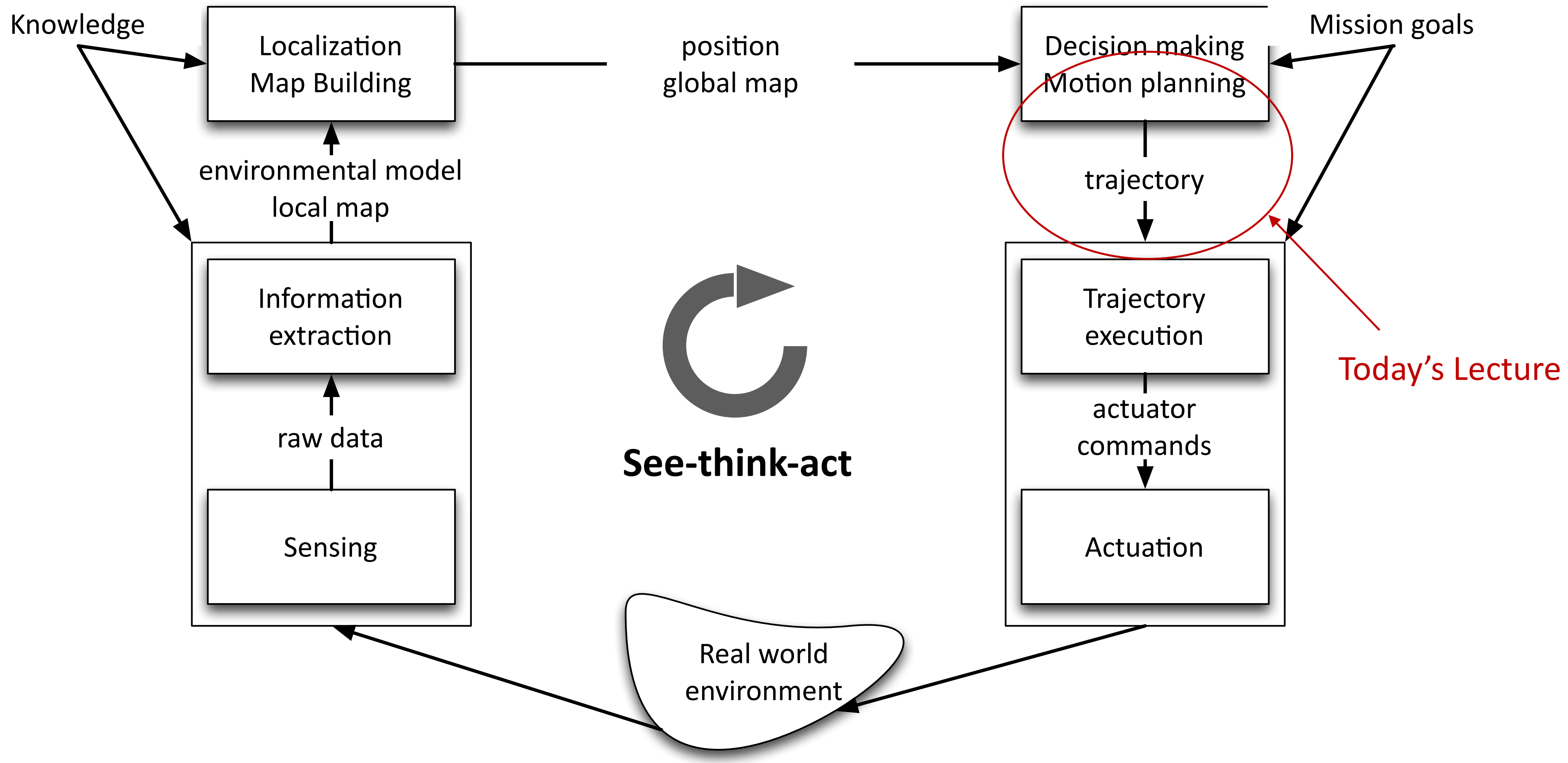
Principles of Robot Autonomy I

Open-loop motion control and differential flatness

Logistics

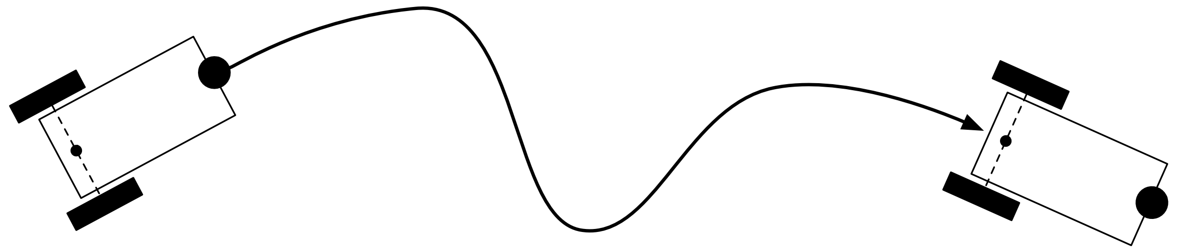
- Masks
- New TA: Aniket Bhatia
- Homework 1 due in one week: Tuesday, 10/11 (11:59PM)
- Sections start this week
  - Section 1: intro to programming tools; ROS teaser
- Waitlist is being processed
  - permission codes will be sent soon according to criteria we outlined
  - After that, waitlist process will run automatically
- High Resolution Course feedback
- iPad notes

The see-think-act cycle



Motion control

- Given a nonholonomic system, how to control its motion from an initial configuration to a final, desired configuration



- Aim
  - Learn about main techniques in optimal control and trajectory optimization
  - Learn about differential flatness and its use for trajectory optimization

- Readings
  - B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010. Chapter 11.

Kinematic / dynamic models

- In lecture 1 we saw how to derive models that describe the equations of motion of a robot in the form of differential equations (DE)

$$\dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t)$$

- DEs are equations relating the derivatives of an unknown function to the unknown function itself and known quantities

- DEs can be *integrated* numerically, for example, via the Euler method

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t_i \mathbf{a}(\mathbf{x}_i, \mathbf{u}_i, t_i), \quad i = 0, \ldots, N-1$$

where $\Delta t_i = t_{i+1} - t_i$, $\mathbf{u}_i = \mathbf{u}(t_i)$, and $\mathbf{x}_0 = \mathbf{x}(t_0)$

Optimal control problem

The problem:

$$\begin{aligned} \min_{\mathbf{u}} \quad & h(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \\ \text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) \\ & \mathbf{x}(t) \in \mathcal{X}, \quad \mathbf{u}(t) \in \mathcal{U} \end{aligned}$$

where $\boldsymbol{x}(t) \in R^n, \boldsymbol{u}(t) \in R^m$, and $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$

- We'll focus on the case $\mathcal{X} = R^n$; state constraints will be addressed in the context of motion planning
- Good reference: D. K. Kirk. Optimal Control Theory: An introduction. 2004.

Form of optimal control

- If a functional relationship of the form

$$\mathbf{u}^*(t) = \pi(\mathbf{x}(t), t)$$

can be found, then the optimal control is said to be in *closed-loop* form

- If the optimal control law is determined as a function of time for a specified initial state value

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

then the optimal control is said to be in *open-loop* form

- A good compromise: two-step design

Reference trajectory

$$\mathbf{u}^*(t) = \mathbf{u}_d(t) + \pi(\mathbf{x}(t), \underbrace{\mathbf{x}(t) - \mathbf{x}_d(t)}_{\text{Tracking error}})$$

Reference control (open-loop)

Trajectory-tracking law (closed-loop)

Open-loop control

- We want to find

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- In general, two broad classes of methods:

  1. Direct methods: transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP $\Rightarrow$ “First discretize, then optimize”
  2. Indirect methods: attempt to find a minimum point “indirectly,” by solving the necessary conditions of optimality $\Rightarrow$ “First optimize, then discretize”

Open-loop control

- We want to find

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- In general, two broad classes of methods:

  1. Direct methods: transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP $\Rightarrow$ "First discretize, then optimize"
  2. Indirect methods: attempt to find a minimum point "indirectly," by solving the necessary conditions of optimality $\Rightarrow$ "First optimize, then discretize"

- For an in-depth study of direct and indirect methods, see AA203 "Optimal and Learning-based Control" (Spring 2023)

Direct methods: nonlinear programming transcription

$$\min \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)\ dt$$

(**OCP**)

$$\dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t),\ t \in [t_0, t_f]$$

$$\mathbf{x}(0) = \mathbf{x}_0,\ \ \mathbf{x}(t_f) \in M_f$$

$$\mathbf{u}(t) \in U \subseteq \mathbb{R}^m,\ \ t \in [t_0, t_f]$$

Direct methods: nonlinear programming transcription

(OCP)

$$\min \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)\ dt$$

$$\dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t),\ \ t \in [t_0, t_f]$$

$$\mathbf{x}(0) = \mathbf{x}_0,\ \ \mathbf{x}(t_f) \in M_f$$

$$\mathbf{u}(t) \in U \subseteq \mathbb{R}^m,\ \ t \in [t_0, t_f]$$

Forward Euler time discretization

1. Select a discretization $0 = t_0 < t_1 < \cdots < t_N = t_f$ for the interval $[t_0, t_f]$ and, for every $i = 0, \dots, N-1$, define $\mathbf{x}_i \sim \mathbf{x}(t)$, $\mathbf{u}_i \sim \mathbf{u}(t)$, $t \in (t_i, t_{i+1}]$ and $\mathbf{x}_0 \sim \mathbf{x}(0)$

2. By denoting $\Delta t_i = t_{i+1} - t_i$, (**OCP**) is transcribed into the following nonlinear, constrained optimization problem

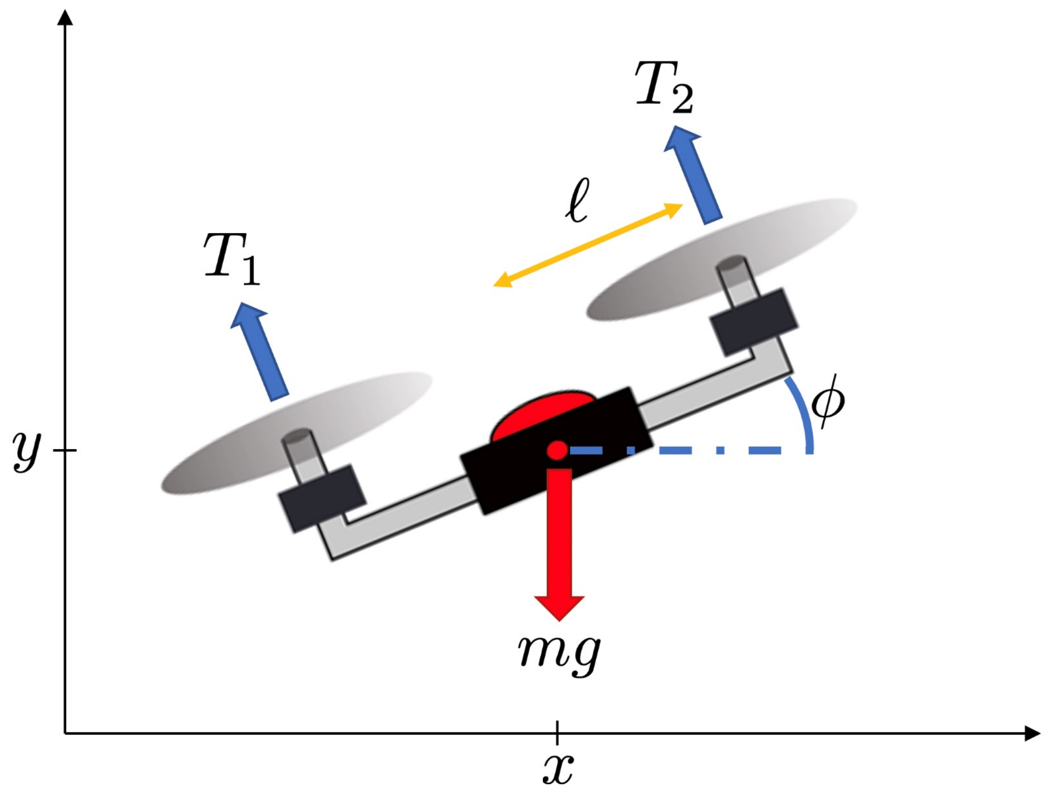
(NLOP)

$$\min_{(\mathbf{x}_i, \mathbf{u}_i)} \sum_{i=0}^{N-1} \Delta t_i g(\mathbf{x}_i, \mathbf{u}_i, t_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t_i \mathbf{a}(\mathbf{x}_i, \mathbf{u}_i, t_i), \qquad i = 0, \dots, N-1$$

$$\mathbf{u}_i \in U, i = 0, \dots, N-1 , \qquad F(\mathbf{x}_N) = 0$$

Illustrative example: planar quadrotor



$$\min \int_0^{t_f} T_1(t)^2 + T_2(t)^2 dt$$

(energy objective)

subject to dynamics

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2)\sin\phi}{m} \\ v_y \\ \frac{(T_1+T_2)\cos\phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

Direct methods: software packages

Some software packages:

- DIDO: http://www.elissarglobal.com/academic/products/
- PROPT: http://tomopt.com/tomlab/products/propt/
- GPOPS: http://www.gpops2.com/
- CasADi: https://github.com/casadi/casadi/wiki
- ACADO: http://acado.github.io/
- Trajax: https://github.com/google/trajax

In addition to implementing efficient trajectory optimization algorithms, many of these tools provide easier-to-use modeling languages for problem specification.

Differential flatness

- Computing “good” feasible trajectories is often sufficient for trajectory generation purposes, and typically much faster than computing optimal ones
- A class of systems for which trajectory generation is particularly easy are the so-called differentially flat systems
- Reference: M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998.

Motivating example: simple car

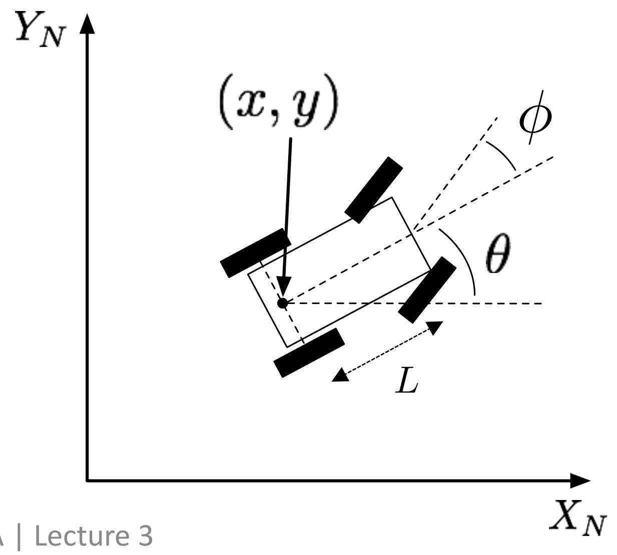
Consider the problem of finding a *feasible* solution that satisfies the dynamics:

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}), \qquad \mathbf{x}(0) = \mathbf{x}_0, \qquad \mathbf{x}(t_f) = \mathbf{x}_f$$

Example: simple car steering

$$\dot{x} = \cos\theta\, v \qquad \dot{y} = \sin\theta\, v, \qquad \dot{\theta} = \frac{v}{L}\tan\phi$$

- State: (x, y, θ)
- Inputs: (v, ϕ)



Simple car

Structure of the dynamics for simple car steering

- Suppose we are given a (smooth) trajectory for the rear wheels of the system, $x(t)$ and $y(t)$
    1. we can use this solution to solve for the angle of the car by writing

$$\frac{\dot{y}}{\dot{x}} = \frac{\sin\theta}{\cos\theta} \qquad \Rightarrow \qquad \theta = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$$

    2. we can solve for the velocity

$$\dot{x} = v\cos\theta \qquad \Rightarrow \qquad v = \dot{x}/\cos\theta \qquad (\text{or}\, v = \dot{y}/\sin\theta)$$

    3. and finally

$$\dot{\theta} = \frac{v}{L}\tan\phi \qquad \Rightarrow \qquad \phi = \tan^{-1}\left(\frac{L\,\dot{\theta}}{v}\right)$$

Structure of the dynamics for simple car steering

- Bottom line: all of the state variables and the inputs can be determined by the trajectory of the rear wheels and its derivatives!

- We say that the system is *differentially flat* with *flat output* $\mathbf{z} = (x, y)$

- This provides a dramatic simplification for the purposes of trajectory generation (more on this later)

Differential flatness

Differential flatness: A nonlinear system $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u})$ is differentially flat if there exists a function α such that

$$\mathbf{z} = \alpha(\mathbf{x}, \mathbf{u}, \ldots, \mathbf{u}^{(p)})$$

and we can write the solutions of the nonlinear system as functions of $\boldsymbol{z}$ and a finite number of derivatives

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

In words, a system is differentially flat if we can find a set of outputs (equal in number to the number of inputs) such that all states and inputs can be determined from these outputs *without integration*

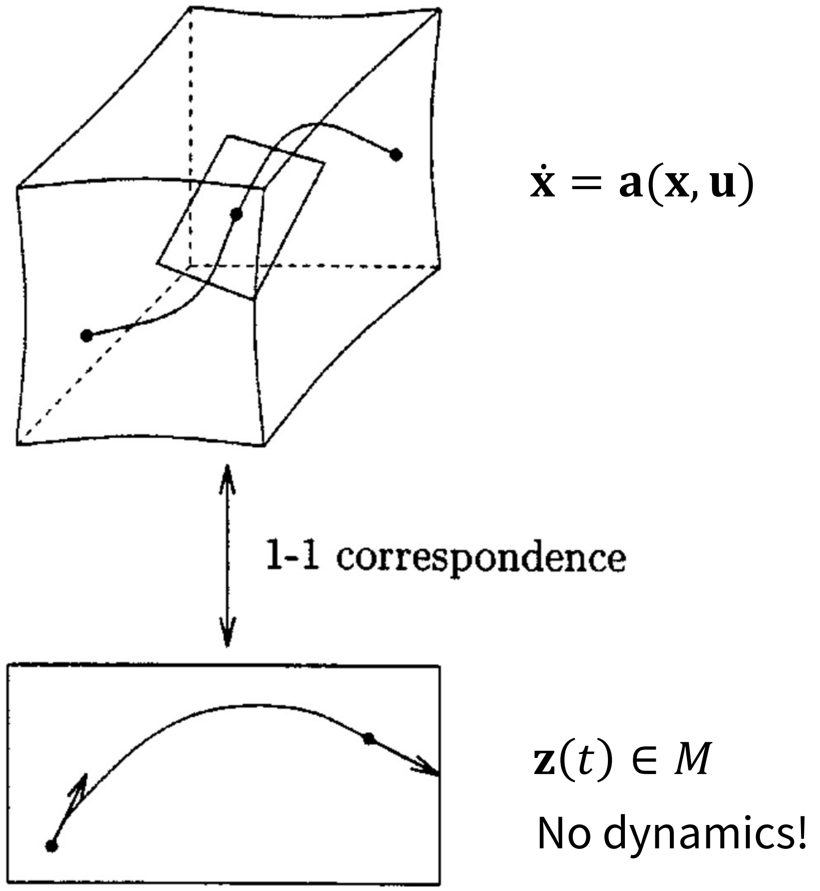
Differential flatness

- Implication for trajectory generation: to every smooth, differentiable curve $t \to \mathbf{z}(t)$, there is a corresponding trajectory

$$t \to \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{pmatrix} = \begin{pmatrix} \beta(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \\ \gamma(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \end{pmatrix}$$

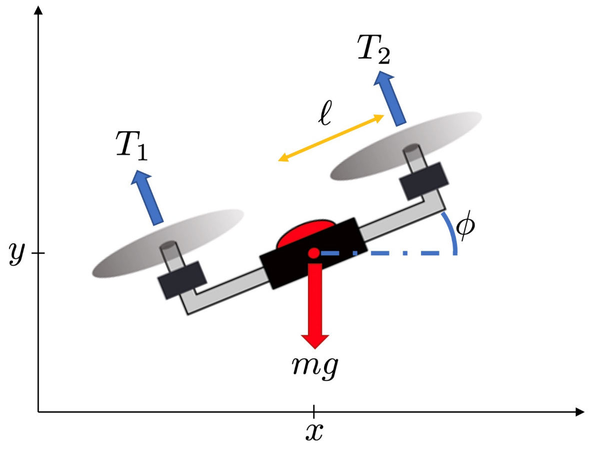
that identically satisfies the system equations

- The simple car is differentially flat with the position of the rear wheels as the flat output



From Nieuwstadt, Murray. 1998.

Another example: planar quadrotor



$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2)\sin\phi}{m} \\ v_y \\ \frac{(T_1+T_2)\cos\phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\left.\begin{aligned} x &= x \\ v_x &= \dot{x} \\ y &= y \\ v_y &= \dot{y} \\ \phi &= \tan^{-1}\left(-\frac{\ddot{x}}{\ddot{y}+g}\right) \\ \omega &= \frac{\dddot{y}\,\ddot{x} - (\ddot{y}+g)\dddot{x}}{(\ddot{y}+g)^2 + \ddot{x}^2} \end{aligned}\right\} \beta$$

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}}, \ddddot{\mathbf{z}})$$

Practical implications

This leads to a simple, yet effective strategy for trajectory generation

1. Find the initial and final conditions for the flat output:

Given	**Find**
$(t_0, \mathbf{x}(t_0), \mathbf{u}(t_0))$	$(\mathbf{z}(t_0), \dot{\mathbf{z}}(t_0), \ldots, \mathbf{z}^{(q)}(t_0))$
$(t_f, \mathbf{x}(t_f), \mathbf{u}(t_f))$	$(\mathbf{z}(t_f), \dot{\mathbf{z}}(t_f), \ldots, \mathbf{z}^{(q)}(t_f))$

2. Build a smooth curve $t \rightarrow \mathbf{z}(t)$ for $t \in [t_0, t_f]$ by interpolation, possibly satisfying further constraints
3. Deduce the corresponding trajectory $t \rightarrow (\mathbf{x}(t), \mathbf{u}(t))$

More on step 2

- We can parameterize the flat output trajectory using a set of smooth basis functions $\psi_i(t)$

$$z_j(t) = \sum_{i=1}^{N} \alpha_i^{[j]} \psi_i(t)$$

- and then solve (Problem 1 in pset)

$$\begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f) & \dot{\psi}_2(t_f) & \dots & \dot{\psi}_N(t_f) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_f) & \psi_2^{(q)}(t_f) & \dots & \psi_N^{(q)}(t_f) \end{bmatrix} \begin{bmatrix} \alpha_1^{[j]} \\ \alpha_2^{[j]} \\ \vdots \\ \alpha_N^{[j]} \end{bmatrix} = \begin{bmatrix} z_j(t_0) \\ \dot{z}_j(t_0) \\ \vdots \\ z_j^{(q)}(t_0) \\ z_j(t_f) \\ \dot{z}_j(t_f) \\ \vdots \\ z_j^{(q)}(t_f) \end{bmatrix}$$

For more details see: "Optimization-Based Control" by Richard Murray

Key points

- Nominal trajectories and inputs can be computed in a computationally-efficient way (solving a set of *algebraic equations*)

- Other constraints on the system, such as input bounds, can be transformed into the flat output space and (typically) become limits on the curvature or higher order derivative properties of the curve
  - Alternative: time scaling, i.e., break down trajectory planning in (1) finding a path (via differential flatness) and (2) defining a timing law on the path (Problem 1 in pset) -- more on this next time

- If there is a performance index for the system, this index can be transformed and becomes a functional depending on the flat outputs and their derivatives up to some order

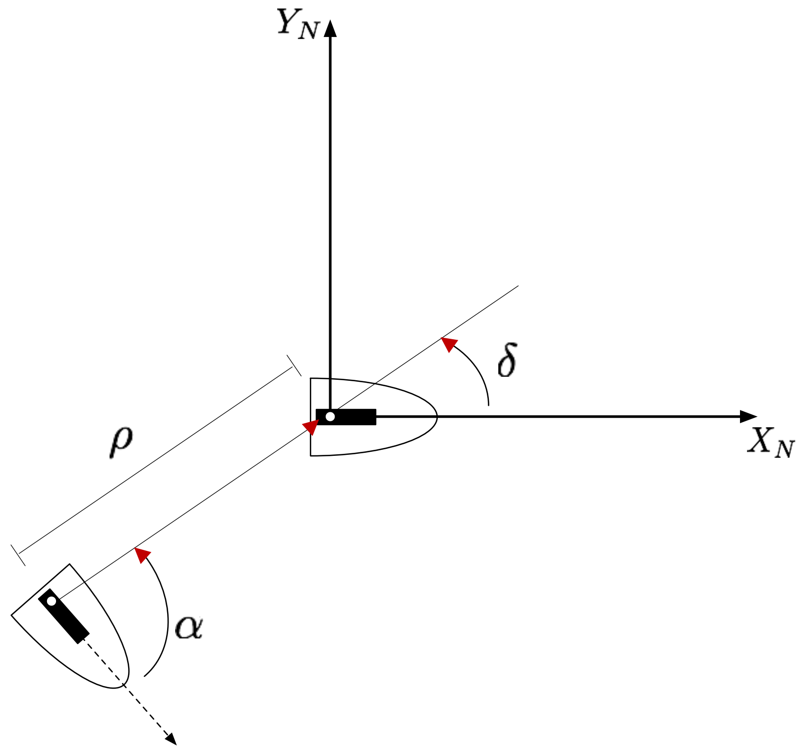
When is a system differentially flat?

- The existence of a general, computable criterion to decide if the dynamical system $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u})$ is differentially flat remains open

- Some results in this direction are, however, available

- Further readings:
  - Application to trajectory optimization:
    1. M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998
    2. R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. 1995
    3. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010
    4. D. Mellinger. Trajectory Generation and Control for Quadrotors. 2012.
  - Theory:
    1. J. Levine. Analysis and control of nonlinear systems: A flatness-based approach. 2009
    2. G. G. Rigatos, Gerasimos. Nonlinear control and filtering using differential flatness approaches: applications to electromechanical systems. 2015

Next time: trajectory tracking and closed-loop control


Y_N
δ
ρ
X_N
α