Lecture 20, Oct 22, 2025

Spline Interpolation

- Given $q^0, \dots, q^N \in \mathbb{R}^n$ and user-specified times $t_1, \dots, t_N$ where $t_i < t_{i+1}$, the goal is to find a twice-differentiable $q(t)$ such that $q(0) = q^0$ and $q(t_i) = q_i$
  - This is a method to smooth out the trajectories we get from path planning algorithms
- In 1 dimension, a *cubic spline* is a collection of cubic polynomials $P_i(t) = a_3^i t^3 + a_2^i t^2 + a_1^i t + a_0^i, t \in [t_i, t_{i+1}]$ for segments $i = 0, \dots, N-1$
  - There are $4N$ unknowns, $(a_3^i, a_2^i, a_1^i, a_0^i)$
- Each piece of the spline is constrained by the following:
  1. Interpolation constraints: $P_i(t_i) = q^i$ for $i = 0, 1, \dots, N-1$ (all intermediate points) and $P_{N-1}(t_N) = q^N$ (for the final endpoint)
  2. Continuity constraint: $P_i(t_{i+1}) = P_{i+1}(t_{i+1})$ for $i = 0, 1, \dots, N-2$
  3. Differentiability constraint: $\dot{P}_i(t_{i+1}) = \dot{P}_{i+1}(t_{i+1})$ for $i = 0, 1, \dots, N-2$
  4. Twice-differentiability constraint: $\ddot{P}_i(t_{i+1}) = \ddot{P}_{i+1}(t_{i+1})$ for $i = 0, 1, \dots, N-2$
- In total we have $N+1$ constraints from interpolation, and $3(N-1)$ constraints from continuity, giving $4N-2$ constraints
  - To get the same number of constraints as unknowns, we add the constraints that in the beginning and end of motion, the robot has acceleration of 0
  - This translates to $\ddot{P}_0(t_0) = \ddot{P}_{N-1}(t_N) = 0$
- Now we can solve for all the $(a_3^i, a_2^i, a_1^i, a_0^i)$ by solving a linear system
  - Notice that our spline is linear in the parameters, and all constraints are also linear
  - In the end we get a linear system in the form $Ax = b$, very easy to solve
  - e.g. the first constraint translates to $\begin{bmatrix} t_i^3 & t_i^2 & t_i & 1 \end{bmatrix} \begin{bmatrix} a_3^i \\ a_2^i \\ a_1^i \\ a_0^i \end{bmatrix} = q^i$ and so on

Independent Joint Control (Decentralized Robot Control)

- To actually execute the motion, we need to track a reference signal $q^r(t)$, such that $e(t) = q^r(t) - q(t) \to 0$ as $t \to \infty$ by generating inputs $U(t)$
  - One way to do this is to fully model the dynamics of the robot, which is used for high-precision manipulators or large manipulators like industrial robots (*computed torque control*)
    - $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = U$ where $M(q)$ is mass, $C(q, \dot{q})$ are the Coriolis forces, and $G(q)$ are the gravitational forces
    - This also needs to incorporate the motor models
  - However, a much easier way to achieve this for low-fidelity designs is to use only the motor model, and treat the physics as a disturbance (*independent joint control*)
    - This is known as independent joint control since it controls each joint independently and considers all interactions between them to be disturbances
- Formally, given a twice-differentiable reference signal $q^r(t) = \begin{bmatrix} q_1^r(t) & \cdots & q_n^r(t) \end{bmatrix}^T \in \mathbb{R}^n$, the control problem involves finding feedback control inputs $u_1, \dots, u_n$ to each joint motor, such that $q_i(t) \to q_i^r(t)$ as $t \to \infty$ with desired properties
- We will restrict ourselves to revolute joints for simplicity
- The canonical motor model is $J_m \ddot{\theta}_m + \left(B_m + \frac{k_m k_b}{R} \right) \dot{\theta}_m = \frac{k_m}{R} v - \tau_l$
  - θ_m is the angle of the motor and τ_l is an applied load
    - This contains terms for back EMF, applied load, voltage, etc
  - Let $J = J_m, B = B_m + \frac{k_m k_b}{R}$ and rescaled input $u = \frac{k_m}{R} v$
  - The simplified model is $J\ddot{\theta}_m + B\dot{\theta}_m = u - \tau_l$, which is all we need
- For simplicity, assume that $\theta_m = q \in \mathbb{R}$ for each joint (note there are often offsets, and for a higher fidelity model, there are effects such as backlash and spring/flexibility terms); and also assume $\tau_l = 0$

(so loads are treated as disturbances)

- For a single joint, let $e(t) = q^r(t) - q(t)$
  - Take derivatives: $\dot{e}(t) = \dot{q}^r(t) - \dot{q}(t) \implies \ddot{e}(t) = \ddot{q}^r(t) - \ddot{q}(t) = \ddot{q}^r - \left(-\frac{B}{J}\dot{q} + \frac{1}{J}u\right)$
  - $\ddot{e} = \ddot{q}^r + \frac{B}{J}\dot{q} - \frac{1}{J}u = \ddot{q}^r + \frac{B}{J}\left(\dot{q}^r - \dot{e}\right) - \frac{1}{J}u$
  - $\ddot{e} + \frac{B}{J}\dot{e} = \ddot{q}^r + \frac{B}{J}\dot{q}^r - \frac{1}{J}u$
    * This is a second-order system, and if we didn't have input, we can see that there is a zero eigenvalue, so the system is unstable
- Using a PD controller: $u = J\ddot{q}^r + B\dot{q}^r + K_p e + K_d \dot{e}$
  - The first terms, $J\ddot{q}^r + B\dot{q}^r$, is a *feedforward* signal that cancels the $\ddot{q}^r$ in our equation; this ensures that even when the (position) error is zero, we still drive the motors enough to achieve the desired velocity and acceleration
  - Substituting this we get: $\ddot{e} + \left(\frac{B}{J} + \frac{K_D}{J}\right)\dot{e} + \frac{K_p}{J}e = 0$
  - Now by choosing K_p and K_d, we can place the poles of this second-order system anywhere we want, using classical control methods