This is an overview of the data analysis software to track urine marks, as reported in Keller, Stowers et al., Nature Neuroscience, 2018. As described in the paper, we use cameras above and below to track mice and unobstructed urine marks, and the MATLAB scripts track the urinated pixels and spots aver time. We used Adobe After Effects to crop, trim, and render videos, but this can be done with much simpler software or even automated with better hardware controls. The top level MATLAB script (“smufAnalysisTopLevelScript.m”) then takes a rendered MP4 (other formats are possible with the VideoReader class in MATLAB) video filepath as input, and produces MAT files with extracted data matrices by calling the “smufVideoToPixels.m” and “smufMakeEum.m” functions.

**MATLAB toolboxes used**: image processing, statistics

**smufAnalysisTopLevelScript** – this is the top level script to analyze urine marking. It calls "smufMakeEum" and "smufVideoToPixels" functions, groups data,, calculates statistics, and makes various plots. Steps are run independently by setting the following PROGRAM CONTROL options, as well as using the appropriate data filenames specified in the “mouseNums”, “matNameSaveSummary”, and “groupsToCompare” cell arrays. This allows flexibly combining data in different ways. Different plots are controlled by uncommenting them in blocks in the final “plotData” section.

findROIs = 1; % first: manual step; do this only, manually excluding stimulus spots; then check EUM outputs

findUrine = 0; % second: automated step; run and then QC output videos

combineMice = 0; % third, make groups to compare as necessary

computeStats = 0; % fourth, compute stats for group

plotData = 0; % fifth, plot data - different plots are manually controlled by uncommenting them; this should be run along with the computeStats option normally

**smufMakeEum** – this function takes the last few frames of video data and creates a “end urine map” or EUM, which is later searched to find urinated pixels when processing the rest of the video. It allows manual selection of ROIs to exclude from analysis (i.e. stimulus marks), since we do not spectrally or otherwise separate tonic water or semale stimuli. It is possible to build a setup to always make these appear in the same place, and thus automate this step, but it is also useful to have flexibility in where to pipette stimuli when an animal is moving around. Running only this step on all videos first allows the more time-consuming urine detection across all frames to be done unsupervised in batches. Note that a few parameters are hard coded in this function and “smufVideoToPixels”, which should be changed together when setting up behavior for the first time. They set the spatial limits of the video file (thus minimizing the search space), and the single threshold parameter for urine detection, which will vary according to the particular lighting and camera conditions. Note also that the MATLAB imsharpen() function is used to sharpen urine borders in the combined red and green channels for urine detection. We used RGB webcams, but note that a monochrome camera with a green filter can also capture urine fluorescence, and would only require small changes to the code here, such as using a simpler threshold. Alternatively, noisier data (ex. lighting conditions are not controlled well) may benefit from an adaptive threshold or more preprocessing, and it is up to the user to quality control the results by comparing output files against raw video. However, once this is done for a particular setup, the code should be robust to changes across mice/experiments.

**smufVideoToPixels** – this is the main function that takes a video and EUM and searches for new urine spots in each frame, using simple thresholding (after sharpen filter preprocessing) and connected components. The EUM defines the possible pixels to search, and the search space shrinks as pixels are detected. The number of individual urine spots (in the connected components sense) are also tracked, and a spot is decided to be filled in the EUM when it reaches the “spotOverlap” fraction (I used 0.5). This fraction is the percentage overlap needed before marking a spot as filled, and is a rather arbitrary tradeoff between speed of detection and robustness. An output video with overlay of pixels detected or spots detected is written for quality control.