16-665 Robot Mobility (Fall 2022)

Trajectory Generation and Tracking

Sebastian Scherer and Micah Corah

November 1st , 2022

(Based on Nathan Michael's slides 2020, 2019)

Carnegie Mellon
THE ROBOTICS INSTITUTE

Lectures

- Oct 11: Introduction to Aerial Robotics: Challenges and Current State of the Art
- Oct 13: Multi-rotor and Fixed-Wing Dynamic Models
- Oct 25: Linear Control
- Oct 27: Model Predictive and Adaptive Control
- Nov 1: Trajectory Generation and Tracking & Project FAQ
- Nov 3: Trajectory Generation and Tracking & Project FAQ

Project due <date>

Review: Linear Quadratic Regulator

LQR Problem Statement:

- Define the cost function:

$$J(x(t), u(t)) = \int_0^{\infty} (x(t)^{\mathrm{T}} Q x(t) + u(t)^{\mathrm{T}} R u(t)) dt$$

where Q and R are:

- *Symmetric*, i.e., $Q = Q^{\mathrm{T}}$ or $q_{ij} = q_{ji}$.
- *Positive definite*, i.e., $x^{\mathrm{T}} Q x > 0$ for any $x \in \mathbb{R}^n$ and $u^{\mathrm{T}} R u > 0$ for any $u \in \mathbb{R}^m$ (or all of the eigenvalues of Q and R are positive).

Review: Model Predictive Control (MPC)

- The general form of Model Predictive Control (MPC) is (from the prior slide):

$$\begin{aligned} \min_{u[k]} \quad & \sum_{k=1}^{N} \left(e[k]^{\mathrm{T}} Q e[k] + u[k]^{\mathrm{T}} R u[k] \right) \\ \text{s.t.} \quad & x_{k+1} = A x_k + B u_k \\ & G_x x_k \leq g_x \\ & G_u u_k \leq g_u \end{aligned}$$

- It is an optimization problem that you solve for $u[k]$ at *each time step*.

Introduction to Model Predictive Control (MPC)

- **Optimization approach**
  - Model and simulate system *(need a good model!)*
  - Optimize control inputs w. *constraints* & *costs*
  - **Solving:** *Convex* & *QP* approximations of *nonlinear* & *non-convex* problems
- **Uses**
  - Control complex & nonlinear systems
  - Satisfy strict safety guarantees
  - High or low level control: May control full state or provide reference trajectory to inner-loop control (*short horizon planning*)
- **Cons**
  - Can be slow/expensive, specific optimization problems can be limiting

Lecture Outline

- Differential flatness
- Optimal trajectory generation
- Control architectures
- Safe navigation

Intro: Flatness, Feed-Forward, and Geometric Control

Next lecture: *Trajectory Generation*

- Geometric control
  - Dynamics and control with *“natural”* representations of state-geometry & error
  - No singularities, valid&stable for any reference + large error
- Flatness
  - Direct correspondence between trajectory (time-parametrized polynomials&position) and full state (including rotation)
  - Simple optimization problems without MPC
  - Feasibility: Sufficient smoothness of trajectory → continuous state
- Feedforward control
  - Compute nominal state & thrusts via *flatness*
  - Feedforward anticipates via derivatives / MPC anticipates explicitly

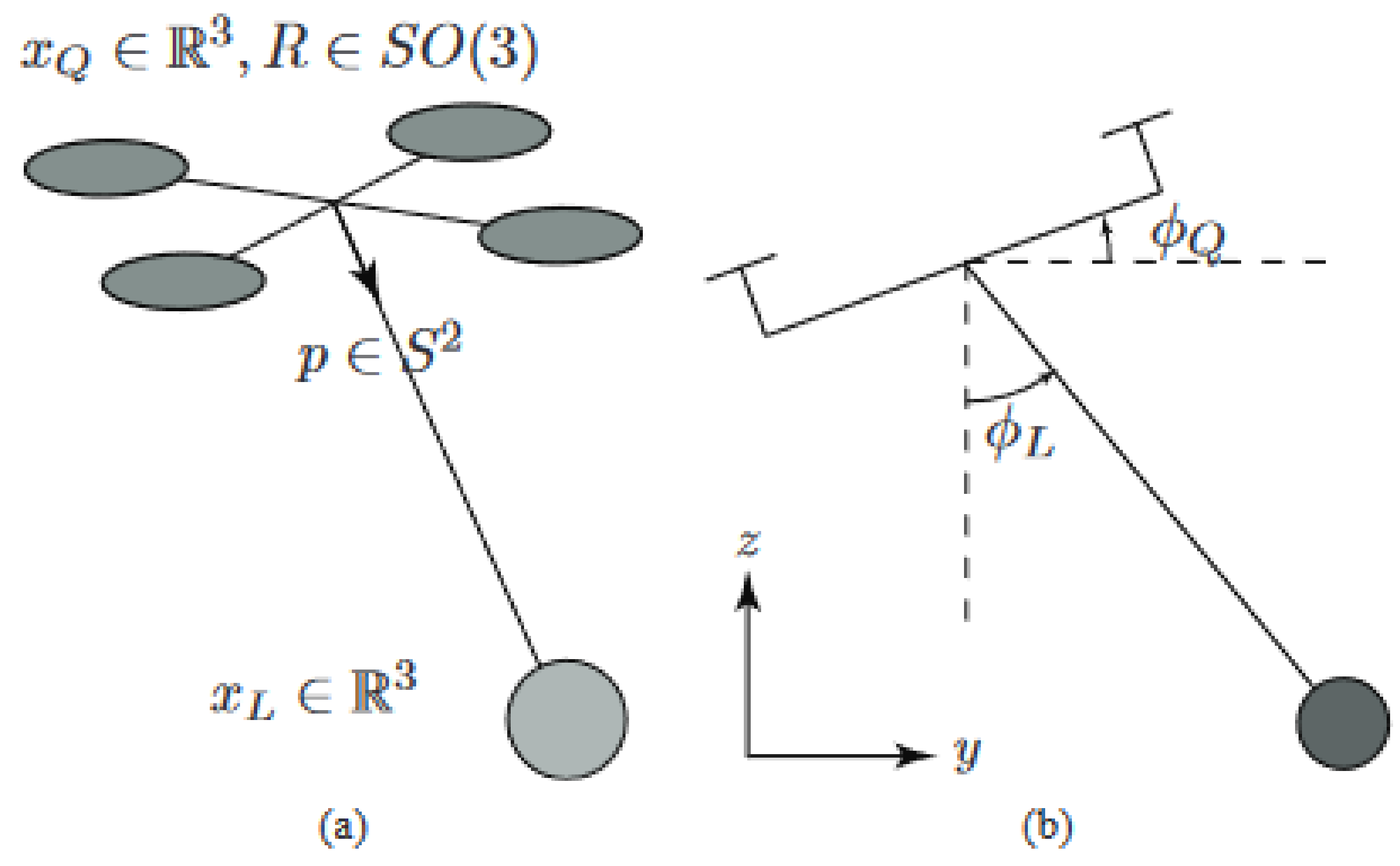
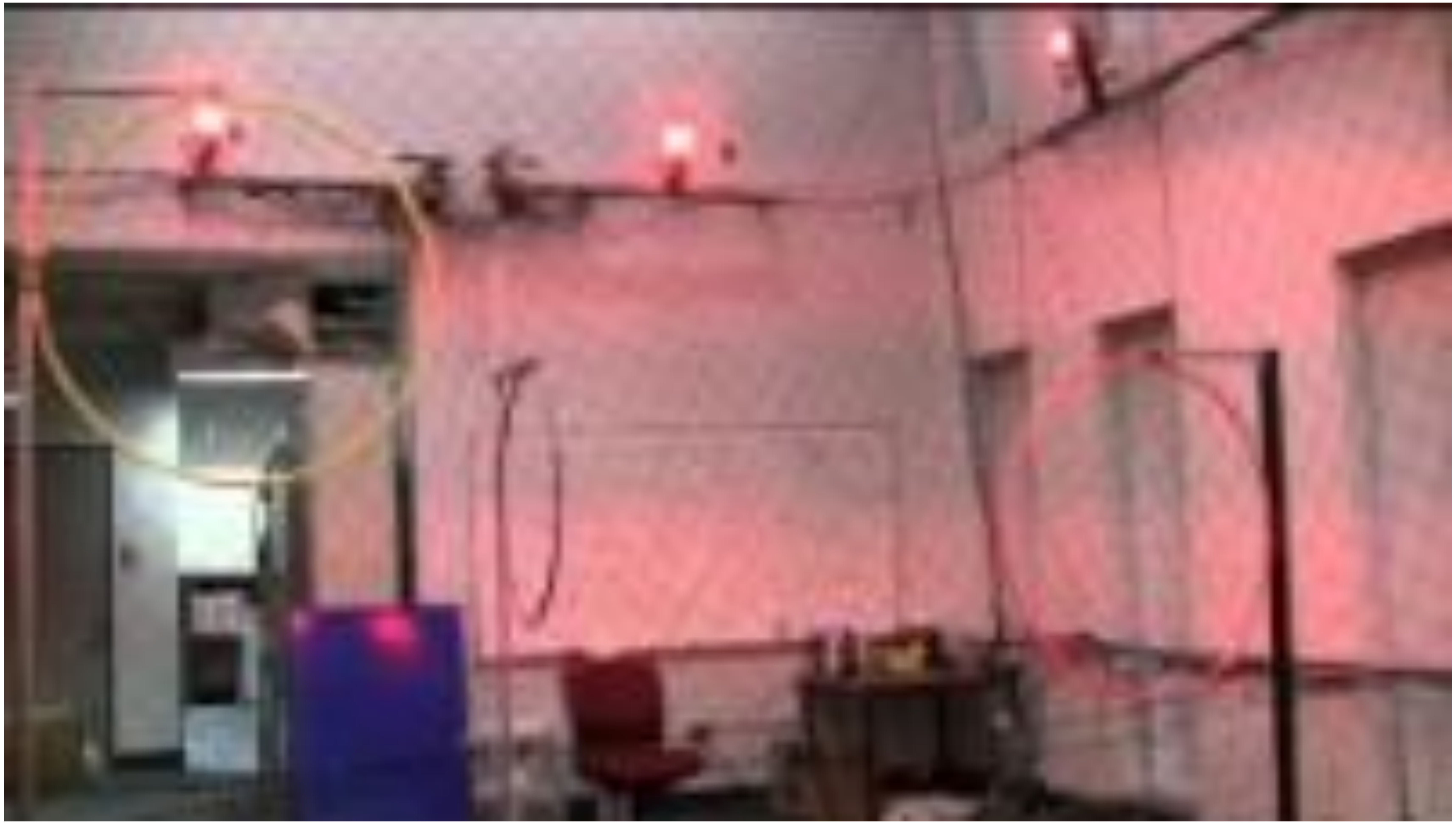


Fig. 1: (a) A 3D quadrotor with a cable suspended load. When the cable is taut, the system evolves on $SE(3) \times S^2$, and has 8 degrees of freedom with 4 degrees of underactuation. (b) A planar quadrotor with a cable suspended load evolving on $SE(2) \times S^1$

[1] Sreenath, K., Michael, N., & Kumar, V. (2013, May). Trajectory generation and control of a quadrotor with a cable-suspended load-a differentially-flat hybrid system. In 2013 IEEE international conference on robotics and automation.
[2] Lee, T., Leok, M., & McClamroch, N. H. (2010, December). Geometric tracking control of a quadrotor UAV on SE (3). In 49th IEEE conference on decision and control (CDC).

Early flatness based and minimum snap control (circa 2010)



Differential Flatness: Trajectories

Consider ways to specify a trajectory:

Pose

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \end{bmatrix}$$

State Space

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \\ \dot{x}^d(t) \\ \dot{y}^d(t) \\ \dot{z}^d(t) \\ \dot{\phi}^d(t) \\ \dot{\theta}^d(t) \\ \dot{\psi}^d(t) \end{bmatrix}$$

Flat Output
(trajectory reference)

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \psi^d(t) \end{bmatrix}$$

Differential Flatness: Trajectories

Consider ways to specify a trajectory:

Pose

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \end{bmatrix}$$

State Space

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \\ \dot{x}^d(t) \\ \dot{y}^d(t) \\ \dot{z}^d(t) \\ \dot{\phi}^d(t) \\ \dot{\theta}^d(t) \\ \dot{\psi}^d(t) \end{bmatrix}$$

Flat Output
(trajectory reference)

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \psi^d(t) \end{bmatrix}$$

Differential Flatness

- Consider the following system:

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

- The system is *underactuated* if $m < n$.
- A system is differentially flat if we can find a set of outputs (equal in number to the number of inputs) such that we can express all states and inputs in terms of those outputs and their derivatives.
  - It is “hard” to plan for underactuated system, it is easy to plan for differentially flat systems.

Differential Flatness

The system:

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

is differentially flat if we can find the outputs $z \in \mathbb{R}^m$ of the form:

$$z = \zeta\left(x, u, \dot{u}, \ldots, u^r\right)$$

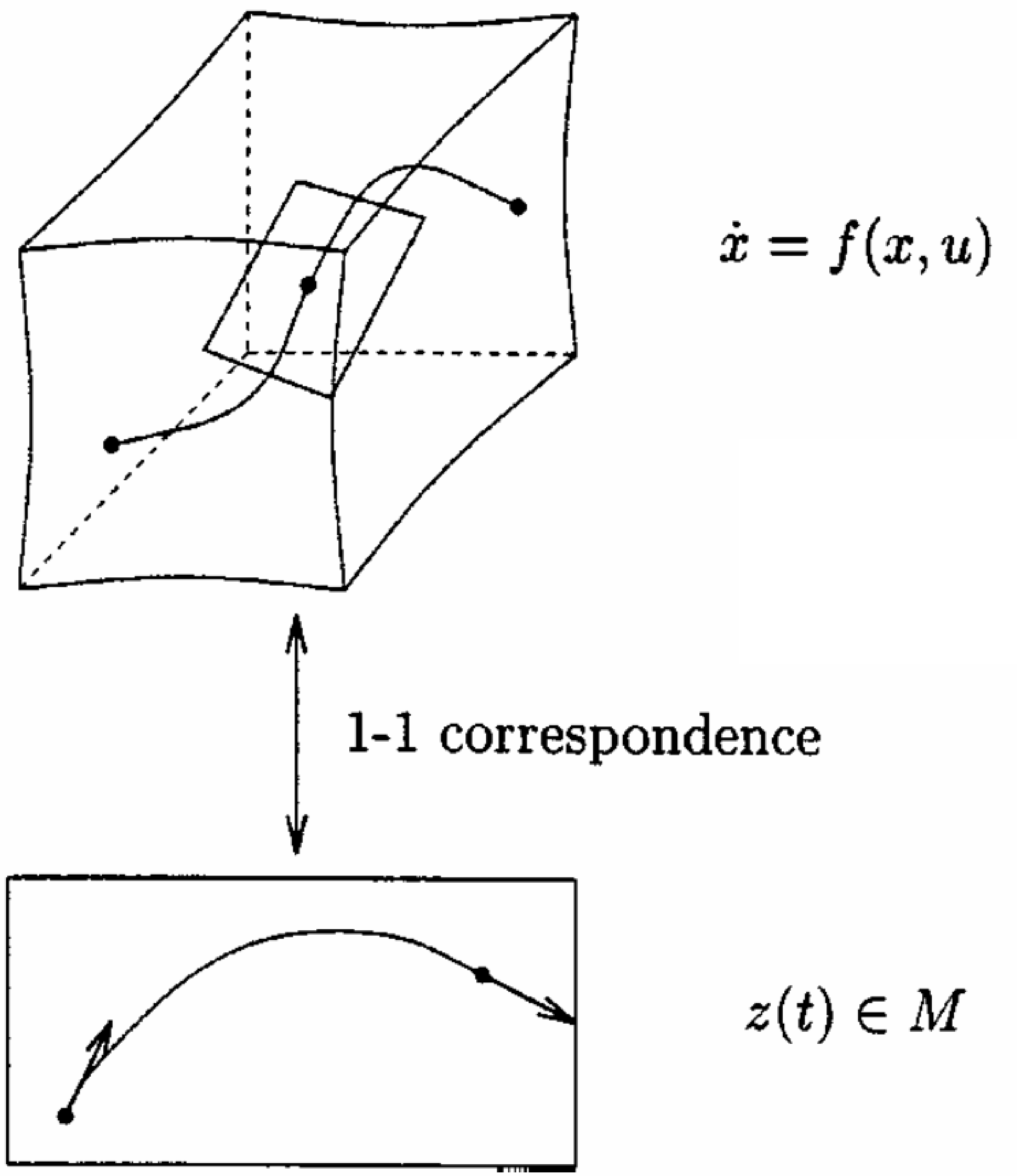
such that:

$$x = x\left(z, \dot{z}, \ldots, z^r\right) \triangleq x\left(\bar{z}\right)$$

$$u = u\left(z, \dot{z}, \ldots, z^r\right) \triangleq u\left(\bar{z}\right)$$

[Van Nieuwstadt and Murray, 1998]

Differential Flatness



Van Nieuwstadt, M., & Murray, R. M. (1996). Real time trajectory generation for differentially flat systems. *IFAC Proceedings Volumes*.

It’s Arbitrary: Specification of Flat Outputs is Not Unique

- Flatness is a property of a system *(existence of flat outputs)*
  - Outputs are not necessarily unique

Differential Flatness

- For differentially flat systems, one focuses on the trajectory generation problem instead of the linearized feedback control problem.
  - Simple linear controllers can be employed versus more complex techniques.

Is the quadcopter model differentially flat?

Differential Flatness

- Recall that we can rewrite the desired roll and pitch as functions of the reference trajectory:

$$\begin{bmatrix}\phi^d \\ \theta^d\end{bmatrix} = \frac{1}{g}\begin{bmatrix}\sin\psi^d & -\cos\psi^d \\ \cos\psi^d & \sin\psi^d\end{bmatrix}\begin{bmatrix}\ddot{e}_x + \ddot{x}^d \\ \ddot{e}_y + \ddot{y}^d\end{bmatrix}$$

- The attitude controller requires knowledge of the desired angular attitude, rates, and acceleration (e.g., ϕ^d, $\dot{\phi}^d$, $\ddot{\phi}^d$).
- Therefore, we can write the states and inputs as a function of the flat outputs: x, y, z, and ψ (and their derivatives).

Differential Flatness

- Defining the trajectory given the flat outputs:

$$\gamma(t) : [t_i,\, t_f] \to \mathbb{R}^3 \times SO(2) \longrightarrow \gamma(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \psi(t) \end{bmatrix}$$

- We can fully specify the desired inputs from this trajectory so long as:

$$\gamma_{xy}(t) \in \mathcal{C}^4$$

$$\gamma_z(t) \in \mathcal{C}^2 \longleftarrow \text{Why?}$$

$$\gamma_\psi(t) \in \mathcal{C}^2$$

Comprehension: Differential Flatness

Which represents a quadrotor trajectory in the flat output space?

1: left hand

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \psi^d(t) \end{bmatrix}$$

2: right hand

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \\ \dot{x}^d(t) \\ \dot{y}^d(t) \\ \dot{z}^d(t) \\ \dot{\phi}^d(t) \\ \dot{\theta}^d(t) \\ \dot{\psi}^d(t) \end{bmatrix}$$

3: both hands

$$\gamma(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \phi^d(t) \\ \theta^d(t) \\ \psi^d(t) \end{bmatrix}$$

Optimal Trajectory Generation

- We would like to generate a trajectory that goes from an initial to a final location over the time interval $[t_i, t_f]$ that is optimal.
  - How shall we optimize this trajectory?
    - Note that the inputs are written as a function of the flat outputs up to r.
      - Optimizing wrt the r^{th} derivative corresponds to minimizing the input.
- Formulate the optimal trajectory generation problem as an optimization problem.

Minimum Snap Trajectories

Consider lateral motion and derivatives near hover:

Position	
Velocity	Integrated acceleration
Acceleration	Attitude, thrust
Jerk	Angular rate
Snap	Angular acceleration… *Rotor rates!*

Minimizing integral of squared snap (4^{th} deriv.) approximately minimizes control effort.

Optimal Trajectory Generation

- Define the optimal trajectory as parameterized by polynomial:

$$\gamma^d(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \psi^d(t) \end{bmatrix} \longleftarrow \quad x^d(t) = \alpha_0 + \sum_{i=1}^{N} \alpha_i t^i$$

- We wish to find the polynomial coefficients that minimize the r^{th} derivative (so $r = 4$ for x and y, $r = 2$ otherwise).

Optimal Trajectory Generation

- The resulting polynomial trajectory takes the form:

$$\gamma^d(t) = \begin{bmatrix} x^d(t) \\ y^d(t) \\ z^d(t) \\ \psi^d(t) \end{bmatrix} \longrightarrow x^d(t) = \alpha_0 + \sum_{i=1}^{N} \alpha_i t^i$$

- Given initial conditions for position, velocity, and acceleration, we find that the first four polynomial coefficients can be written as a function of the ICs. $\alpha_0 = x_0 \qquad \alpha_1 = \dot{x}_0 \qquad \alpha_2 = \frac{\ddot{x}_0}{2} \qquad \alpha_3 = \frac{\dddot{x}_0}{6}$

Optimal Trajectory Generation

- Substituting into the polynomial trajectory definition (for *x*(*t*)):

$$x^d(t) = x_0 + \dot{x}_0 t + \ddot{x}_0 \frac{t^2}{2} + \dddot{x}_0 \frac{t^3}{6} + \sum_{i=4}^{N} \alpha_i t^i$$

Initial conditions

Computed via QP.

- Note use of normalized time in Mathematic example (force time to always range from 0 to 1 by rescaling ($t = t_f \tau, \tau \in [0,\ 1]$)).

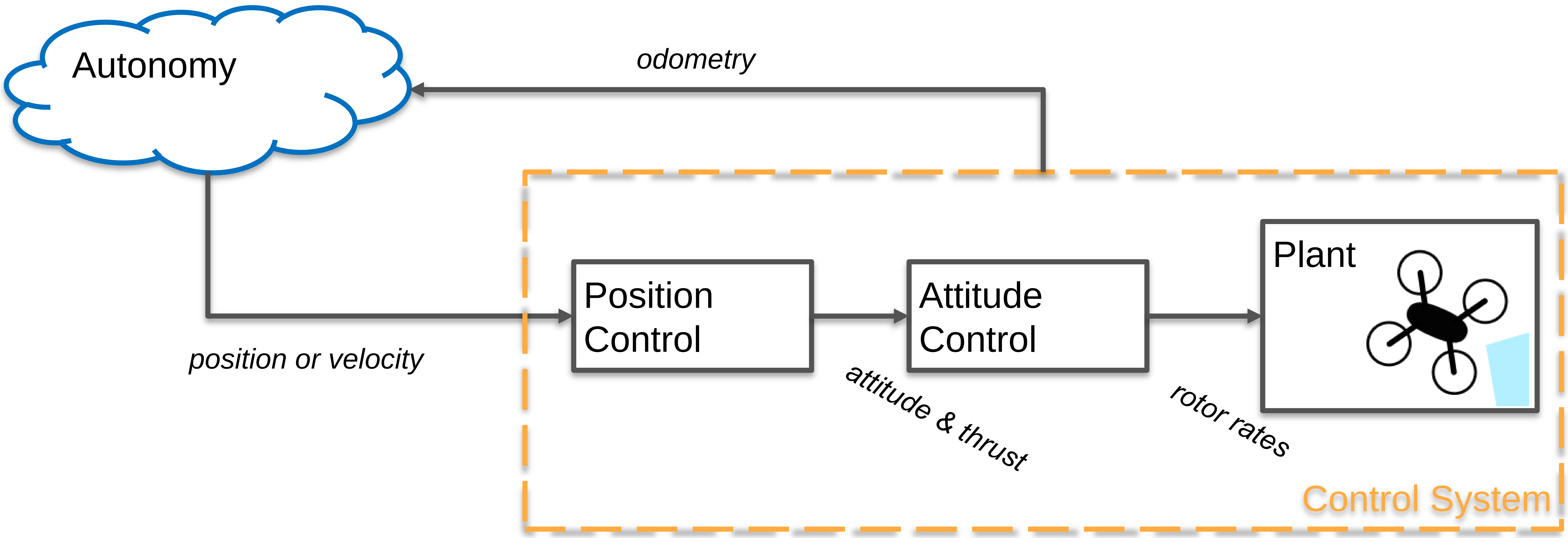
Technical Considerations for Trajectory Generation

- Optimization problems may be poorly conditioned
  - Use orthogonal polynomials in practice *(covered in robomath)*
- Splines
  - Trajectories often consist of sequences of polynomials
  - Timing may also go into the optimization problem

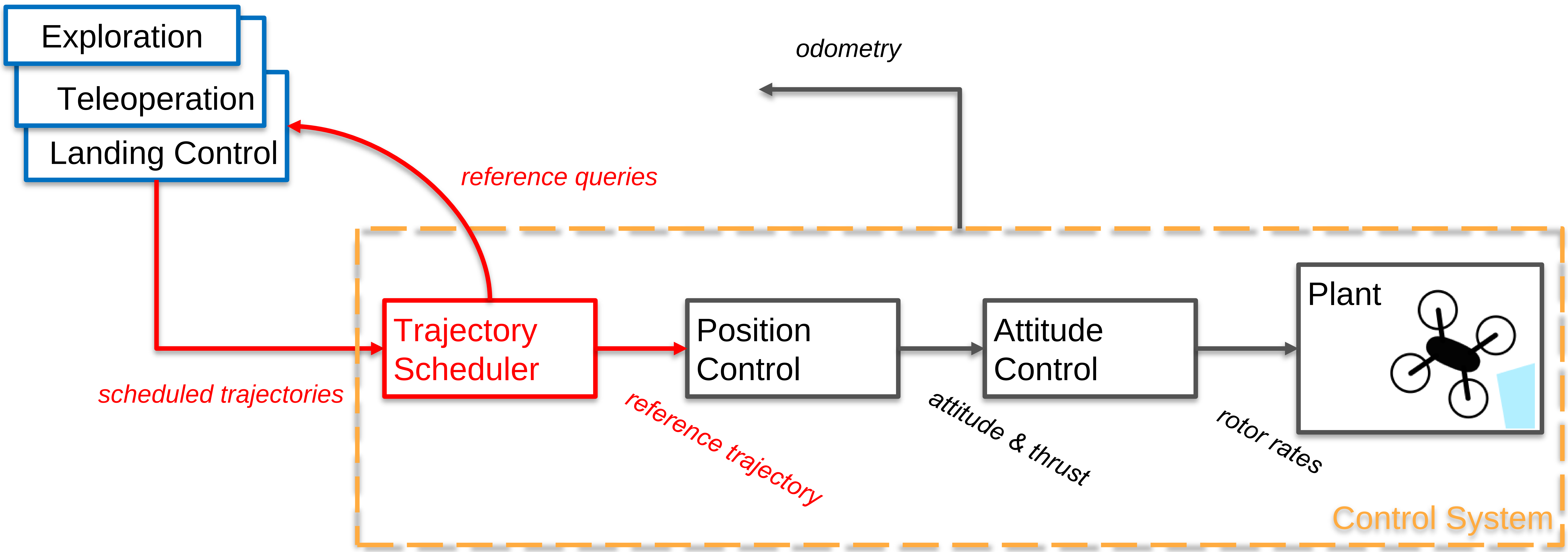
Fun With Flat Trajectories

- Patching and sequencing *(just solve a quick boundary value problem…)*
  - Computing a trajectory to "patch" (slightly) discontinuous trajectories
  - Generate smooth on/off-ramps for canned trajectories
- General analytic trajectories *(not just polynomials!)*
  - Circles, Lissajous curves *(e.g. figure eights)*

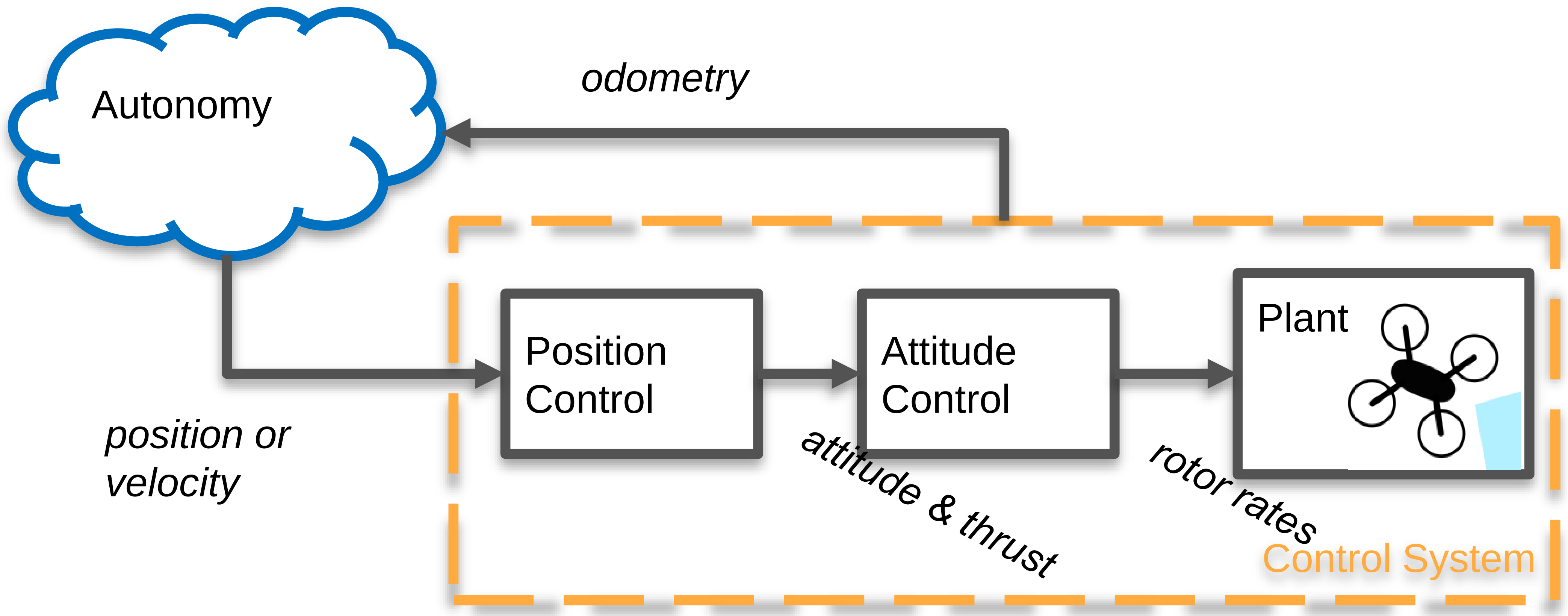
A Typical Multirotor Control Architecture

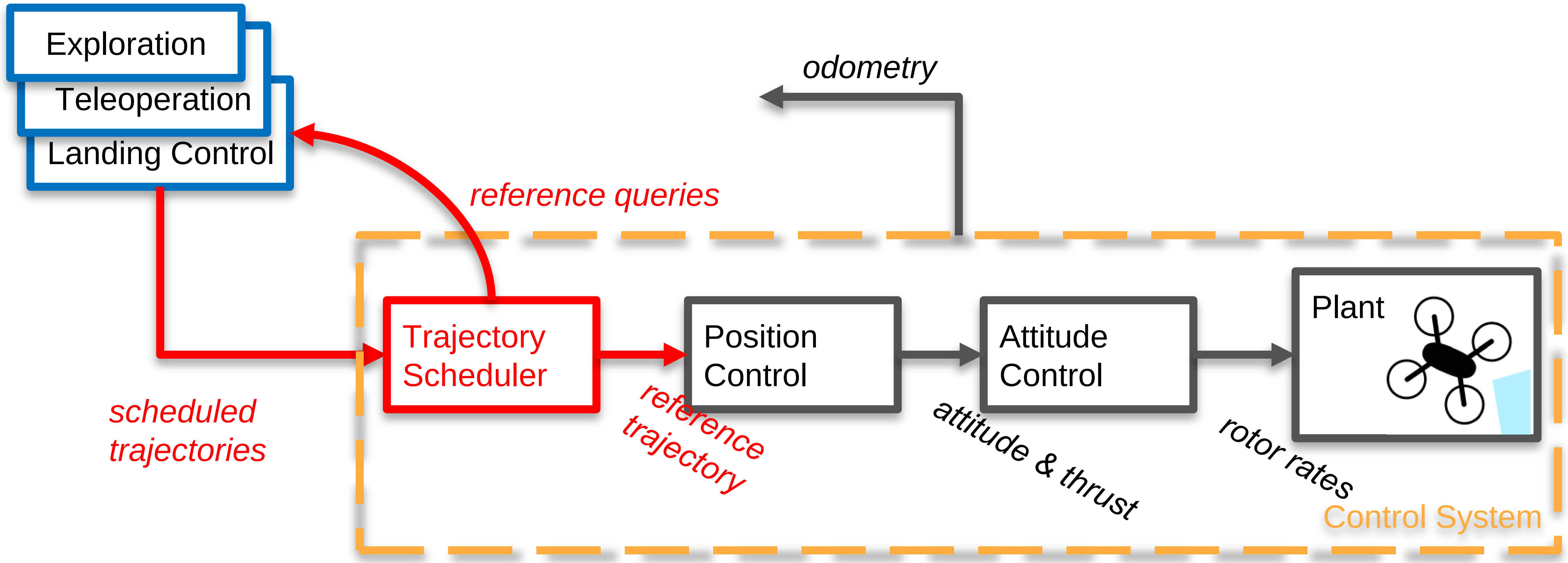


Flatness-Based System Architecture Design



Discussion: Contrasting Control Architectures





Standard vs. Flatness-based

Can these be combined? Compare to other robots? Implications for autonomy? Complexity and implementation? Other architectures? Agile motions? Safety?

Summary for Control Architectures

Standard *(position or velocity reference)*

- Easy to integrate or implement
- Less dynamic operation
- **Behavior switching:** at level of user autonomy stack
- **Safety:** Low speed, revert to zero velocity, operate in open space

Flatness-based *(trajectory scheduling)*

- Relatively complex *(more tightly integrated)*
- Supports high speeds & dynamic operation
- **Behaviors**
  - Managed transitions between autonomy systems
  - Smooth transitions by querying future references
- **Safety:** Trajectories should come to a stop *(derivatives are zero)*

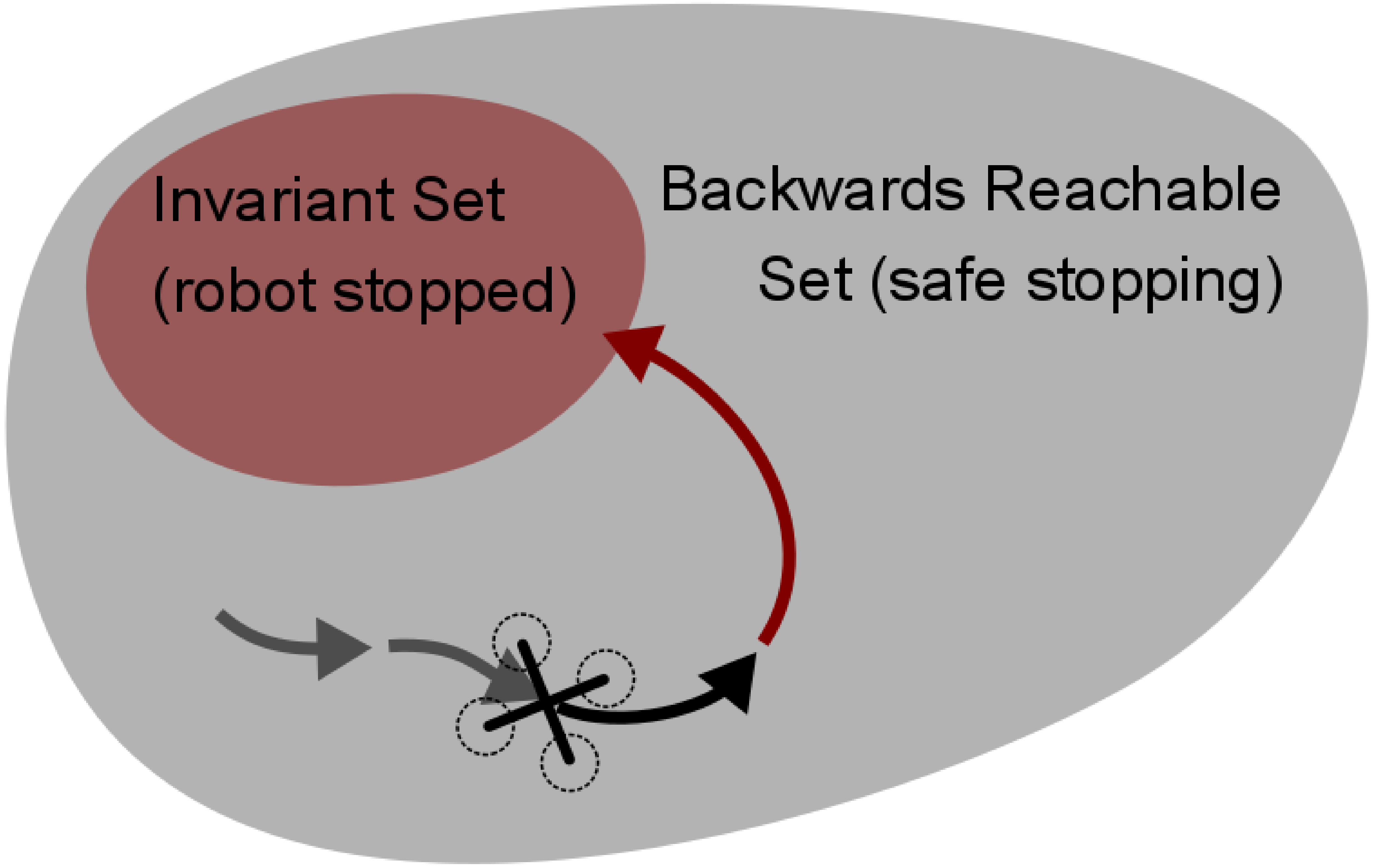
Safe Navigation with Dynamic Robots

What if:

- *Autonomy fails at high speed?*
- *Wall at end of trajectory?*
- *Behavior transitions with different requirements?*

Safe Navigation with Dynamic Robots

Maintain safety via reachability and invariance:



By repeatedly scheduling trajectories, robots navigate safely and continuously

- Same approach is relevant to MPC

Summary for Trajectory Generation

- **Control framework:** Flatness-based, feed-forward, geometric control
- **Flatness:** Alternative space. (Mostly) arbitrary curves ↔ control inputs
- **Flat outputs (quadrotors):** *x*, *y*, *z*, *yaw*
- **Trajectory optimization:** Typically polynomials optimized via QPs
- **Architectures:** Trajectory scheduling
- **Safe navigation:** Trajectories terminate in invariant states (stopping etc.)

Project FAQ