The linear quadratic regulator (LQR controller)

summary: robot dynamics, LQR

a linear quadratic regulator has the form of a *state feedback controller* with gain *K:*

$$\boldsymbol{u} = -K\boldsymbol{q}$$

it minimizes the quadratic (scalar) cost function given by

$$J = \int_0^\infty (\boldsymbol{q}^T Q \boldsymbol{q} + \boldsymbol{u}^T R \boldsymbol{u}) dt$$

Where $Q \geq 0$ and $R > 0$ are symmetric matrices. This approach shifts the design problem from gain choices to weight choices

---

dynamics with pure vectors

$$\Sigma \boldsymbol{f} = m\dot{\boldsymbol{v}}$$

$$\Sigma \boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}$$

dynamics for simulation

$$\Sigma \boldsymbol{f} = m\dot{\boldsymbol{v}}$$ **world coordinates**

$$\Sigma \boldsymbol{\tau}' = \mathbf{J}\dot{\boldsymbol{\omega}}' + \boldsymbol{\omega}' \times \mathbf{J}\boldsymbol{\omega}'$$ **body-attached coordinates**

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}'^{\times}$$ **orientation**

$$\dot{\boldsymbol{p}} = \boldsymbol{v}$$ **world coordinates**

remarks on linear quadratic regulator

- computing *K* requires solving "algebraic riccati equation"
  - tricky to solve & requires numerical iteration ⇒ best to use software from experts

    $$K = -R^{-1}B^TP$$
    $$0 = PA + A^TP - PBR^{-1}B^TP + Q$$

    `ct.lqr(A,B,Q,R)` or MATLAB `lqr(A,B,Q,R)`
- LQR is a special case of general optimization problem: find ***u*** that minimizes given cost function and satisfies constraints (e.g. max throttle). This can be used to guide system to desired state/trajectory
- sketch for how LQR is solved:
  - use pontryagin's maximum principle (variational calculus)
  - in special case of linear, time-invariant system, quadratic cost, and infinite time horizon, result is that input is a linear function of state: ***u***=-*K**q***
- for control far away from equilibrium (e.g. aggressive maneuvers), need full nonlinear *trajectory tracking*
  - common "engineering" approach is receding horizon control (a.k.a. "model predictive control"): repeatedly calculate optimal ***u*** over a short horizon
  - biology-inspired: explore the solution space, reward if success (reinforcement learning). parameterize controller/"policy" with a neural network

more remarks

- Full 3D flight control requires two separate, parallel 2D controllers. Also required are coordinate rotations between inner and outer loops.
    - See Fuller2019, "Four Wings: An Insect-Sized Aerial Robot With Steering ability and payload capacity for autonomy", *Robotics and Automation Letters* (2019) on the course web page for one approach.
- Simulating $\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}'^{\times}$ in 3D leads to ill-formed R matrices because of numerical inaccuracy. Better to parameterize R with Euler Angles or quaternions
    - Euler Angles: see Mellinger2012: "Trajectory generation and control for precise aggressive maneuvers with quadrotors" *Int. J. Robot. Res.* on course we page
- To compensate for steady-state disturbances, e.g. torque bias that we must correct for, do *integral action* by adding a state z:

$$\frac{d}{dt}\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ y - r \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ Cx - r \end{bmatrix}$$

← integral of (output) error

$$\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x} - \boldsymbol{K}_i\boldsymbol{z}$$