CDS 110/ChE 105: Lecture 6-1
Trajectory Generation

Richard M. Murray

6 May 2024

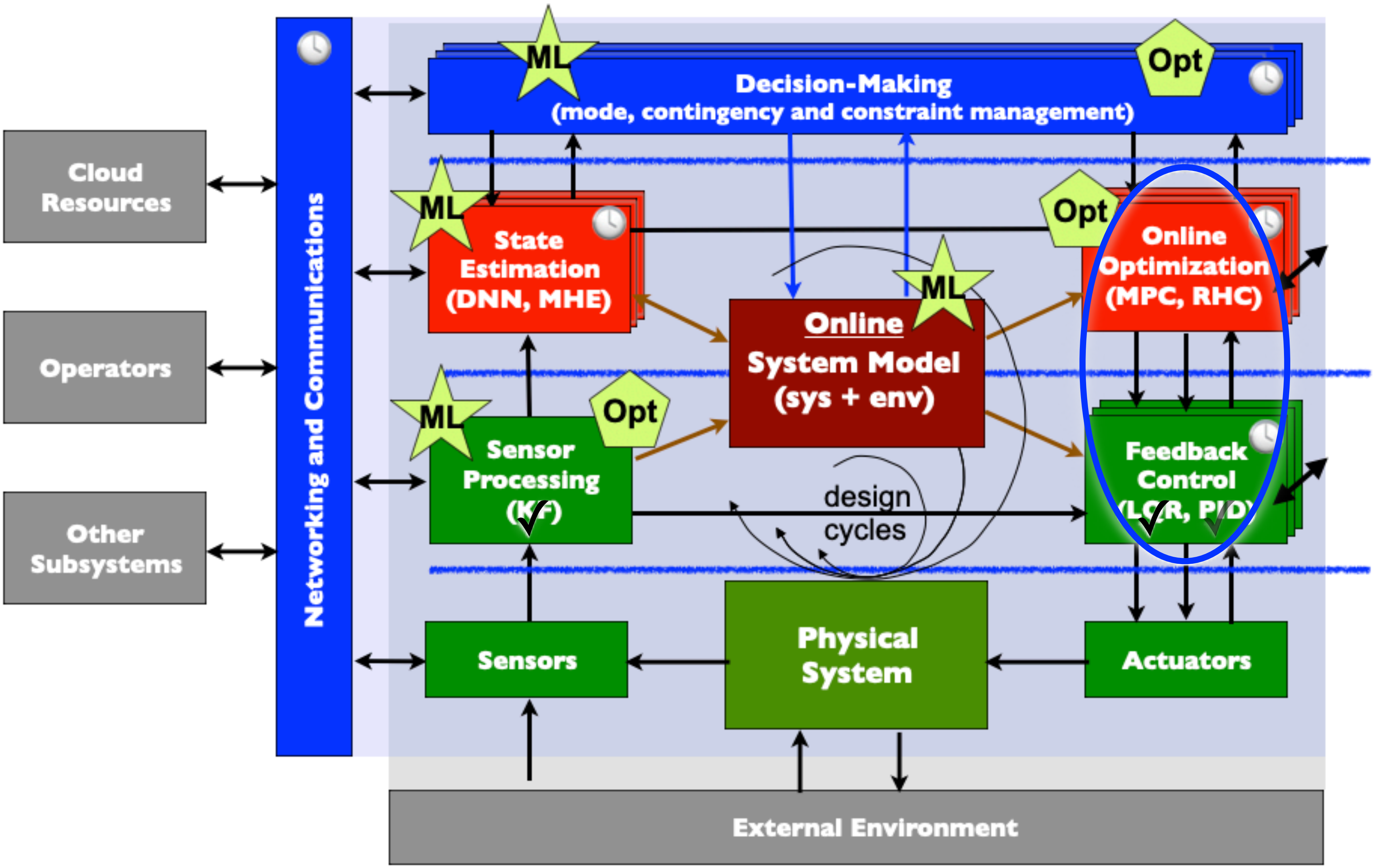
Goals:

- Introduce two-degree of freedom control design (feedforward + feedback)
- (Re-) Introduce reachability, in the context of trajectory generation
- Describe methods for generation of feasible trajectories (w/ Python code)

Reading:

- Åström and Murray, *Feedback Systems,* Sections 7.1 (reachability), 8.5
- Murray, *Optimization-Based Control,* Chapter 1 (overview), Section 2.1
- Google Colab: L6-1_kincar-trajgen.ipynb

Review: Multi-Layer Control Systems

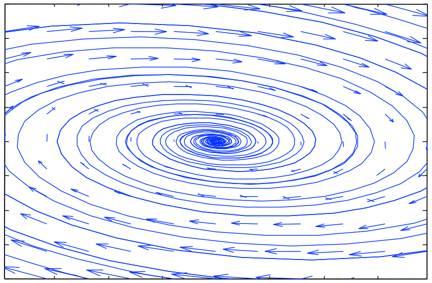


State Space Control Design Concepts (L4-1)

System description: single input, single output system (MIMO also OK)

$$\dot{x} = f(x,u) \qquad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = h(x,u) \qquad u \in \mathbb{R},\ y \in \mathbb{R}$$

Stability: stabilize the system around an equilibrium point

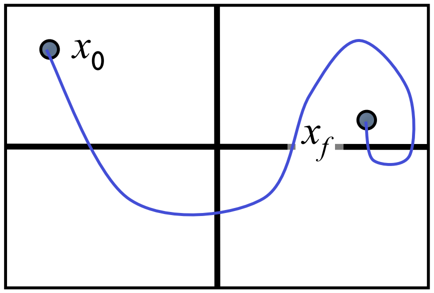


- Given equilibrium point $x_e \in \mathbb{R}^n$, find control "law" $u = \alpha(x)$ such that

$$\lim_{t\to\infty} x(t) = x_e \text{ for all } x(0) \in \mathbb{R}^n$$

- Often choose x_e so that $y_e = h(x_e)$ has desired value r (constant)

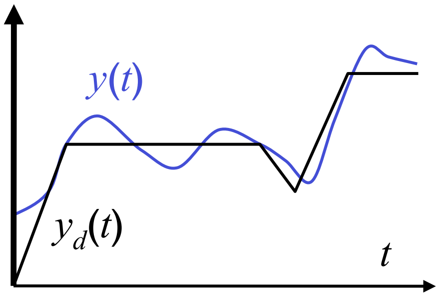
Reachability: steer the system between two points



- Given $x_o, x_f \in \mathbb{R}^n$, find an input $u(t)$ such that

$$\dot{x} = f(x, u(t)) \text{ takes } x(t_0) = x_0 \to x(T) = x_f$$

Tracking: track a given output trajectory



- Given $r = y_d(t)$, find $u = \alpha(x,t)$ such that

$$\lim_{t\to\infty} \big(y(t) - y_d(t)\big) = 0 \text{ for all } x(0) \in \mathbb{R}^n$$

Real-Time Trajectory Generation

Goal: find a feasible trajectory that satisfies dynamics/constraints

$$\min J = \int_{t_0}^{T} q(x,u)\,dt + V(x(T),u(T))$$

$$\dot{x} = f(x,u) \qquad\qquad lb \le g(x,u) \le ub$$

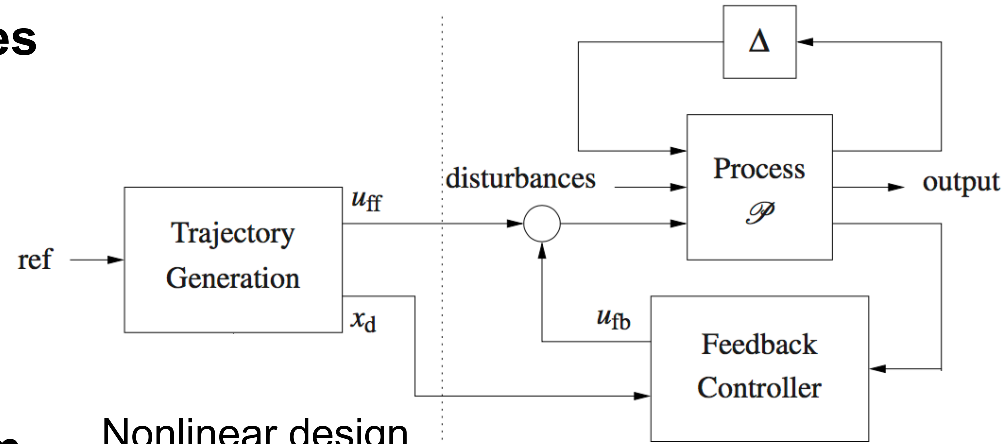
Solve as constrained optimization problem

- Various tricks to get very fast calculations
- Need to update solutions at the rate at which the reference (task description) is modified

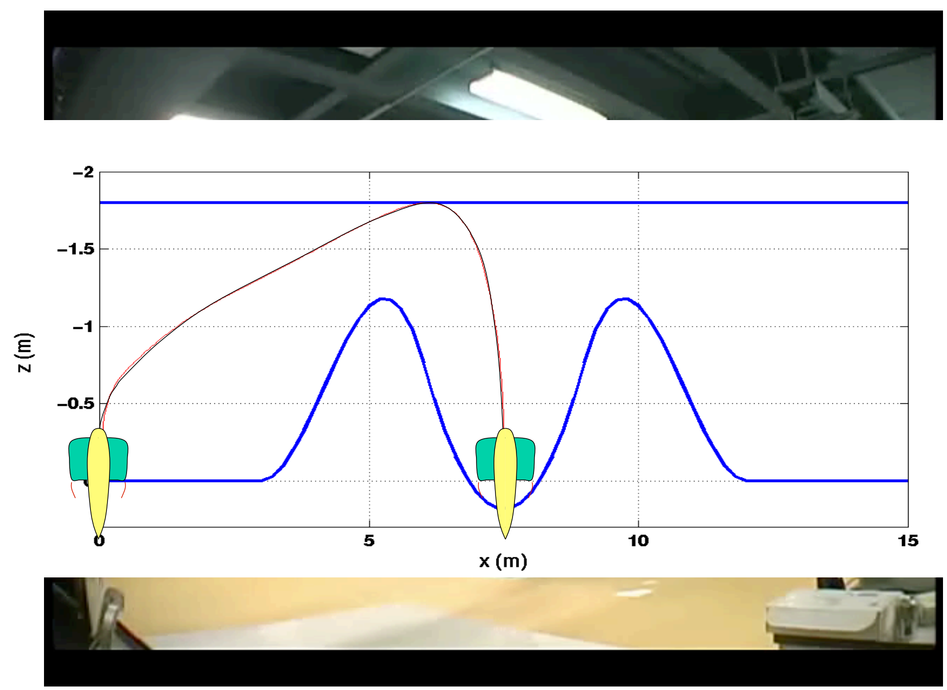
Use feedback to track trajectory

- Trajectory generation provides feasible trajectory plus nominal input
- Feedback used to correct for disturbances and model uncertainties
- Example of "two degree of freedom" design

Q: when can we find a feasible trajectory?

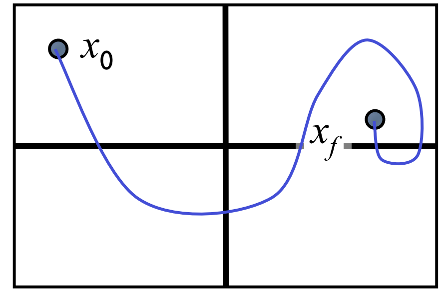


Nonlinear design
- global nonlinearities
- input saturation
- state space constraints



Reachability of Input/Output Systems

$$\dot{x} = f(x,u) \quad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = h(x,u) \quad u \in \mathbb{R},\ y \in \mathbb{R}$$



Defn An input/output system is *reachable* if for any $x_o, x_f \in \mathbb{R}^n$ and any time $T > 0$ there exists an input $u_{[0,T]} \in \mathbb{R}$ such that the solution of the dynamics starting from $x(0) = x_0$ and applying input $u(t)$ gives $x(T) = x_f$.

Remarks

- In the definition, x_0 and x_f do not have to be equilibrium points $\Rightarrow$ we don't necessarily stay at x_f after time T.
- Reachability is defined in terms of states $\Rightarrow$ doesn't depend on output
- For *linear systems,* can characterize reachability by looking at the general solution:

$$\dot{x} = Ax + Bu \qquad\qquad x(T) = e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)\,d\tau$$
$$y = Cx + Du$$

- If integral is "surjective" (as a linear operator), then we can find an input to achieve any desired final state.

Tests for Reachability

$$\dot{x} = Ax + Bu \qquad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = Cx + Du \qquad u \in \mathbb{R},\ y \in \mathbb{R}$$

$$x(T) = e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)d\tau$$

Thm A linear system is reachable if and only if the $n \times n$ *reachability matrix*

$$\begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

is full rank.

Note: the term "controllability" is also commonly used to describe this concept

Remarks

- *Very* simple test to apply. In python-control, use `ct.ctrb(A, B)` and check rank w/ `det()`
- If this test is satisfied, we say "the pair (A, B) is reachable"
- Some insight into the proof can be seen by expanding the matrix exponential

$$\begin{aligned} e^{A(T-\tau)}B &= \left(I + A(T-\tau) + \frac{1}{2}A^2(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}(T-\tau)^{n-1} + \cdots\right)B \\ &= B + AB(T-\tau) + \frac{1}{2}A^2B(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}B(T-\tau)^{n-1} + \cdots \end{aligned}$$

Trajectory Generation

Given that a (linear) system is reachable, how do we *compute* the inputs??

- Method #1: create a stabilizing control law to an equilibrium point: $u = u_e + \alpha(x-x_e)$

$$\lim_{t\to\infty} x(t) = x_e \text{ for all } x(0) \in \mathbb{R}^n \qquad \Longrightarrow \qquad x(0) = x_0 \;\to\; x(\infty) = x_e$$

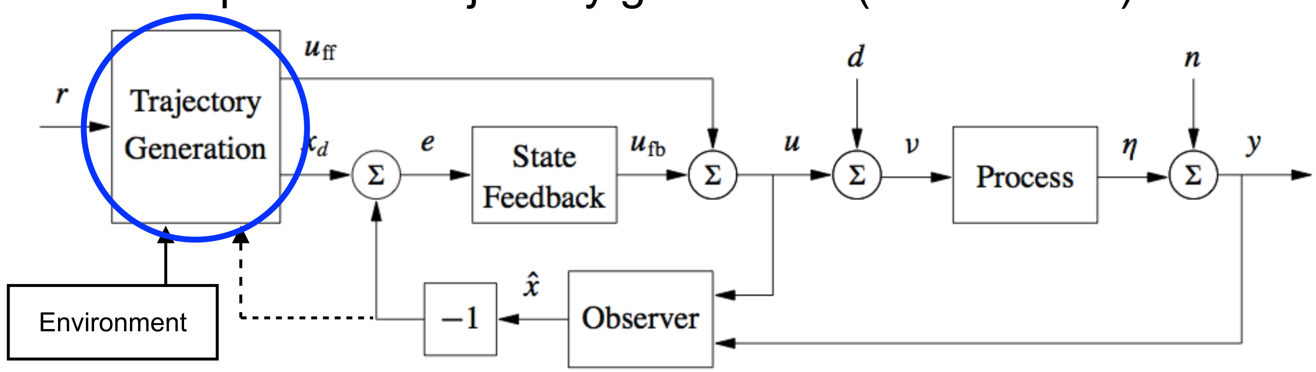
- Method #2: formulate as an "optimal control problem" and solve numerically

$$\boxed{\min_{u(\cdot)} \int_0^T L(x,u)\,dt \qquad \text{subject to} \qquad \dot{x} = f(x,u), \qquad x(0) = x_0,\; x(T) = x_f}$$

- These methods *only* work if the system is reachable and almost always require that the linearization at a nearby equilibrium point be reachable (which we can check)

Given feasible input/state trajectory, use feedback to manage uncertainty

- General picture = trajectory generation (feedforward) + feedback compensation



Types of uncertainty:

- Process disturbances
- Sensor noise
- Unmodeled dynamics

Trajectory Generation in Python-Control

Classes for representing optimal control problems and results

`OptimalControlProblem` (sys, timepts, ...[, ...])	Description of a finite horizon, optimal control problem.
`OptimalControlResult` (ocp, res[, ...])	Result from solving an optimal control problem.

Functions for setting up optimal control problems

`input_poly_constraint` (sys, A, b)	Create input constraint from polytope
`input_range_constraint` (sys, lb, ub)	Create input constraint from polytope
`output_poly_constraint` (sys, A, b)	Create output constraint from polytope
`output_range_constraint` (sys, lb, ub)	Create output constraint from range
`quadratic_cost` (sys, Q, R[, x0, u0])	Create quadratic cost function
`solve_ocp` (sys, timepts, X0, cost[, ...])	Compute the solution to an optimal control problem.
`state_poly_constraint` (sys, A, b)	Create state constraint from polytope
`state_range_constraint` (sys, lb, ub)	Create state constraint from range