PSL432 Theoretical Physiology
Department of Physiology
University of Toronto

ASSIGNMENT 1

Assigned 30 January 2024
Due 13 February 2024

This assignment is worth 25% of your final mark in the course. Please email your submission, in the form of an m-file, to douglas.tweed@utoronto.ca and ankit.roy@mail.utoronto.ca by 11:59 p.m. on 13 Feb. Your code will be graded partly on concision and clarity, so keep it brief and orderly, and make your m-file self-contained so Ankit and I can run it in Matlab on its own — that is, don't write any other files that are called from it.

Your job is to simulate a simplified, two-joint arm and design a policy with Hurwitz desired dynamics and internal feedback that makes the arm reach to and track a moving target.

In continuous time, the arm's dynamics (apart from joint-bounding, described below) are described by the differential equation,

$$\begin{aligned}\tau &= M(q)\,\ddot{q} + \Gamma(q,\dot{q})\,\dot{q} \\ &= \begin{bmatrix} \psi_1 + 2\psi_2\cos(q_2) & \psi_3 + \psi_2\cos(q_2) \\ \psi_3 + \psi_2\cos(q_2) & \psi_3 \end{bmatrix}\ddot{q} + \psi_2\sin(q_2)\begin{bmatrix} -\dot{q}_2 & -(\dot{q}_1 + \dot{q}_2) \\ \dot{q}_1 & 0 \end{bmatrix}\dot{q},\end{aligned}$$

where q is the configuration vector containing the shoulder and elbow angles in that order, $\dot{q}$ and $\ddot{q}$ are velocity and acceleration, τ is the vector of torques at the two joints, and ψ_1, ψ_2, and ψ_3 are constants: $\psi_1 = 1.88$, $\psi_2 = 0.6$, and $\psi_3 = 0.48$.

The torque vector τ itself depends on the neural command or action according to the equation,

$$\dot{\tau} = a - \varsigma\tau,$$

where $\zeta = 0.75$.

In your code, use the name `torque` for the τ vector, and `zeta` for ζ, and integrate the dynamics using Euler's method, with $\Delta t = 0.01$ s.

You should also *bound* the joint angles to keep them inside their motion ranges. That is, define `q_min = [-2; 0]`, `q_max = [1.5; 2.5]`, and after Euler integration write:

```
q = max(q_min, min(q_max, q));
q_vel = q_vel - q_vel.*( (q == q_max).*(q_vel > 0) + …
                         (q == q_min).*(q_vel < 0) );
```

Program the arm's reaching target (the desired joint-angle vector q^*) so that it jumps, every 1 s, to a new, random location inside the arm's joint ranges *and then glides* at a random, constant velocity between jumps, with speeds ranging between 0 and about 2.5 to 3.5 rad/s, always staying inside the joint ranges, as shown by the dotted lines in the plot on the next page.

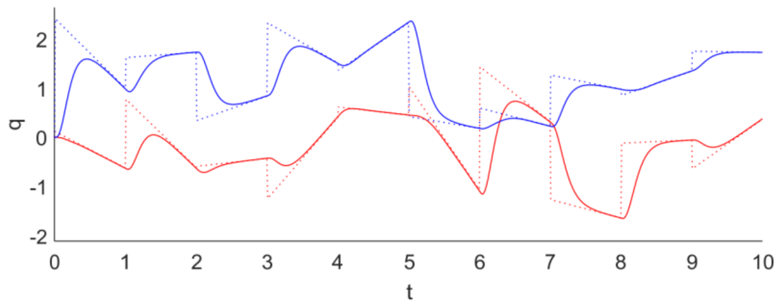
Choose Hurwitz desired dynamics that get both joints to their target values within about 0.75 s or quicker and then keep them on the gliding target until the next jump. Design a policy that implements those desired dynamics and keeps both elements of a smaller than about 25,000 in absolute value (you needn't prove mathematically that a never crosses those bounds — just look at your simulations to check that it usually stays in about that range).

Your agent must rely on *internal* feedback. The only things it can know directly are its own action a, the position and velocity of the target, and the *forms* of the dynamics equations including the joint bounds and where the various constants and variables (ψ_1, ψ_2, ψ_3, q, $q^{(1)}$, a and so on) plug into those equations. Give your agent exactly correct estimates of ψ_1, ψ_2, ψ_3, and ζ, and have it use internal simulations to estimate q and $q^{(1)}$ and any other necessary variables. Give the agent correct initial estimates of those variables, and reset those estimates to their correct values every time the target jumps. You can use the sample code Saccadic_internal.m as a partial guide, but your agent should not assume that actual velocity equals desired velocity.

The agent doesn't know anything else about the *target* except its current position and velocity. For instance, it doesn't know that the target will jump at 1-s intervals or maintain a constant velocity between jumps.

In the handout notes, we derived policies using scalar operations, but here you will need vectors and matrices, because *q* has more than one element. For vector and matrix operations anywhere in your code, use Matlab's built-in functions, e.g. if you need to multiply *A* and *B* or invert or transpose *A*, then write `A*B` or `inv(A)` or `A'` — *don't* write out the complicated, element-wise formulae for these operations.

Display your results in a figure with 2 panels. In the upper panel, plot shoulder and elbow angles (as red and blue solid lines) and their targets (as red and blue dotted lines) versus time, with y-axis limits set using `ylim(1.05*[min(q_min), max(q_max)])`. In the lower panel, plot both elements of *a* versus time. Simulate 10 s in all, or in other words 10 reaches to 10 targets. If all goes well, your upper-panel plot should look something like this:



Please name your variables as in these pages, the notes, and the sample code: `q`, `q_star`, `q_vel`, `q_acc`, `psi`, `zeta`, `M`, `GAMMA`, `a`, and so on. Submit an m-file called Yourgivennameinitial_Yoursurname_A1, as for example D_Tweed_A1.m.