Overview

In this tutorial we are going to walk you through building an online placefield display that interfaces with Cheetah using Matlab. You will be introduced to our NetCom application programming interface (API), and how that API can be used to both monitor and control Cheetah during an experiment. We assume that you have some Matlab programming experience and are familiar with Cheetah.

There will be three phases in this tutorial. First, we will show you how to connect your Matlab script to Cheetah using NetCom and how to plot streaming video tracker locations. In Phase 2, we will add the streaming data from a spike acquisition entity and plot spikes at the location where they occurred. Finally, Phase 3 will use color coding to display which cells are firing and show how to send commands to Cheetah based on analysis of the streaming data.

The Matlab script that results from this tutorial is intended for educational purposes only. While it is a useful script, it takes some shortcuts and glosses over some details that may cause unsatisfactory results when used in a real experiment. These steps were taken to both keep the tutorial short and to focus on learning to control Cheetah using NetCom. This script can be a useful starting point for creating a fully featured placefield display to use with your experiments.

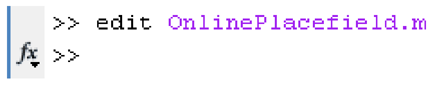
Tutorial Setup

In order to run the scripts in this tutorial, a live Cheetah acquisition system with at least one tetrode and a video tracker is required. The tutorial can be completed without connecting to a live system, but the scripts will not be able to be run. You will also need to have the Matlab NetCom scripts and DLL unzipped into a directory that has been added to Matlab's path. For more information on adding a directory to Matlab's path, see (http://www.neuralynx.com/forum/showthread.php?t=216).

Phase1: Streaming Video Tracker Positions into a Matlab Plot

The script you are about to write will be responsible for connecting to Cheetah, opening a data stream for a video tracker and then plotting the location from each video frame to a Matlab scatter plot.

Now, let's create our script:



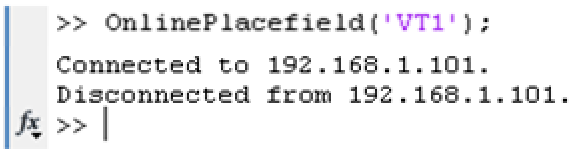
```
>> edit OnlinePlacefield.m
>>
```

In the edit window, type the following script. The comments are in *Green italics* and are intended to provide information as to what the code is doing and are purely instructional. They do not need to be entered into your script.

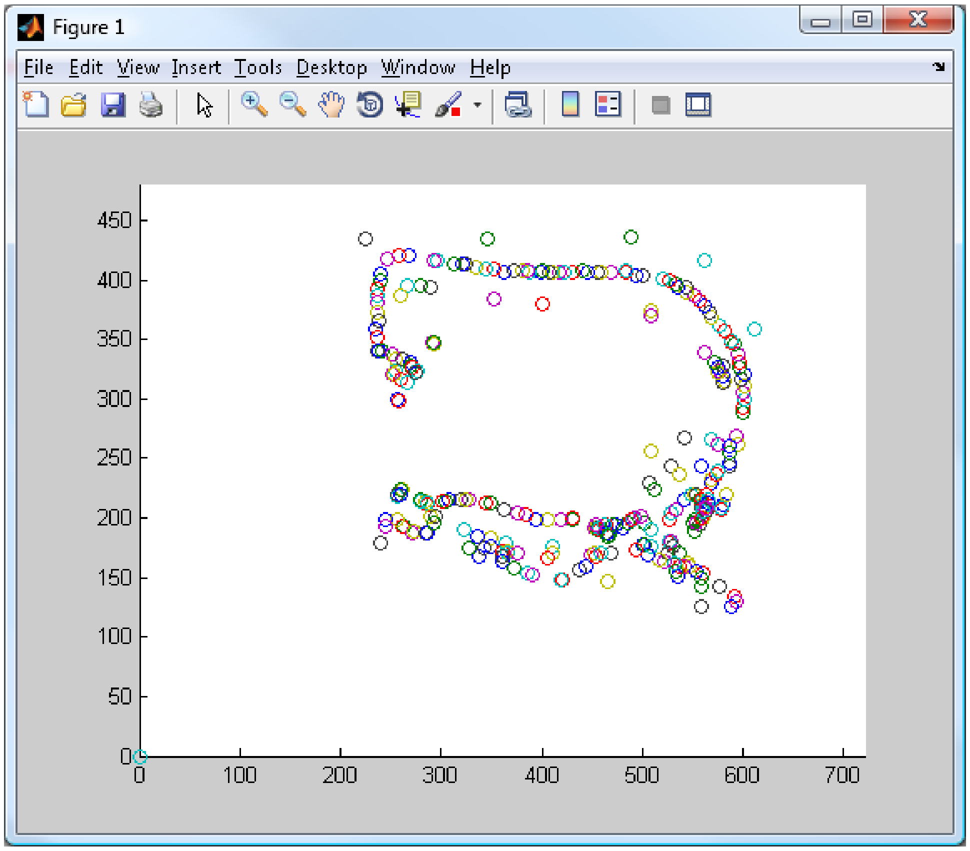
```
function  OnlinePlacefield(vt_acq_ent)

%variables used throughout this script
serverName = '192.168.1.101'; %the server we are going to connect with to get streaming data
                               %Replace this IP with the IP of the computer that is running Cheetah
vtXResolution = 720; %number of pixels of horizontal resolution for the video tracker
vtYResolution = 480; %number of pixels of vertical resolution for the video tracker
numberOfPasses = 10; %the number of times we will poll the server for new data before
                  %disconnecting and exiting the script

%Load NetCom into Matlab, and connect to the NetCom server if we aren’t connected
if NlxAreWeConnected() ~= 1
       succeeded = NlxConnectToServer(serverName);
       %Every NetCom function returns a value indicating if the command succeeded.
       %It is a good practice to check the succeeded value after every function call.
       %To keep the tutorial short, we will not do this for every function.
       if succeeded ~= 1
       fprintf('FAILED connect to %s. Exiting script.\r', serverName);

              return;
       else
              fprintf('Connected to %s.\r', serverName);
       end
end

%Identify this program to the server.
NlxSetApplicationName('Online Placefield Demo in Matlab');

%Get a list of all objects in Cheetah, along with their types.  Then, check to see
%if the acquisition entities we want actually exist in Cheetah.
%It is a good practice to replace unused return values with ~, telling Matlab to
%ignore them.  This both reduces memory usage and makes the program easier to read.
%For a complete list of return values, see the help for the function in question.
[~, cheetahObjects, ~] = NlxGetCheetahObjectsAndTypes();
if isempty(strmatch(vt_acq_ent, cheetahObjects))
       fprintf('FAILED the acquisition entity %s does not exist.\r', vt_acq_ent);

       %You should always disconnect from a server before terminating your script.
       NlxDisconnectFromServer();
       return;
end

%Open the data stream for the VT acquisition entity.  This tells Cheetah to begin %streaming
data for the VT acq ent.
NlxOpenStream(vt_acq_ent);

%setup the figure we are going to use to display the VT locations
figure(1);
clf(1);
axis([ 0, vtXResolution, 0, vtYResolution]);
hold on;

%Poll NetCom until our number of passes has been reached.
for pass = 1:numberOfPasses

       %Pause to let some new data load up in NetCom's buffers.
       pause(1);

       %Request all new VT data that has been acquired since the last pass.
       [~, ~, locationArray, ~, VTRecsReturned, ~] = NlxGetNewVTData(vt_acq_ent);

       %Loop through all of the retrieved data.
       for vtRecordNumber = 1:VTRecsReturned
```

```
                %Get the X and Y location value for this VT frame.  The
                %locationArray is in the form [x1, y1, x2, y2, ... xN, yN].
                xValue = locationArray(2 * vtRecordNumber - 1);
                yValue = locationArray(2 * vtRecordNumber);

                %Plot the VT's location
                scatter(xValue, yValue);
        end
end


%Close the stream for the VT acq ent.  This stops streaming of VT data without
%disconnecting from the server
NlxCloseStream(vt_acq_ent);

%Disconnect from the server.
NlxDisconnectFromServer();
fprintf('Disconnected from %s.\r', serverName);

end
```

Once all the code is entered into the script, save it and close the editor. Next we will launch the script:



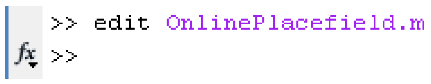
```
>> OnlinePlacefield('VT1');
Connected to 192.168.1.101.
Disconnected from 192.168.1.101.
>>
```



Phase2: Streaming Spikes and Plotting Spikes at VT Locations

Now we will modify the script from Phase 1 to also open up a spike acquisition entity stream. We will then take the spike and video tracker data and make a scatter plot showing where spikes are being triggered according to the video tracker locations.

As in the previous phase, open OnlinePlacefield.m for editing:



```
>> edit OnlinePlacefield.m
>>
```

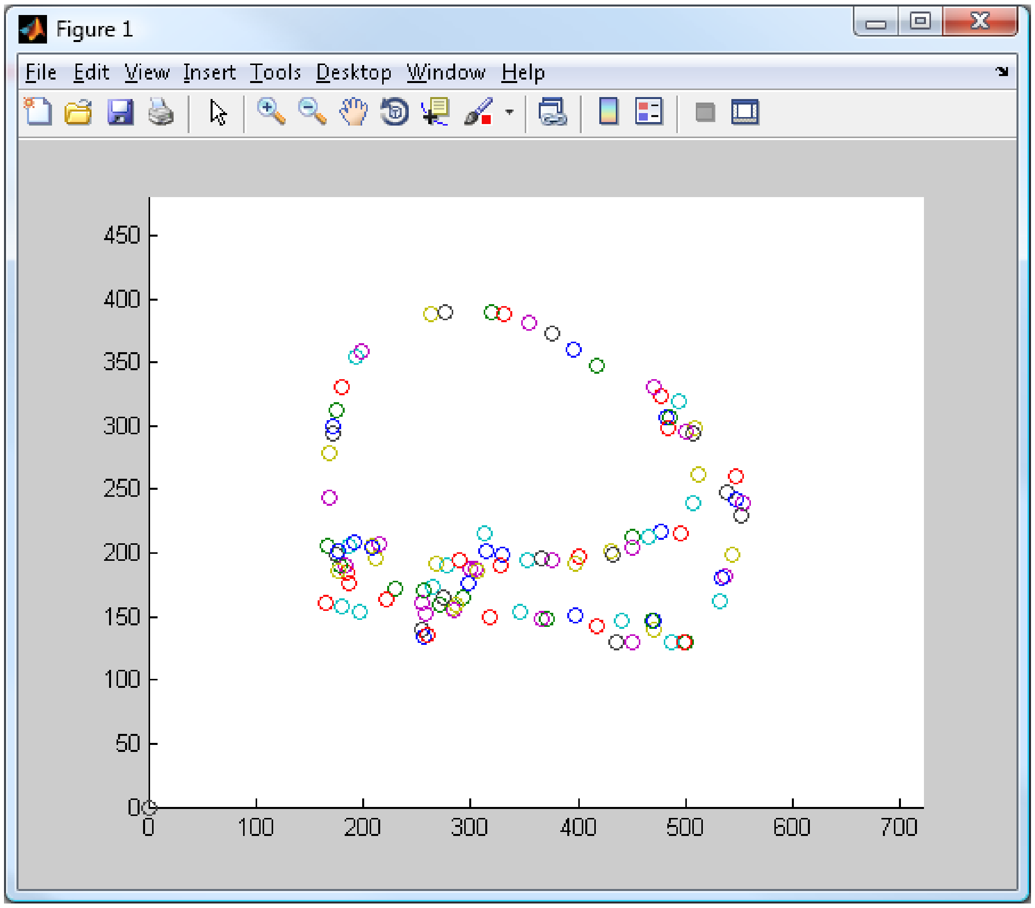
Modify your script to match the following code. All added and changed code is highlighted. All code in normal text is from the previous code listing, and is intended to show where the new code should be added to your script. Some code has not been shown to save space. As before, *green italic* comments describe what the code is doing, but there are now additional comments in underlined red that describe how the code should be written.

```
function  OnlinePlacefield(vt_acq_ent, spike_acq_ent)

%CODE OMITTED TO SAVE SPACE
NlxOpenStream(vt_acq_ent);
%Open the data stream for the spike acq ent
NlxOpenStream(spike_acq_ent);

%CODE OMITTED TO SAVE SPACE
 [~, vtTSArray, locationArray, ~, VTRecsReturned, ~ ] = NlxGetNewVTData(vt_acq_ent);
%Get new spike data
     [~, ~, spikeTSArray, ~, cells, ~, spikeRecsReturned, ~ ] = NlxGetNewTTData(spike_acq_ent);

%Loop through all of the retrieved data.
     for vtRecordNumber = 1:VTRecsReturned  - 1
       %Stopping 1 record early since we are comparing the timestamps
       %of two consecutive records.
       for spikeRecordNumber = 1: spikeRecsReturned
              %Check to see when the spike occurred.  This tells us which VT location
              %corresponds to which spike.  Since VT timestamps correspond
              %to the beginning of a frame, all spikes occurring between two frames are
              %placed at the location of the first of the two frames.
              %The following two lines should be written on a single line, ending in + 1)
            if vtTSArray(vtRecordNumber ) <= spikeTSArray(spikeRecordNumber) &&
spikeTSArray(spikeRecordNumber) < vtTSArray(vtRecordNumber + 1)
                    xValue = locationArray(2 * vtRecordNumber - 1);
                    yValue = locationArray(2 * vtRecordNumber);
```

```
                        scatter(xValue, yValue);
                end
            end
        end
    end
%CODE OMITTED TO SAVE SPACE
NlxCloseStream(vt_acq_ent);
%Close the stream for the spike acq ent
 NlxCloseStream(spike_acq_ent);
%CODE OMITTED TO SAVE SPACE
```

Save the script when you are done entering code, close the editor window and run the script again. This time we will also pass a spike acquisition entity to monitor.

```
>> OnlinePlacefield('VT1', 'TT1');
Connected to 192.168.1.101.
Disconnected from 192.168.1.101.
>>
```



Phase3: Heatmap and Command Responses

Now that we have a functioning placefield, let's make it look a little better by using color to show which cells are firing at each location. In order for this to work during a, you need to define your cells online using Spike Sort 3D. We have already defined a few dummy cells for the purposes of this tutorial.

Add the color coding to your script with the following code:

```
function  OnlinePlacefield(vt_acq_ent, spike_acq_ent)

%CODE OMITTED TO SAVE SPACE
hold on;
cell_colors = [.75 .75 .75; 1 0 0; 0 1 0; 0 0 1]; %the colors to use for each defined cell

%CODE OMITTED TO SAVE SPACE
for pass = 1:numberOfPasses
                    %CODE OMITTED TO SAVE SPACE
                    yValue = locationArray(2 * vtRecordNumber );

                    %all cells numbered >= 3 will be blue and we'll adjust the cell
                    %numbering to be one based.
                    cellNumber = min(cells(spikeRecordNumber), 3) + 1;
                    %adjust the scatter plot to use the correct cell color
                    %using a 50 sized filled circle for the location.
                    scatter(xValue, yValue, 50, cell_colors(cellNumber, :), 'filled');
              end
          end
       end
end
%CODE OMITTED TO SAVE SPACE
```

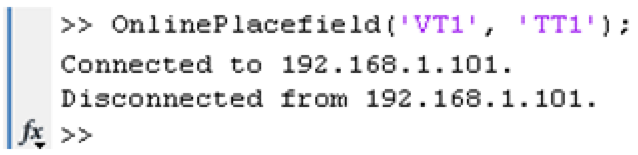
Save the changes to the script, but this time do not exit the editor.

So far, this script has just received data from Cheetah. NetCom can also be used to control Cheetah during an experiment by sending commands. In this example, we will set a threshold for the number of spikes that occur at a single location. Once that threshold is eclipsed, we will send an event to Cheetah indicating how many spikes have been received at that location. A complete list of Cheetah commands is located in the Cheetah Reference Guide which is under the Help menu in Cheetah.

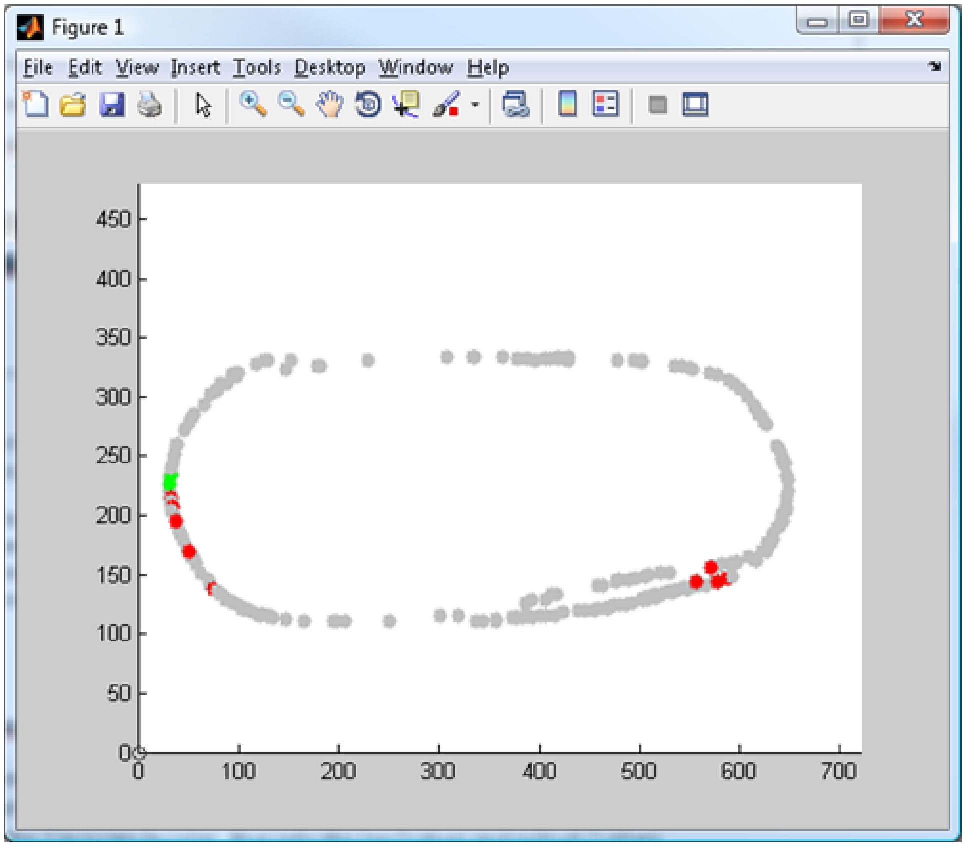
Finally, let's add a command response to our script:

```
function  OnlinePlacefield(vt_acq_ent, spike_acq_ent)
%CODE OMITTED TO SAVE SPACE
numberOfPasses = 10;
eventResponseThreshold = 10; %the number of times that a spike needs to be fired in a
                             %single location in order to trigger an event response
%CODE OMITTED TO SAVE SPACE
cell_colors = [.75 .75 .75; 1 0 0; 0 1 0; 0 0 1];
binnedSpikes = zeros(vtYResolution, vtXResolution); %holds the spike counts at each location
for pass = 1:numberOfPasses
             %CODE OMITTED TO SAVE SPACE
             yValue = locationArray(2 * vtRecordNumber );
             %Since Matlab uses 1 based indices, we are going to
             %adjust all 0 valued coordinates to 1.
             xValue = max(1, xValue);
             yValue = max(1, yValue);
             %Increment the number of spikes that occurred at this location.
             binnedSpikes(yValue, xValue) = binnedSpikes(yValue, xValue  ) + 1;
             %Send an event to Cheetah if a single location has more spikes than the
             %threshold we set.
             if binnedSpikes(yValue, xValue) > eventResponseThreshold
                    %The following two lines should be written on a single line, ending in ;
                    NlxSendCommand(sprintf('-PostEvent "%d spikes at %d,%d" 0
0',binnedSpikes(yValue, xValue), xValue, yValue));
             end

             cellNumber = min(cells(spikeRecordNumber), 3) + 1;


%CODE OMITTED TO SAVE SPACE
```

Now save, close the editor, and run the script one more time.



```
>> OnlinePlacefield('VT1', 'TT1');
Connected to 192.168.1.101.
Disconnected from 192.168.1.101.
>>
```



Conclusion

This tutorial should have given you a good starting point for creating your own Matlab experiment control scripts. To begin working with the Matlab NetCom interface, you will need to download the NetCom Development Package from the Software Downloads page on our website (www.neuralynx.com/software). In that package, you will find all of the NetCom M-files as well as another example showcasing the Matlab NetCom interface. If you have any problems or questions, please check our support page (http://www.neuralynx.com/support) for support options.

Appendix: Complete Code Listing for OnlinePlacefield.m

```
function  OnlinePlacefield(vt_acq_ent, spike_acq_ent)

%variables used throughout this script
serverName = '192.168.1.101'; %the server we are going to connect with to get streaming data
vtXResolution = 720;  %number of pixels of horizontal resolution for the video tracker
vtYResolution = 480;  %number of pixels of vertical resolution for the video tracker
numberOfPasses = 10;  %the number of times we will poll the server for new data before
                      %disconnecting and exiting the script
eventResponseThreshold = 10;  %the number of times that a spike needs to be fired in a
                              %single location in order to trigger an event response


%Load NetCom into Matlab, and connect to the NetCom server if we aren't connected
if NlxAreWeConnected() ~= 1
    succeeded = NlxConnectToServer(serverName);
    %Every NetCom function returns a value indicating if the command succeeded.
    %It is a good practice to check the succeeded value after every function call.
    %To keep the tutorial short, we will not do this for every function.
    if succeeded ~= 1
        fprintf('FAILED connect to %s. Exiting script.\r', serverName);
        return;
    else
        fprintf('Connected to %s.\r', serverName);
    end
end

%Identify this program to the server.
NlxSetApplicationName('Online Placefield Demo in Matlab');

%Get a list of all objects in Cheetah, along with their types.  Then, check to see
%if the acquisition entities we want actually exist in Cheetah.
%It is a good practice to replace unused return values with ~, telling Matlab to
%ignore them.  This both reduces memory usage and makes the program easier to read.
%For a complete list of return values, see the help for the function in question.
[~, cheetahObjects, ~] = NlxGetCheetahObjectsAndTypes();
if isempty(strmatch(vt_acq_ent, cheetahObjects)) || isempty(strmatch(spike_acq_ent,
cheetahObjects))
       fprintf('FAILED the acquisition entity %s or %s does not exist.\r', vt_acq_ent,
spike_acq_ent);

       %You should always disconnect from a server before terminating your script.
       NlxDisconnectFromServer();
       return;
end

%Open the data stream for the VT acquisition entity.  This tells Cheetah to begin
%streaming data for the VT acq ent.
NlxOpenStream(vt_acq_ent);
%Open the data stream for the spike acq ent
NlxOpenStream(spike_acq_ent);
```

```
%setup the figure we are going to use to display the VT locations
figure(1);
clf(1);
axis([ 0, vtXResolution, 0, vtYResolution]);
hold on;
cell_colors = [.75 .75 .75; 1 0 0; 0 1 0; 0 0 1]; %the colors to use for each defined cell
binnedSpikes = zeros(vtYResolution, vtXResolution); %holds the spike counts at each location

 %Poll NetCom until our number of passes has been reached.
 for pass = 1:numberOfPasses

    %Pause to let some new data load up in NetCom's buffers.
    pause(1);

    %Request all new VT data that has been acquired since the last pass.
    [~, vtTSArray, locationArray, ~, VTRecsReturned, ~] = NlxGetNewVTData(vt_acq_ent);
    %Get new spike data
    [~, ~, spikeTSArray, ~, cells, ~, spikeRecsReturned, ~] = NlxGetNewTTData(spike_acq_ent);

       %Loop through all of the retrieved data.
       for vtRecordNumber = 1:VTRecsReturned - 1
        %stopping 1 record early since we are comparing the timestamps
        %of two consecutive records
        for spikeRecordNumber = 1:spikeRecsReturned
            %Check to see when the spike occurred.  This tells us which VT location
            %corresponds to which spike.  Since VT timestamps correspond
            %to the beginning of a frame, all spikes occurring between two frames are
            %placed at the location of the first of the two frames.
            if vtTSArray(vtRecordNumber ) <= spikeTSArray(spikeRecordNumber) &&
spikeTSArray(spikeRecordNumber) < vtTSArray(vtRecordNumber + 1)

                %Get the X and Y location value for this VT frame.  The
                %locationArray is in the form [x1, y1, x2, y2, ... xN, yN].
                xValue = locationArray(2 * vtRecordNumber - 1);
                yValue = locationArray(2 * vtRecordNumber);

                %Since Matlab uses 1 based indices, we are going to
                %adjust all 0 valued coordinates to 1.
                xValue = max(1, xValue);
                yValue = max(1, yValue);
                %Increment the number of spikes that occurred at this location.
                binnedSpikes(yValue, xValue) = binnedSpikes(yValue, xValue  ) + 1;
                %Send an event to Cheetah if a single location has more spikes than the
                %threshold we set.
                if binnedSpikes(yValue, xValue) > eventResponseThreshold
                    NlxSendCommand(sprintf('-PostEvent "%d spikes at %d,%d" 0
0',binnedSpikes(yValue, xValue), xValue, yValue));
                end

                %all cells numbered >= 3 will be blue and we'll adjust the cell
                %numbering to be one based.
                cellNumber = min(cells(spikeRecordNumber), 3) + 1;
                %adjust the scatter plot to use the correct cell color
                %using a 50 sized filled circle for the location.
                scatter(xValue, yValue, 50, cell_colors(cellNumber, :), 'filled');

            end
        end
     end
  end
```

```
%Close the stream for the VT acq ent.  This stops streaming of VT data without
%disconnecting from the server
NlxCloseStream(vt_acq_ent);
%Close the stream for the spike acq ent
NlxCloseStream(spike_acq_ent);

%Disconnect from the server.
NlxDisconnectFromServer();
fprintf('Disconnected from %s.\r', serverName);

end
```