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

Cloud Resources

Operators

Other Subsystems
State Space Control Design Concepts (L4-1)

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

\[
\begin{align*}
\dot{x} &= f(x, u) \quad x \in \mathbb{R}^n, \ x(0) \text{ given} \\
y &= h(x, u) \quad u \in \mathbb{R}, \ y \in \mathbb{R}
\end{align*}
\]

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
  \[
  \lim_{t \to \infty} (y(t) - y_d(t)) = 0 \text{ for all } x(0) \in \mathbb{R}^n
  \]
Real-Time Trajectory Generation

Goal: find a feasible trajectory that satisfies dynamics/constraints

\[
\min J = \int_{t_0}^{T} q(x, u) \, dt + V(x(T), u(T)) \\
\dot{x} = f(x, u) \quad lb \leq g(x, u) \leq ub
\]

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?
Defn An input/output system is reachable if for any \(x_0, x_f \in \mathbb{R}^n\) 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 ⇒ we don’t necessarily stay at \(x_f\) after time \(T\).
- Reachability is defined in terms of states ⇒ doesn’t depend on output
- For linear systems, can characterize reachability by looking at the general solution:
  \[
  \begin{align*}
  \dot{x} &= Ax + Bu \\
  y &= Cx + Du \\
  x(T) &= e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)\,d\tau
  \end{align*}
  \]
- If integral is “surjective” (as a linear operator), then we can find an input to achieve any desired final state.
Tests for Reachability

\[ \dot{x} = Ax + Bu \quad x \in \mathbb{R}^n, \quad x(0) \text{ given} \]
\[ y = Cx + Du \quad u \in \mathbb{R}, \quad y \in \mathbb{R} \]
\[ x(T) = e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)d\tau \]

**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.

**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

\[
e^{A(T-\tau)}B = \left( I + A(T-\tau) + \frac{1}{2!}A^2(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}(T-\tau)^{n-1} + \cdots \right)B
\]
\[
= B + AB(T-\tau) + \frac{1}{2}A^2B(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}B(T-\tau)^{n-1} + \cdots
\]
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: \( u = \omega + \alpha(x-x_e) \)

\[
\lim_{t \to \infty} x(t) = x_e \text{ for all } x(0) \in \mathbb{R}^n \implies x(0) = x_0 \to x(\infty) = x_e
\]

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

\[
\min_{u(\cdot)} \int_{0}^{T} L(x, u) \, dt \quad \text{subject to} \quad \dot{x} = f(x, u), \quad x(0) = x_0, \ x(T) = x_f
\]

- 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

<table>
<thead>
<tr>
<th>Class Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OptimalControlProblem(sys, timepts, ...[, ...])</code></td>
<td>Description of a finite horizon, optimal control problem.</td>
</tr>
<tr>
<td><code>OptimalControlResult(ocp, res[, ...])</code></td>
<td>Result from solving an optimal control problem.</td>
</tr>
</tbody>
</table>

## Functions for setting up optimal control problems

<table>
<thead>
<tr>
<th>Function Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input_poly_constraint(sys, A, b)</code></td>
<td>Create input constraint from polytope</td>
</tr>
<tr>
<td><code>input_range_constraint(sys, lb, ub)</code></td>
<td>Create input constraint from polytope</td>
</tr>
<tr>
<td><code>output_poly_constraint(sys, A, b)</code></td>
<td>Create output constraint from polytope</td>
</tr>
<tr>
<td><code>output_range_constraint(sys, lb, ub)</code></td>
<td>Create output constraint from range</td>
</tr>
<tr>
<td><code>quadratic_cost(sys, Q, R[, x0, u0])</code></td>
<td>Create quadratic cost function</td>
</tr>
<tr>
<td><code>solve_ocp(sys, timepts, X0, cost[, ...])</code></td>
<td>Compute the solution to an optimal control problem.</td>
</tr>
<tr>
<td><code>state_poly_constraint(sys, A, b)</code></td>
<td>Create state constraint from polytope</td>
</tr>
<tr>
<td><code>state_range_constraint(sys, lb, ub)</code></td>
<td>Create state constraint from range</td>
</tr>
</tbody>
</table>