PFI SIMULATOR NOTES

Peter Mao

2018 Mar 21

MOTOR MAPS

- xml files store the map in units of [deg/step]
- for simulation purposes, I prefer units of [step/bin]
  - $\mathrm{map_{sim}}\left[\frac{\mathrm{step}}{\mathrm{bin}}\right] = \dfrac{3.6\left[\frac{\mathrm{deg}}{\mathrm{bin}}\right]}{\mathrm{map_{xml}}\left[\frac{\mathrm{deg}}{\mathrm{step}}\right]}$
  - this conversion takes place in defineBenchGeometry.m
- Life is easier if you work with the cumulative sum of the motor map
  - let cmap = [0, cumsum(map_sim)] generateTrajectory, line 55
  - now cmap tells you the number of steps required to reach the far side of each 3.6 degree bin.
- To work in angles, make a angle array that matches the binwidths of the motor map
  - ang_tht = linspace(0, nbins*binwidth, nbins+1)
  - ang_phi = linspace(0, nbins*binwidth, nbins+1) – pi generateTrajectory, line 64
- Now you can use linear interpolation to go translate from steps to angles or vice-versa through the motor map.
  - in emacs, I use "find-grep" on "interp1" to find all instances of linear interpolation in the matlab code.

NOISE

- I use the α, β model to simulate the stochasticity of the motors.
  - α is the 1-sigma error of a 1-radian move. [radians]
  - β is the error propagation exponent (classically, $\beta = \frac{1}{2}$). Dimensionless.
- I take alpha and beta to be the same for all cobras and for all angles, but this is a gross simplification.
- The fractional bin error is used for applying the noise model to motor maps.
- $f_{\text{BinError}} = \frac{\alpha \Delta\Psi^{\beta}}{\Delta\Psi} = \alpha \Delta\Psi^{\beta-1}$
  - $\Delta\Psi$ is the binwidth. [radians]

APPLYING NOISE TO MAPS

- The motor map tells us how many steps we need to get through a given angular bin.
- Stochasticity in the motors means that the actual displacement per step differs from the motor map prediction.
- Noise is added to the maps by determining how far we go with a given # of steps, but then using that to determine how many steps we need to traverse the bin.
- WARNING: Directly calculating the # noisy steps needed to traverse the bin gives a result that is biased towards a large # of steps!
- If the RNG moves the motor through the bin without using all of the allocated steps, then the number required to traverse the bin is recorded as the noisy map value
- If the RNG only moves the motor part way through, then another move is simulated (with the number of steps to move through the remainder of the bin).
- The second "if" above is repeated until the condition of the first "if" is achieved.

subroutine in addNoise.m:

```
function output = mapFactor(fBE,mapSize)
% determines the map error factor given fractional bin error and
% the size of the map (fibers x bins) based on one estimate of the
% number of steps to get through a bin, per bin.

%fBE = .1; %fractional bin error. realistic is ~0.25
%mapSize = [10000 100]; % 2048x100 for tht

    XX = zeros(1,prod(mapSize)); % distance travelled
    MF = zeros(1,prod(mapSize)); % map multiplication factor
    ctr = 0;

    while numel(XX) ~= sum(sum(XX))
        ctr = ctr + 1;
        moveMe = (XX < 1); % logical of map cells to move

        Xremaining = 1 - XX;

        dx = inf(size(XX));
        dx(moveMe) = randn(1,sum(moveMe)) * fBE + 1;

        Xend = XX + dx; % # bins moved through

        done = Xend >= 1; % bin is done when total travel > 1

        fracMove = Xremaining./dx; % fraction of commanded steps used to reach
                                   % the end of the bin.

        fracMove(~done & moveMe) = 1; % if it didn't make it to the end, then
                                      % all steps are assigned to this
                                      % bin.
        MF = MF + fracMove;
        XX = min(Xend,1);

        X(ctr,:) = XX; % X holds the move history

    end
    MF = reshape(MF,mapSize); % ultimately this multiplies against the
                              % inverse motor map (steps/bin)
    output = MF;
end
```

NOTES ON REVISED FUNCTIONS

- generateTrajectory2:
  - calculates the same sense and opposite sense theta trajectories
  - makes no assumptions about relative timing of moves
  - trajectories are all different lengths
- realizeTrajectory2:
  - picks or is told which trajectory to use
  - picks or is told whether to start a motor early or late
  - produces a trajectory matrix with axes (cobraID, time) and values of angle.
- generateTrajectory:
  - not currently used in simFun.m
  - calculates a trajectory given start and end positions.
  - timing strategy
    - radially outward: theta early, phi late
    - radially inward: theta late, phi early