Principles of Robot Autonomy I

Trajectory Optimization

Stanford University

Autonomous Systems Lab

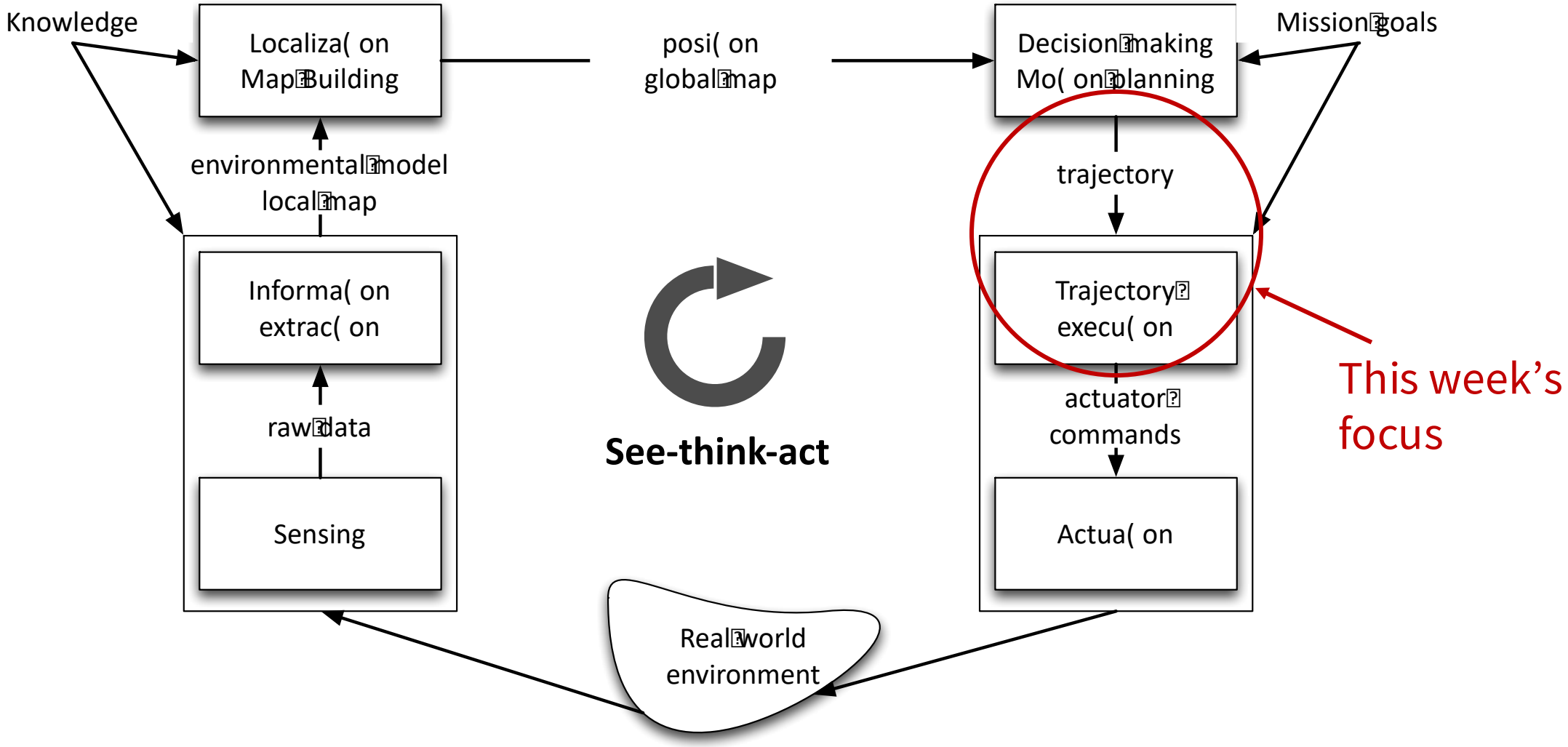
Attendance Form

Agenda

- Trajectory optimization
  - Optimization-based methods
  - Differential flatness

- Readings
  - Chapter 2, section 2.1 and sections 2.3 – 2.4 in D. Gammelli, J. Lorenzetti, K. Luo, G. Zardini, M. Pavone. *Principles of Robot Autonomy*. 2026.

The see-think-act cycle

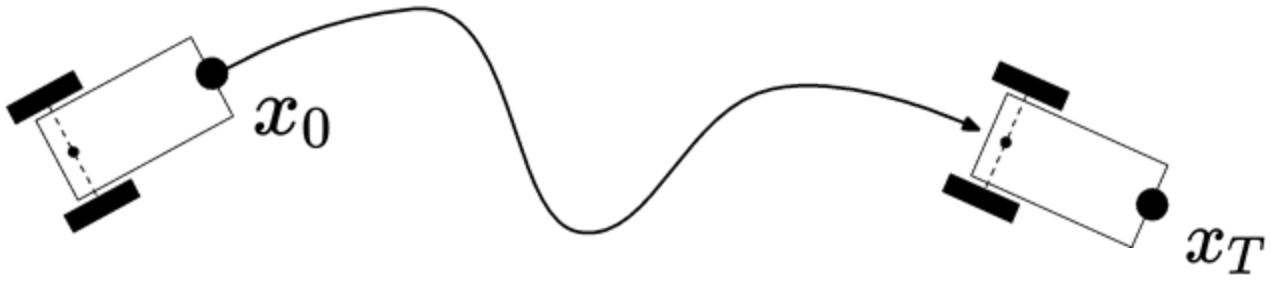


Motion generation and control

Given the state space model of a robotic system, that is

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

how can we control its motion from an initial configuration to a desired configuration?



Optimal control problem

The problem:

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

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

- We'll focus on the case $\mathcal{X} = R^n$; state constraints will be addressed in the context of motion planning

Forms 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

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

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

- A good compromise: two-step design

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

Reference trajectory

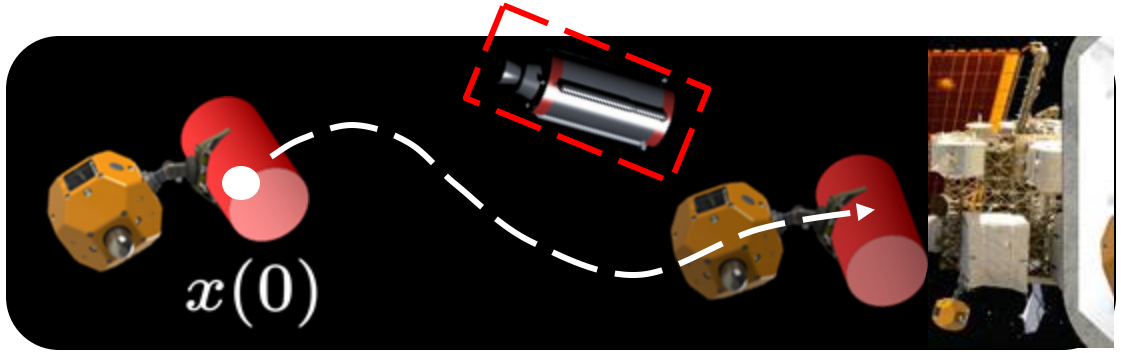
Reference control (open-loop)

Trajectory-tracking law (closed-loop)

Tracking error

Trajectory optimization

We want to find an open-loop control trajectory



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

that solves the *optimal control problem* (OCP)

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

cost (fuel consumption)

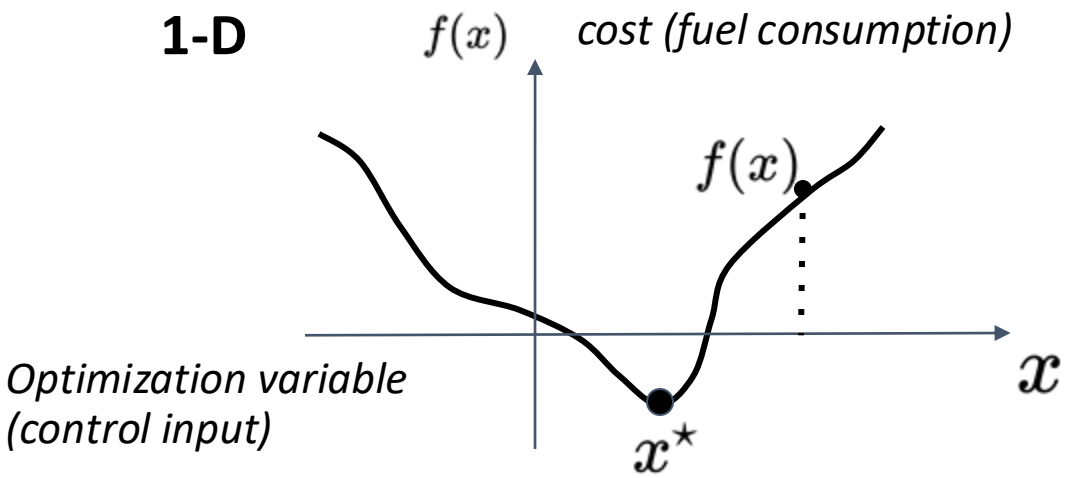
dynamics

constraints

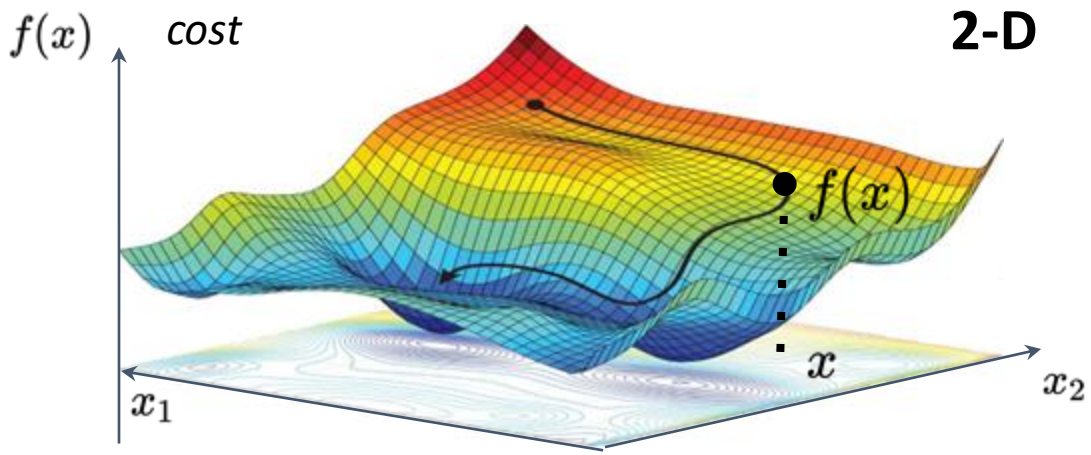
Similar to before, but focus is on open-loop control and we also remove state constraints (more general formulations are possible though!)

Non-linear optimization

1-D



2-D



Unconstrained non-linear program:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \; f(\mathbf{x})$$

- f usually assumed continuously differentiable (and often twice continuously differentiable)

Local and global minima

- A vector $\mathbf{x}^*$ is said an unconstrained *local* minimum if $\exists \epsilon > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \qquad \forall \mathbf{x} \,\big|\, \|\mathbf{x} - \mathbf{x}^*\| < \epsilon$$

- A vector $\mathbf{x}^*$ is said an unconstrained *global* minimum if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \qquad \forall \mathbf{x} \in \mathbb{R}^n$$

- $\mathbf{x}^*$ is a strict local/global minimum if the inequality is strict

Necessary conditions for optimality

Key idea: compare cost of a vector with cost of its close neighbors

- Assume $f \in C^1$, by using Taylor series expansion

$$f(\mathbf{x}^* + \Delta\mathbf{x}) - f(\mathbf{x}^*) \approx \nabla f(\mathbf{x}^*)' \Delta\mathbf{x}$$

- If $f \in C^2$

$$f(\mathbf{x}^* + \Delta\mathbf{x}) - f(\mathbf{x}^*) \approx \nabla f(\mathbf{x}^*)' \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}' \nabla^2 f(\mathbf{x}^*) \Delta\mathbf{x}$$

Necessary conditions for optimality

- We expect that if $\mathbf{x}^*$ is an unconstrained local minimum, the first order cost variation due to a small variation $\Delta\mathbf{x}$ is nonnegative, i.e.,

$$\nabla f(\mathbf{x}^*)'\Delta\mathbf{x} = \sum_{i=1}^{n} \frac{\partial f(\mathbf{x}^*)}{\partial x_i}\Delta x_i \geq 0$$

- By taking $\Delta\mathbf{x}$ to be positive and negative multiples of the unit coordinate vectors, we obtain conditions of the type

$$\frac{\partial f(\mathbf{x}^*)}{\partial x_i} \geq 0, \quad \text{and} \quad \frac{\partial f(\mathbf{x}^*)}{\partial x_i} \leq 0$$

- Equivalently we have the necessary condition

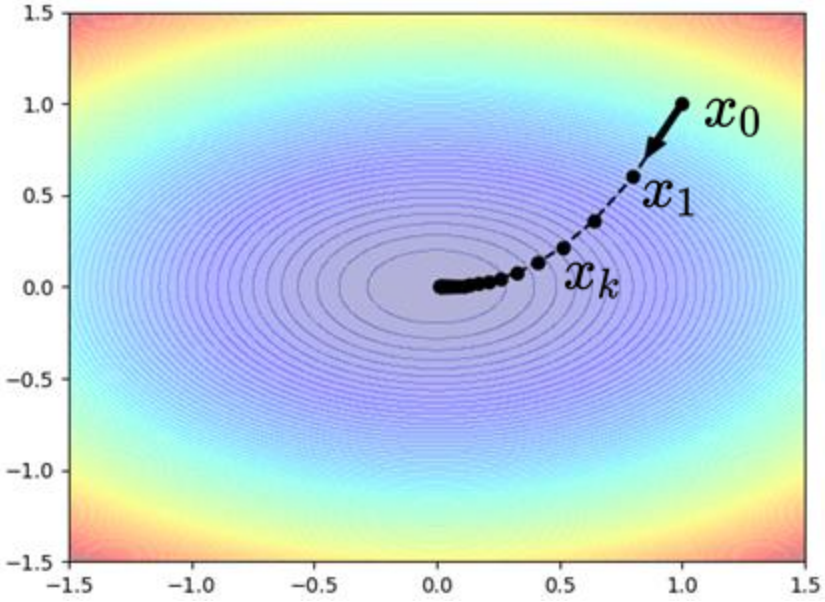
$$\boxed{\nabla f(\mathbf{x}^*) = 0} \qquad (\mathbf{x}^* \text{ is said a stationary point})$$

Computational methods

Key idea: *iterative descent*. We start at some point $\mathbf{x}^0$ (initial guess) and successively generate vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots$ such that f is decreased at each iteration, i.e.,

$$f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k), \qquad k = 0, 1, \ldots$$

The hope is to decrease f all the way to a minimum



Gradient method

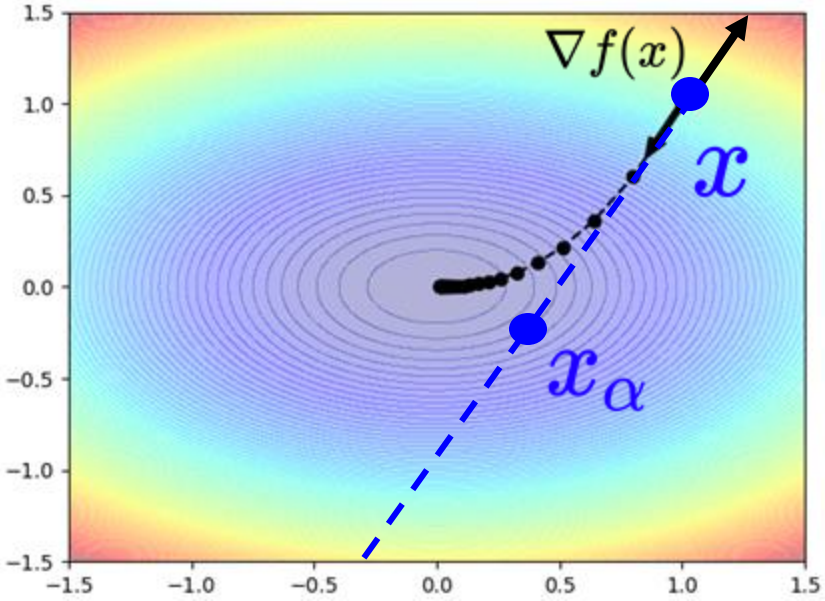
Given $\mathbf{x} \in \mathbb{R}^n$ with $\nabla f(\mathbf{x}) \neq 0$, consider the half line of vectors

$$\mathbf{x}_\alpha = \mathbf{x} - \alpha \nabla f(\mathbf{x}), \qquad \forall \alpha \geq 0$$

From first order Taylor expansion (α small)

$$f(\mathbf{x}_\alpha) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})'(\mathbf{x}_\alpha - \mathbf{x}) = f(\mathbf{x}) - \alpha \|\nabla f(\mathbf{x})\|^2$$

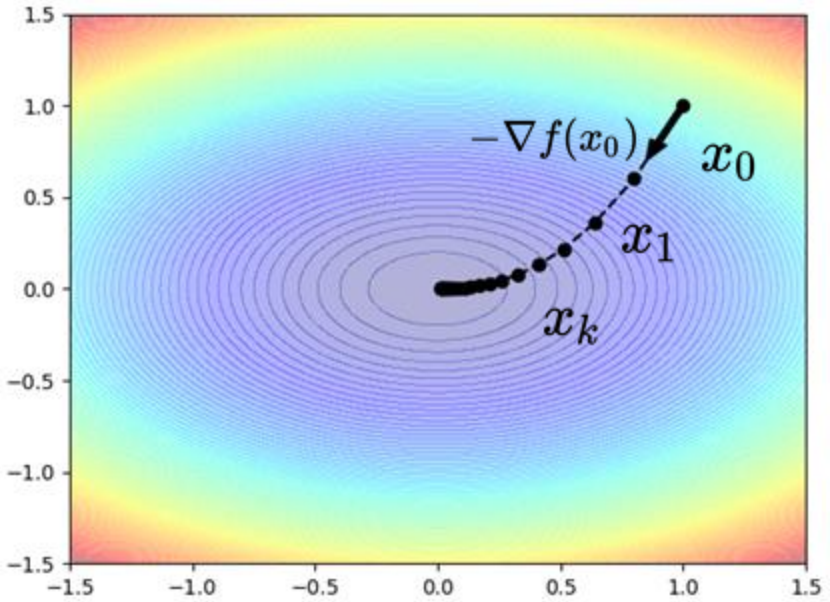
So for α small enough $f(\mathbf{x}_\alpha)$ is smaller than $f(\mathbf{x})$!



Gradient descent

- Pick an initial guess x_0 and a stepsize $\alpha > 0$
- For $k = 0, 1, \ldots$ execute the iteration:

$$x_{k+1} = x_k - \alpha \cdot \nabla f(x_k)$$




```python
import numpy as np
import matplotlib.pyplot as plt
# Define f(x) and its gradient
def f(x1, x2):
    value = 1. * x1**2 + 2. * x2**2
    return value
def f_gradient(x1, x2):
    gradient = np.array([2 * x1, 4. * x2])
    return gradient
# Gradient Descent
num_steps = 20
stepsize = 0.1
x0 = np.ones(2)
xs = np.zeros((num_steps+1, 2))
xs[0], xk = x0, x0
for i in range(num_steps):
    xk = xk - stepsize * f_gradient(xk[0], xk[1])
    xs[i+1] = xk # save result
# Plot
xs_linspace = np.linspace(-1.5, 1.5, 100)
x1s, x2s = np.meshgrid(xs_linspace, xs_linspace)
cs = plt.contourf(x1s, x2s, f(x1s, x2s),
    cmap='jet', levels=100, alpha=0.3)
plt.plot(xs[:, 0], xs[:, 1], 'ko--')
gradient_x0 = -f_gradient(x0[0], x0[1])
plt.arrow(x0[0], x0[1],
    5e-2*gradient_x0[0], 5e-2*gradient_x0[1],
    width=0.025, color='k')
plt.show()
```

Play with lecture_5_1.ipynb

Gradient descent: additional considerations

- Tuning parameter: stepsize α
  - Use a decreasing sequence α_k
  - Select it automatically at each descent step to ensure that $f(x_\alpha) < f(x)$

- Termination criteria: stop after fixed number of iterations or once

$$|f(x_{k+1}) - f(x_k)| < \epsilon \quad \text{or} \quad \|x_{k+1} - x_k\| < \epsilon$$

Back to trajectory optimization

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

- Two broad classes of methods (for more details, see AA203):

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

Direct methods: nonlinear programming transcription

(OCP)

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

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

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

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

Initial and final conditions

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 $h_i = t_{i+1} - t_i$, (**OCP**) is transcribed into the following nonlinear, constrained optimization problem

(NLOP)

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

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

$$\mathbf{u}_i \in U\ , i = 0, \dots, N-1\ , \mathbf{x}_N = \mathbf{x}_{t_f}$$

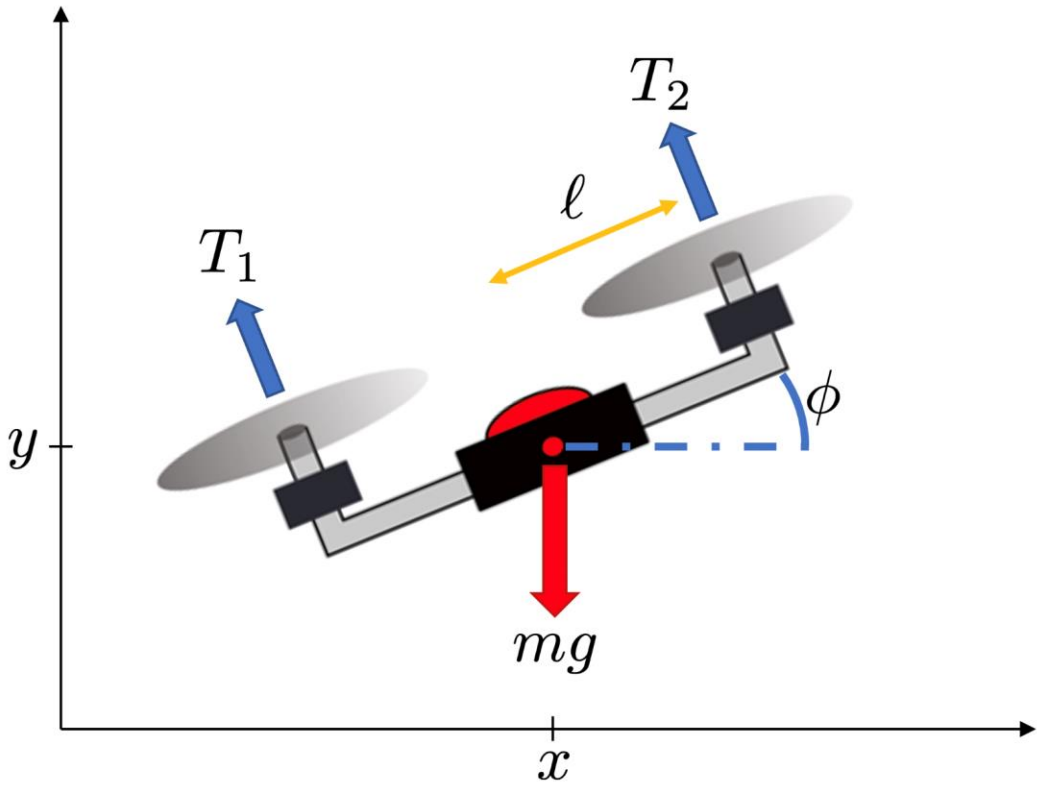
Direct methods: software packages

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

Illustrative example: planar quadrotor



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

(energy objective)

subject to dynamics

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

`Play with lecture_5_2.ipynb`

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

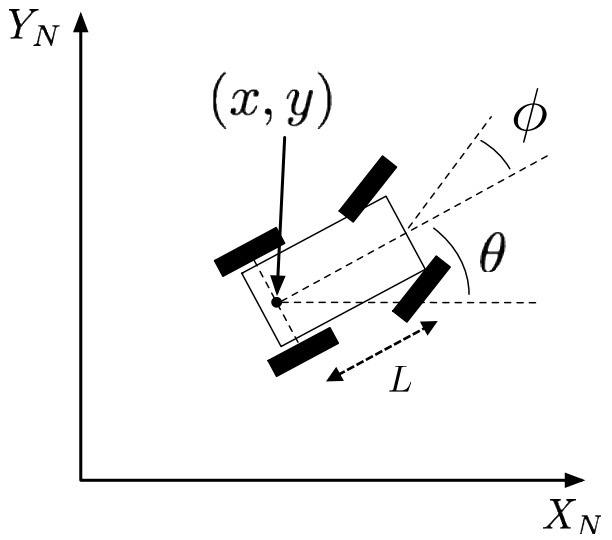
- Reference: M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998.

Motivating example: simple car

Consider the problem of finding a *feasible* trajectory that satisfies the simple car dynamics:

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

- 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

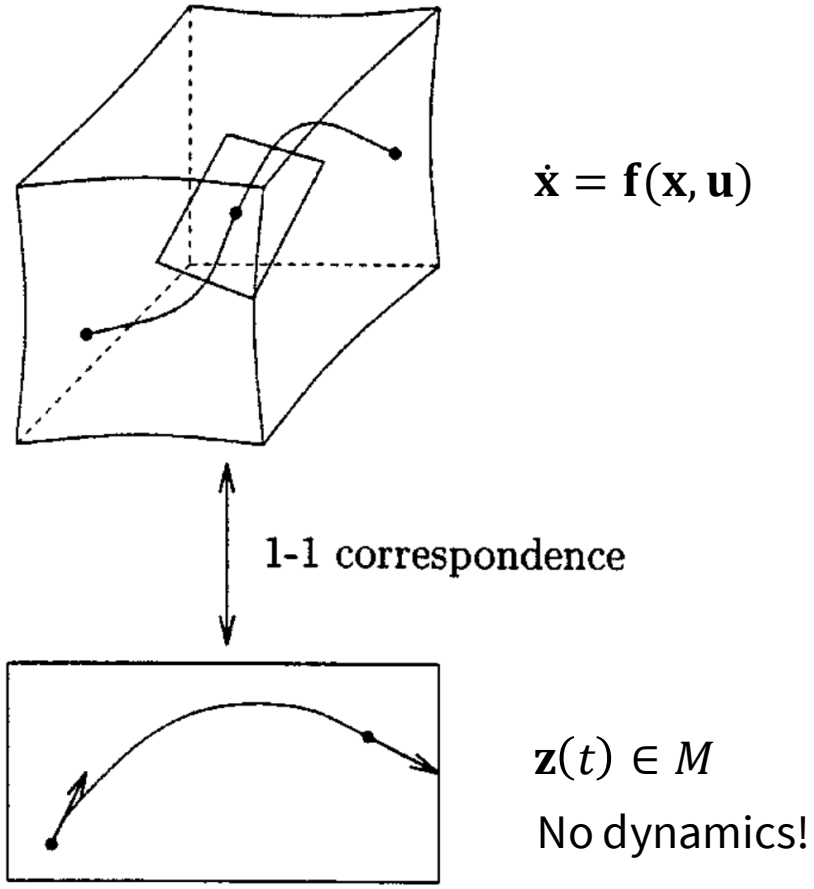
Differential flatness

- Implication for trajectory generation: to every curve $t \to \mathbf{z}(t)$ enough differentiable, there corresponds a 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.

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

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

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 2(i - iv) in pset 1)

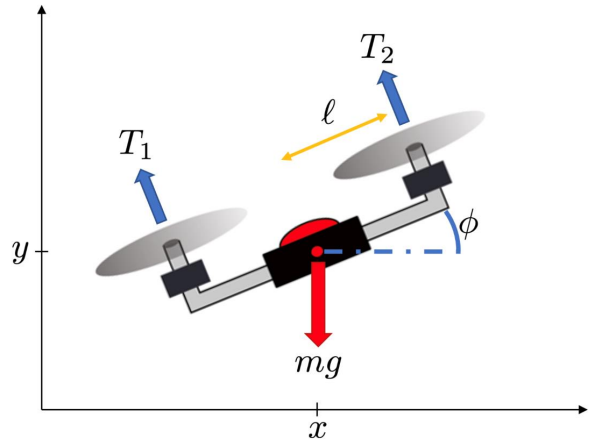
$$\begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f)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

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

The planar quadrotor is differentially flat!



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

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

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

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

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

`Play with lecture_5_2.ipynb`

Next time