual size of what it is detecting), and then adjust the "threshold" which corresponds to the water level to subtract the background and isolate the cells.

Targeted path selection. Our software implements a previously published algorithm (Lillis et al., 2008) and a method to optimize the distance that the laser must travel to pass through all selected cells. We then provide an interface to select those cells, add or remove cells from the targeting, and even define custom ROIs within the path. Once the cellular ROIs are defined, the TPS module will use a genetic algorithm approach to "evolve" an optimized solution to the Traveling Salesman problem, and after a software-specified number of iterations or convergence will present the "pretty good" solution as the optimized scanpath. It then does further computation to distribute the sampling points given the time that it has per lap, to concentrate more sampling points along this path inside the regions of interest, and fewer within the "gaps" between ROIs along this path. The density of sampling points within cells can be adjusted (% in cells) as can the total lap time, or of course the number of cells per lap.

Deconvolution of optical signal to spikes. Our software and protocol implements a previously published deconvolution algorithm that we found especially helpful (Grewe et al., 2010). Here, a "canonical" dF/F waveform corresponding to one action potential is specified by the user, by adjusting several parameters corresponding to the dF/F amplitude, time courses, etc. The parameters can be adjusted and "run" on the waveform to assess quality of fit, and future software versions could automate this fitting process by identifying a "minimal" dF/F event that likely corresponds to a single AP and then running automated curve fitting on that segment. In either case, the canonical dF/F waveform ("template") is then applied to the actual dF/F trace (which can be optionally low-pass filtered to reduce noise) using the following steps: First, the canonical waveform is marched along the actual trace until it encounters a dF/F "bump" that crosses a simple threshold. The template is then positioned at this point, and "peeled off" or subtracted from the actual waveform. This reduces the actual waveform by the amount of the template at that location. A second template is then marched along the actual trace again, starting at the beginning as before, and this time may surpass the previous point and encounter a new point that is above threshold, where it also is "peeled" or subtracted from the actual waveform. This process continues until the actual waveform has been reduced by all of the peels and there are no further

points above threshold. The locations of the peeling events, where the templates were positioned to peel, denote the location of the likely action potentials (minus a small temporal offset to account for the delay between AP and calcium response).

Visor > Specific Software Documentation

Below we have extracted specific files that are particularly useful for overcoming key conceptual or technical hurdles in targeted scanning and stimulation.

These are meant as starting points that can be readily extended or improved.

They refer to functions as described that were implemented at different points in the Visor software folder.

Good luck incorporating and improving on these building blocks, and happy interfacing - by focusing on making each component robust, one at a time, the system should orchestrate as a whole with quite high reliability in the heat of experiments.

Finally, hopefully the software itself can also provide broader examples for how to make open physiology interfaces to customize experiments in Matlab.

Functions and Scripts > Starting the Software

Function: visor()
Location: visor/visor.m
Purpose: runs the software and launches the graphical user interface
Inputs: [none]
Outputs: [none]

Description: This is what you type into the Matlab command line ("visor") to run the software.

Code:

```
% Network Visor
% Nathan Wilson
% 2009-06-01

% all information for Visor is stored in the global "data" structured variable
global data;
data = [];

% add all functions for the software
visor_go();

% load initial settings
visor_load_initial_settings();

% load user settings
visor_load_user_settings();

% load microscope settings
visor_load_rig_settings();

if data.acq.rig_is_verified

    % start the program by launching the GUI
    visor_create_GUI();

    % send a stop siganl to the boards to make sure that everything is in
    ground state
    if visor_strstr(data.user_settings.rig_type, 'Analysis')
    else
        visor_stop_acquisition();
    end

end
```

Functions and Scripts > Starting the Software > Loading Rig Settings

Function: visor_load_rig_settings()
Location: Visor / Software / User Settings / visor_load_rig_settings.m
Purpose: configures the software for data cards to drive 2P imaging and stimulation
Inputs: [none]
Outputs: [none] – changes the global "data" variable to reflect the new settings.

Description:

- This will use the "rig type" that you define in an external file to set some default parameters for the scope of choice.
- IIf you set the rig type as "Prairie", it will use the custom configuration that we use in our lab, which you can modify.
- If you set the rig type to "Sutter", it will set things similar to a default install of ScanImage, and will expect ScanImage to be running alongside (this can be disabled).
- If you set the rig type to "Analysis", it will assume no rig is attached, and you are back at your desk using Visor to analyze code.

What it Does:

- Gets the "rig type" that is specified in an external file
- Sets which of the computer's hardware cards to use for 2P imaging and photostimulation
- Sets which data card channels to use for PMT channels 1 and 2
- Sets which data card channels to use for moving the imaging mirrors
- Sets which data card channels to use for recording mirror feedback
- Sets which data card channels to use for moving the stimulation mirrors
- Sets which digital lines to use for controlling shutters, triggers, toggles, etc.
- Sets appropriate mirror voltage ranges for the microscope vendor
- Sets appropriate spatial scales for the mirrors for the microscope vendor
- Sets laser offset voltages to use for aligning the 2 sets of mirrors
- Sets hardware settings for visual stimulation (optional)
- Configures all of this on the data cards and saves these card handles in "data".

Functions and Scripts > Acquisition > Driving Mirrors and Acquiring

Function: visor_drive_mirrors_and_acquire()
Location: Visor / Software / Acquisition / visor_drive_mirrors_and_acquire.m
Purpose: Runs the mirrors in the requested patterns while organizing acquired data to memory
Inputs: [none]
Outputs: [none] – changes the global "data" variable to reflect the new settings.

Description:

- applies the mirror control signals defined earlier to the boards
- and drives the 2 photon to acquire data
- NOTE: This just starts the boards running.
- Most of the action happens in the callback function, visor_acquire_one_lap() – see below.

What it Does:

- Gets the mirror commands from the global data variable
- Does a safety check to make sure that the mirrors will not break
- Loads the acquisition settings in preparation for acquisition
- Configures for either Raster or Targeted scanning based on whether "is_raster" is set
- Initializes a data matrix to receive and organize the acquired data
- Writes a "time" matrix, timestamping every point that will eventually be saved
- Moves the mirrors into position for scanning
- Records the current board settings in case it needs to give them back to ScanImage, etc.
- Configures the imaging board for fast scanning
- Configures the mirror control board for fast scanning
- Loads up the mirror boards with the scanning waveforms
- Open the shutter
- Do a brief pause to make sure everything is ready
- Send a digital trigger signal to all boards to commence scanning.

Functions and Scripts > Acquisition > Creating a Raster Image Scanpath

Function: visor_create_raster_scanpath()
Purpose: Creates an imaging scan path to sample a raster image
Inputs: [none]
Outputs: [pre_mirror_xy] – vectors for X and Y of which pixels to scan, in order

Description:

- Gets the number of pixels appropriate for the image size
- Generates a raster scan pattern
- Computes the timing needed for the designated sampling rate
- Distributes the time points across the raster scan pattern
- Returns the vectors that implement that pattern in time

Functions and Scripts > Acquisition > Acquiring a Fastscan Lap

Function: visor_acquire_one_lap()
Purpose: Copy one lap of data from the data card into Matlab memory
Inputs: [none]
Outputs: [none] – changes the global "data" variable to reflect the new settings.

What it Does:

- Checks to see if we are done acquiring, and if so finishes acquisition.
- If we are doing raster scanning, it samples a "stripe" (part of the image) from the data card, and stores it in software memory
    - See "visor_get_stripe_from_board()"
- If we are targeted scanning, it retrieves the latest "lap" of data along the targeted scan path from the data card.
    - See "visor_get_data_from_board()"
- This function keeps getting triggered by the data card every time it has enough samples to retrieve. When a set number of laps is met (which is configured at the start of scanning), this function realizes we are done acquiring and stops.

Functions and Scripts > Cell Detection > Circle the Cells

Function: visor_circle_cells()
Purpose: Draw circles or shadings around the cells on an image, visually
Inputs: [none]
Outputs: [none] – changes the global "data" variable to reflect the new settings.

What it Does:

- Check that there are cells to circle
- Get which figure window to draw on
- Load the image data, and the masks of the cell bodies
- Combine the layers into one set of image data and plot
- Turn the image right-side up to match the microscope, etc.

Functions and Scripts > Cell Detection > Click Cells On and Off

Function: visor_toggle_cells()
Purpose: Click on cells to take them out of the scan path, or include them
Inputs:

- Human events that caused this function to be called (mouse click, etc)
- Cell number that was clicked on

Outputs: [none] – changes the global "data" variable to reflect the new settings.

What it Does:

- Get the mouse coordinates where the user clicked (no longer used)
- Get which cell number was clicked
- See if that cell is currently selected or not
- Flip that cell's "selected" state in the global data array
- Color the cell so it is now selected / deselected
- Re-compute the scan path

Functions and Scripts > Targeted Scan Path > Computing Scan Path

Function: visor_generate_scanpath()
Purpose: Given a set of cells, figures out how to scan them all in an optimized order
Inputs: [none] – cell locations are stored in global “data” variable
Outputs: [none] – changes the global “data” variable to reflect the new settings.

What it Does:

- Get the currently selected cells and retrieve their center locations
- Calculate the quasi-optimal order to traverse the cells
- Get optimal cell order in terms of x,y pixel coordinates
- Compute entry and exit points for the cells

Functions and Scripts > Targeted Scan Path > Rasterize the Scan Path

Function: visor_rasterize_scanpath()
Purpose: Given a scan path and a sense of time, distribute the scan path so that more time is spent sampling within the cells than between them.
Inputs: [none] – cell locations are stored in global "data" variable
Outputs:

- Scan path = vector of points that have been distributed in time and space
- In_cell = vector of 1's and 0's for whether each moment of the scan path is in a cell or not
- Cell_id = vector of which cell is targeted at any moment of the scan path

What it Does:

- Get the current scan path
- Start at the first location in the scan path
- See if we are at a cell entrance or exit
- Create a line segment full of a list of x,y points:
    - "dense" if the line segment is in the cell (entrance)
    - "sparse" if the line segment is out of the cell (exit)
- Repeat this for all the line segments that connect the cells
- The result is a unified scan path with all the points included
- Then, compute for which points we are inside cells or not
- And compute which cell (cell id) we are in at each point in the scan path
- Return these values.

Functions and Scripts > Scan Path > Convert Scan Path to Mirror Signals

Function: visor_convert_scanpath_to_mirror_signals()
Purpose: Given a scan path in pixels, turn these into actual mirror commands.
Inputs: x,y vectors of where to scan, in image pixels
Outputs: x,y vectors of where to scan, in mirror commands (voltage signals)

What it Does:

- Get the mirror voltages associated with the image edges
- Re-compute the image to be centered at 0,0 (like mirrors think)
- Convert pixels to voltages
- Scale voltages by the conversion factor for that microscope
- Scale voltages by the optical zoom, etc that the image was collected under
- Shift the voltages by any X or Y offset that might be applied in a GUI
- Invert either X or Y commands depending on stage / microscope alignment
- Return the final mirror voltages' signals in X and Y

Functions and Scripts > Scan Path > Attribute Actual Mirror Positions to Cells

Function: visor_link_scanpath_to_cells_with_mirror_feedback()
Purpose: Given x,y positions of mirrors during the actual scanpath, which cells were targeted when?
Inputs: [implicit in global “data” variable]
Outputs: a vector of cell ids that were targeted at every point in the scan path

What it Does:

- Get the mirror XY feedback vectors that were actually recorded during the lap
- Scale to the range of the feedback vs. actual voltages
- Invert feedback values
- Account for optical zoom
- Convert volts back to pixels
- Shift values to be centered on the image in pixels
- Go through every actual scan path point, and attempt to place it within every 2D cell boundary
- If no cell matches, then that scan path point was not in a cell
- Record a list of cell ids that match every point in the scan path
- Return this vector

Functions and Scripts > Scan Path > Attribute Fluorescence Back to Cells

Function: visor_group_scanpath_results_by_cell()
Purpose: Given when the mirrors were actually in each cell, average the fluorescence sampled and come up with an instantaneous fluorescence for that lap for that cell
Inputs: [implicit in global “data” variable]
Outputs: [implicit in global “data” variable]

What it Does:

- Get the scan path that actually happened, from recorded mirror feedback
- Figure out whether to cut out time on the sides of the cells, etc
- For each cell, get the time moments when the laser was in the cell
- Get the fluorescence recorded during these moments
- Average the fluorescence of the relevant points to get a single value for that lap
- Assign this value to that cell at that time point
- Index the fluorescence for all cells during that lap
- Also index the full frame fluorescence
- Save these values

Functions and Scripts > Stimulation > Aim at a Cell

Function: visor_move_photostim_mirrors_to_position()
Purpose: Aim the stimulation mirrors at a specific cell, while the laser control scripts blink the laser in specific time and intensity patterns
Inputs:

- x = x position of the cell in the image, in pixels
- y = y position of the cell in the image, in pixels
- t = duration to keep the mirrors pointed at the cell during imaging

Outputs: [implicit in global "data" variable]

What it Does:

- Builds vectors for control signals to the stimulation mirrors
- Stops the mirror boards if necessary
- Converts the pixel x,y commands to mirror voltage commands
- Slightly adjusts the x,y position to match the imaging plane, according to calibration offsets
- Take control of the mirrors from the vendor's computer
- Load the control signals onto the mirror board in preparation to move them
- Start the mirror control board to apply the command signals and move the mirrors.
- Note – this moves the stimulation mirrors to a specific position and holds it there using a long, constant string of the same command values. You could just as easily provide a sequence of different values to this vector to move the mirrors to stimulate different sites – see **visor_move_photostim_mirrors_to_position_loop()** below.

Functions and Scripts > Stimulation > Aim at Different Positions

Function: visor_move_photostim_mirrors_to_position_loop()

Purpose: Aims the stimulation mirrors at different places at different times, while the laser control scripts blink the laser in specific time and intensity patterns

Inputs: [none – the positions and timings are created via variables and loops that the user can define from within the function]

Outputs: [implicit in global “data” variable]

What it Does:

- Lets the user define a series of x,y positions that they would like to target
- Lets the user define temporal offsets (t_steps) for moving to those positions
- Uses a similar process as in visor_move_photostim_mirrors_to_position() to control the mirrors and move to specific pixel locations as specified
- Note – this example script moves the stimulation mirrors to specific positions at fixed intervals. However variable intervals could trivially be utilized as well just by using different t_steps.

Functions and Scripts > Stimulation > Driving the Photoactivation Laser

Function: prism_hook_go_evoke_aps()

Purpose: This provides all the description you need to generate arbitrary time and intensity pulses from an optogenetics laser. These laser pulses are then directed at different locations in the network using the mirror steering commands described above. The laser intensity part of the code is from a different software platform than Visor, Prism, that works in exactly the same way (originally for electrophysiological stimulation). All of the code that you need to understand how it works is included in the package. It includes a simple way to specify your time and intensity patterns in Matlab. It then uses a "layered onion" approach to let you look deeper and deeper to understand how the code interacts in a modular and layered fashion, all the way down to controlling the boards that drive the laser.

Inputs: In the script the user can configure (all standard units):

- amplitude – intensity of the photoactivation control pulse, in volts (0-8 V, with 8 V = 100%)
- width – width of the activation pulses, in seconds
- num_pulses – how many pulses to deliver in each sweep (sweep = 1 patter)
- interval – interval between the pulses in a sweep, in seconds
- delay – time to shift the pulses in the sweep before starting num_pulses volley
- num_repeats – number of times to repeat that pulse pattern
- is_led – lets you toggle whether you are controlling a laser, or an LED light source, which in our case had an inverted relationship between command voltage and intensity

Outputs: [implicit in global "data" variable]

What it Does:

- Lets the user define the pulse pattern profile to use in every sweep
- The number of sweeps is set by the num_repeats setting.
- The length of every sweep is specified in this code by what the user has specified for recording (the stimulation "plays along"). Specifically, this is specified in "prism_start_channelrhodopsin_acquisition() (included) which reads out what "horizontal scale" the user has specified for each trace.
- Once these parameters are locked in, the script starts the laser control board and it awaits a start trigger, which comes when either the electrophysiological recording begins (as in Prism), or when imaging recording begins (as in Visor). Using the connections in Table 1, either imaging or electrophysiological recording can trigger the start of the recording + stimulation session, even though recording and stimulation may use different subsystems.

Nested Steps That Can be Re-Used in Your Code:

- All code described here can be found in the folder:

    - All-Optical Software\Key Code Examples\4 - Optical Stimulation\3 - Driving Laser with Time Patterns
- First, the script above takes the parameters that the user has locked in above
- Then, it feeds these into, “prism_channelrhodopsin_regular_pulses()” (included)
    - This turns the waveform parameters into an actual waveform vector (time varying pattern) with which to drive the laser.
    - It then calls “prism_put_channelrhodopsin_pattern()” to put the pattern on the board
        - This loads up the internal variables with the laser driving time pattern, the number of repeats to use, etc.
- Then, it runs prism_start_channelrhodopsin_acquisition() (also included)
    - This sets some more internal variables to specify how to record electrophysiology during optical stimulation
    - This then calls prism_start_data_acquisition() (also included)
    - Which calls prism_initialize_data_acquisition() (also included)
    - These provide good insight into how to drive a stimulation laser through Matlab, using the parameters that you specified above.

Functions and Scripts > Analysis > Removing Photoartifacts

Function: visor_dff_viewer_show_chr2 ()

Purpose: in an 2p recording, blank the photoactivation artifacts
Inputs: [none - implicit in global “data” variable]
Outputs: [none - implicit in global “data” variable]

What it Does:

- Lets the user set a dFF artifact threshold for stimulation events
- Scans the trace for each cell (and in fact each trial, if desired) to identify the dFF artifact time windows
- It checks for positive and negative artifacts
- It then replaces those artifacts with “zeroed” ranges of points that are equal to the local baseline.

Functions and Scripts > Stimulation > Conducting Spiral Stimulation

Function: visor_spiral_stimulation_nw()
Purpose: Drive the mirrors and recording in a spiral pattern about the cells, to fill in the cell area with a 2P stimulation laser, as in Figure 7b, and Rickgauer 2009.
Inputs: [none - implicit in global "data" variable]
Outputs: [none - implicit in global "data" variable]

What it Does:

- Lets the user set a spiral radius to use
- Configures the Visor system for stimulation and recording as usual, except specify that vertices should be scanned using a "spiral" pattern
- Runs visor_drive_mirrors_and_acquire() as defined above
- When the driving happens, it applies a different waveform to the "vertex" of a cell:
    - Normally the cells are scanned using a "line" waveform
    - Here we specify a "spiral" vertex
    - Any vertex spatiotemporal pattern can be defined and used here.
- This then uses the function "visor_get_path_within_cell_spiral()" to generate a spiral at each vertex, in line with the scan path.

Summary:

- These are only some of the most critical components contained in the software package included.
- The package also includes a huge array of tools, from different eras of our development, for:
    - Image management
    - File type management from different microscopes
    - Data analysis
- The file structure is pretty well organized and commented, but please contact us if you have any questions on how things might be done – we would love to hear from you!
- You are welcome to use any bits and pieces that you find here in your own code with or without attribution – the idea is to make any small amount of work we were able to accomplish more shareable and further a spirit of open source for the community.
- Also check our website where more recent versions may be posted.
- Good luck!! With a custom physiology mentality, and more time than we now have, you will do great things.

Controlling a Prairie Microscope with ScanImage Software

Nathan Wilson / Sur Laboratory

Overview:

ScanImage from HHMI Janelia Farm is a powerful software package for doing 2P imaging and learning about the associated principles. Prairie Technologies makes 2P imaging systems that are increasingly used. In working with Prairie microscopes and hacking them for custom control it became apparent that the same configuration could enable any custom software to control the Prairie setup. The steps below are meant to illustrate this and help you on your way in using ScanImage or other software packages with a commercial Prairie microscope. This is instructional / use at your own risk, but we hope you find it helpful in your own work.

About the Integration:

The essence of the integration is to set up a second computer that can "borrow" the Prairie scanners and run alongside the normal Prairie setup. The goal is to not remove any part of the existing Prairie setup, but rather to add an extra control channel to the setup.

Basically what you will do is add a second "auxiliary" line of controls to the mirrors via the galvo box, and then add a toggle command signal that can switch the galvo box to listen to your second computer instead of the original Prairie computer. The Prairie computer remains connected to the galvo box as well, using the original inputs.

Finally, you will attach in (via BNC splitter) a control line from your second computer to the existing existing shutter command line from the Prairie computer. When you scan using your second, custom computer, it will send a "high" signal to the shutter to open it, send a "high" signal to the galvo toggle to tell it to listen to its X/Y mirror commands instead of the Prairie computer's, and then it will do the scanning. When it is done scanning it will send low signals to close the shutter and release the galvos. You could then immediately take a scan with the Prairie computer. Pockels and PMTs on the Prairie are controlled either manually using neither computer, or from the Prairie computer and software – no control lines need to intercept these from the custom computer, though in principle it is of course possible.

Driving a Prairie with ScanImage:

Details of this integration may over time stop making sense after this publication. However, the general principles should remain sound, and we may update our respective websites with new information as ScanImage or Prairie evolves.

1. Perform the steps in the Procedure, ***III. Steps 9-21: Connecting computer to two-photon imaging microscope*** For your Prairie microscope, making the connections described in Figure 2 and Table 2. You will now have a second "custom" computer that can control the Prairie using Visor, etc.

2. Install ScanImage from the Janelia Farm website.

3. In the "Machine Data File Configuration", replicate the relevant settings for PMT inputs, mirror commands, shutter control, and trigger control that are set in the Visor software.

Running ScanImage and Visor at the same time:

Finally, you can also run ScanImage and Visor at the same time – check Visor code that is instantiated during scan start and scan stop under the "Sutter" microscope configuration to see how Visor can gently "borrow" the data acquisition cards from ScanImage, and then "give back" the cards at the end of the scan.