Computation Methods of Trajectory Optimization in Robotics

Yuhan Zhao

NYU TANDON SCHOOL OF ENGINEERING

Group Meeting

Outline

Outline

Outline

What is Trajectory Optimization?

A continuous-time functional optimization problem subject to different constraints.

- Objective:

$$\min_{x(t),u(t)} \quad \Phi(x(t_F)) + \int_{t_0}^{t_F} L(\tau, x(\tau), u(\tau)) d\tau.$$

  - $x(t), u(t)$ are state and control trajectories, $\dim x = n$, $\dim u \in m$.
  - $\Phi(\cdot), L(\cdot)$ are terminal and stage costs.
  - t_0, t_F are the initial and final time. They can also be treated as decision variables.
  - Can be extended to multiple phases.

What is Trajectory Optimization

Constraint sets:

- System dynamics:

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

- Path constraint:

$$h(t, x(t), u(t)) \leq 0.$$

- Boundary constraint:

$$g(t_F, x(t_F)) \leq 0.$$

- Control constraint:

$$u(t) \in \mathcal{U}.$$

Comparison With Similar Terminologies

Trajectory optimization:

- Often used in robotics. Interchangeable with OC in robotics.
- Relies on optimization techniques. Computation side.

Optimal control:

- Wider scope in decision making fields.
- Wider methodology not limited to optimization, such as PMP, DP.

Path planning and Motion planning:

- Find a valid path in the configuration space that moves the object from the source to destination.
- No dynamics is involved.
- Searching methods (RRT), sampling based methods (PRM).

Kinodynamic planning:

- A motion problem that has velocity, acceleration, and force/torque constraints, and kinematic constraints such as avoiding obstacles.
- Proposed by Donald (1993), predecessor of trajectory optimization.

Outline

Classification of Computational Methods

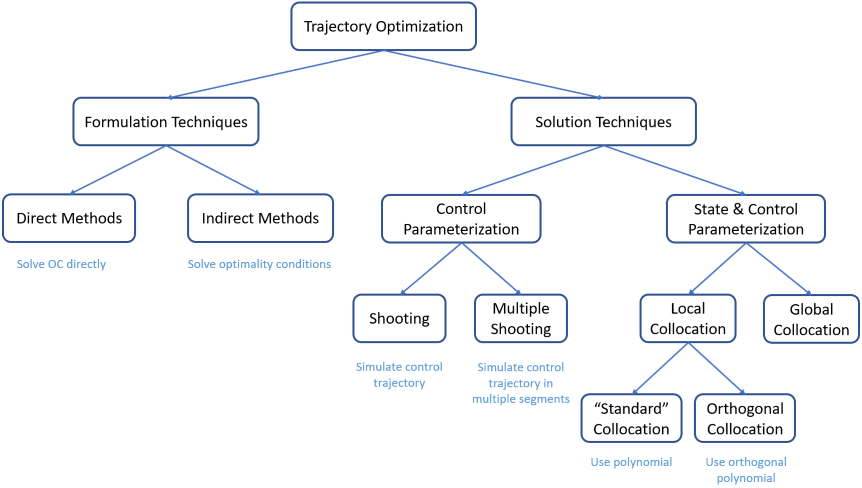


Figure: Overview of trajectory optimization.

Outline

Applications in Robotics

A very fundamental approach in robotics.

Application domain:

- Industrial robotics
- Medical robotics
- Service robotics
- Space robotics
- Autonomous driving

Robot type:

- Aerial robots
- Underwater robots
- Wheeled robots
- Legged robots
- Swarm robots

Task objective:

- Trajectory tracking.
- Path planning (minimum energy, minimum time).
- Collision avoidance and safety operation.

Outline

Outline

Two Questions

Before we proceed, we may wonder ...

- Why do we consider continuous-time settings in trajectory optimization at the beginning?
- What are the challenges to solve continuous-time problems? (leads to parameterization and discretization)

Why Continuous-time Settings?

Newton's second law:

$$F = ma, \quad \text{or} \quad F = m\frac{dv}{dt}.$$

Most robots are composed by rigid bodies (motors, non-deformable links). A robot can be described by a set of links and joints.

Figure: Common robots composed by rigid bodies.

Why Continuous-time Settings?

The Lagrangian mechanics can be used to formulate the robot model in a generalized coordinate system [1]:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) + J(q)^T f_{ext} = u.$$

Euler-Lagrange Equations:

- $q \in \mathbb{R}^n$ is the joint variable (n joints in total), $\dot{q}$ is the velocity.
- $M(q) \in \mathbb{S}^n_{++}$ is the generalized mass matrix.
- $C(q, \dot{q})$ accounts for centrifugal and Coriolis effects.
- $G(q) \in \mathbb{R}^n$ relates to gravity forces.
- $J(q)$ is the velocity Jacobian and f_{ext} is external forces (not control).
- $u \in \mathbb{R}^n$ is the control on the joint variable. e.g., motor forces.

[1] M. Spong, Robot Modeling and Control, 2008.

Why Continuous-time Setting?

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + J(q)^T f_{ext} = u.$$

A Unified Robotics Description Format (URDF) file describes a robot.

```
<joint name="arm_joint" type="revolute">
    <parent link="slider_link"/>
    <child link="arm_link"/>
    <origin xyz="0.25 0 0.15" rpy="0 0 0"/>
    <axis xyz="0 -1 0"/>
    <limit lower="0" upper="${pi/2}" velocity="100" effort="100"/>
</joint>
```

Figure: Code snippet of a URDF file.

A URDF parser can identify q and computes $M(q)$, $C(q,\dot{q})$, $J(q)$ numerically given a joint variable q.

Why Continuous-time Setting?

Two-link planner robot as an example.

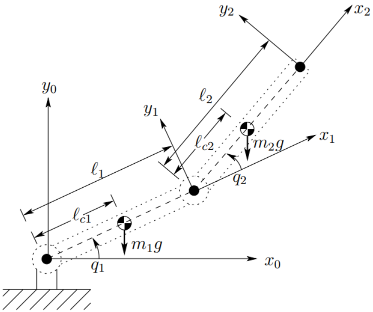


Figure: Two-link planner robot.

- Generalized coordinate q_i (rotation angle).
- Link mass m_i, moment of inertia I_i.
- Link length ℓ_i, center of mass ℓ_{ci}.

Why Continuous-time Setting?

- Link mass $m_1 = m_2 = 1$.
- Moment of inertia $I_1 = I_2 = 1$.
- Link length $\ell_1 = \ell_2 = 1$.
- Center of mass $\ell_{c1} = \ell_{c2} = \frac{1}{2}$.
- Gravity $g = 10$.

We have

$$M(q) = \begin{bmatrix} \frac{7}{2} + \cos(q_2) & \frac{5}{4} + \frac{1}{2}\cos(q_2) \\ \frac{5}{4} + \frac{1}{2}\cos(q_2) & \frac{5}{4} \end{bmatrix},$$

$$C(q, \dot{q}) = \begin{bmatrix} -\frac{1}{2}\sin(q_2)\dot{q}_2 & -\frac{1}{2}\sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ \frac{1}{2}\sin(q_2)\dot{q}_1 & 0 \end{bmatrix},$$

$$G(q) = \begin{bmatrix} 15\cos(q_1) + 5\cos(q_1 + q_2) \\ 5\cos(q_1 + q_2). \end{bmatrix}.$$

Why Continuous-time Setting?

For real humanoid robots, such as Atlas, we have $q \in \mathbb{R}^{28}$.

Figure: Humanoid robot Atlas in Boston Dynamics.

What Challenges for Continuous-time Problem?

Analytical solutions are challenging to obtain under complex dynamics and constraints.

Consider two-link planner robots with no constraints and quadratic costs:

$$\begin{aligned} \min_{x(t),u(t)} \quad & \|x(t_F)\|_2^2 + \int_{t_0}^{t_F} \|x(\tau)\|_2^2 + \|u(\tau)\|_2^2 \\ \text{s.t.} \quad & M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = u, \end{aligned}$$

where $x(t) = [q(t), \dot{q}(t)]$. In reality, we usually need $q_1 \in [0, \pi]$.

What Challenges for Continuous-time Problem?

Constraints are also hard to process in continuous LQR problems.

$$
\begin{aligned}
\min_{u(\cdot)} \quad & \int_0^\infty \|x(t)\|_2^2 + \|u(t)\|_2^2 dt \\
\text{s.t.} \quad & \dot{x} = Ax + Bu \\
& \|u(t)\|_2 \leq 1.
\end{aligned}
$$

The HJB equation becomes

$$
0 = \min_{\|u(t)\|_2 \leq 1} \left[\frac{\partial V}{\partial x} (Ax(t) + Bu(t)) + \|x(t)\|_2^2 + \|u(t)\|_2^2 \right].
$$

We need to convert infinite dimensional problem into finite dimensional approximation.

Outline

1. Introduction
   - Problem Definition
   - Classification of Computational Methods
   - Applications in Robotics

2. Computational Methods
   - Two Questions
   - Methods Overview
   - Direct Methods
   - Indirect Methods
   - Practical Issues

3. Other Methods for Trajectory Optimization

Methods Overview

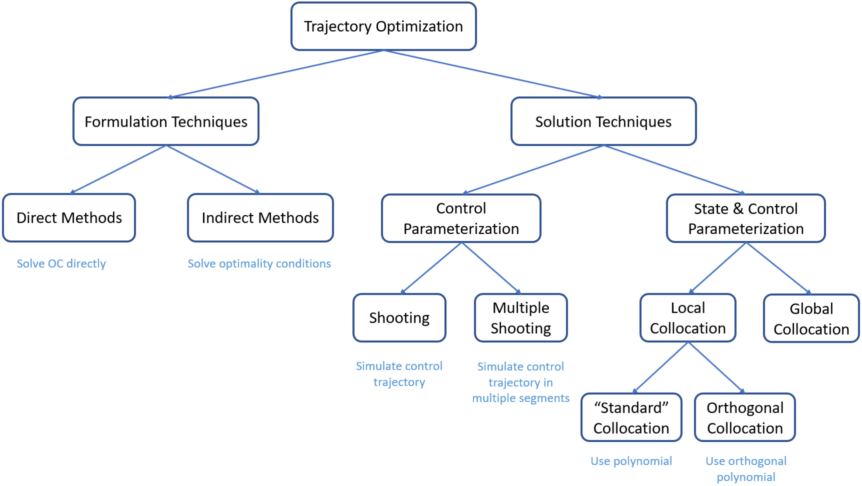


Figure: Overview of trajectory optimization methods.

Method Overview

General formulation of trajectory optimization problem:

$$
\begin{aligned}
\min_{u(\cdot)} \quad & J(u(\cdot)) := \Phi(x(t_F)) + \int_{t_0}^{t_F} L(\tau, x(\tau), u(\tau)) d\tau \\
\text{s.t.} \quad & \dot{x}(t) = f(t, x(t), u(t)), \\
& h(t, x(t), u(t)) \leq 0, \\
& g(t_F, x(t_F)) \leq 0, \\
& u(t) \in \mathcal{U}.
\end{aligned}
\tag{TO}
$$

Method Overview

From formulation techniques:

- Direct methods directly work on (TO) and parameterize the problem using different solution techniques.
- Indirect methods construct optimality conditions of (TO) and solve the conditions using different solution techniques.

From solution techniques:

- Shooting: parameterize $u(t)$ and simulate state trajectories; then optimize $u(t)$.
- Multiple shooting: parameterize $u(t)$ and simulate state trajectories in multiple segment; then optimize $u(t)$.
- Collocation: parameterize $u(t)$ and state trajectories, and specify their dynamic relationship; then optimize $u(t)$ and state trajectories.

Outline

Direct Shooting

Idea: parameterize the control and simulate the trajectory.

Ways to parameterize control $u(t)$:

- Base function approximation, $u_\theta(t) = \sum_{i=1}^{c} \theta_i \psi_i(t)$.
- Common choice of $\psi_i(t)$: splines, B-splines[2].

Specifically,

- Simulate $x(t)$ using f and u_θ.
- Decision variable θ.
- Use finite difference to compute gradient.
- Need a good heuristic if constraints exist, i.e., $h(x(t), u_\theta(t)) \leq 0$.

[2] A spline that passes n given knot points.

Direct shooting

Algorithm 1: Direct shooting.

Input: Initial condition x_0, initial parameter $\theta^{(0)}$;
for $n = 0, 1, \ldots$ **do**
  Integrate $x(t_F)$ and compute $J(\cdot)$ and $g(t_F, x(t_F))$ using $u_{\theta^{(n)}}$;
  Let $G = [J, g]$, evaluate $\frac{\partial G(\theta^{(n)})}{\partial \theta_i} = [G(\theta^{(n)} + \delta_i) - G(\theta^{(n)})]/\delta_i$;
  if $g(t_F, x(t_F)) \leq 0$ **and** $\|\frac{\partial J(\cdot)}{\partial \theta}\| < \epsilon$ **then**
    break;
  Update $\theta^{(n+1)}$ using $\frac{\partial G(\theta^{(n)})}{\partial \theta}$;

Reason to write $G = [J, g]$: use compatible simulations to evaluate gradients of different functions.

Direct Shooting

We simulate the entire trajectory to reach the target, acting like shooting.

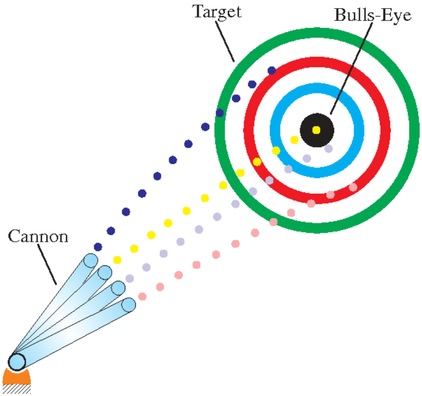


Figure: Schematic of shooting methods, from (Rao, 2009).

Direct Shooting — Example

We use the planner robot as an example.

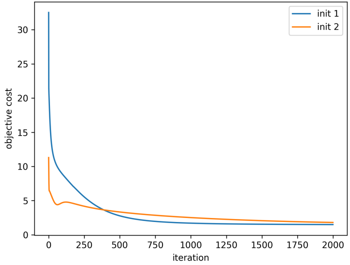
$$
\begin{aligned}
\min_{u} \quad & \|x_F - x_d\|_2^2 + \int_0^T u^2(t)dt \\
\text{s.t.} \quad & x_F = \begin{bmatrix} l_1 \cos(q_1(T)) + l_2 \cos(q_1(T) + q_2(T)) \\ l_1 \sin(q_1(T)) + l_2 \sin(q_1(T) + q_2(T)) \\ v_1(T) \\ v_2(T) \end{bmatrix}, \\
& \dot{q} = v, \\
& \dot{v} = M^{-1}(q)\left(-C(q,v) - G(q) + u\right), \\
& 0 \leq q_1(t) \leq \pi, \\
& -1 \leq u_i(t) \leq 1, \quad i = 1, 2.
\end{aligned}
$$

Direct Shooting — Example

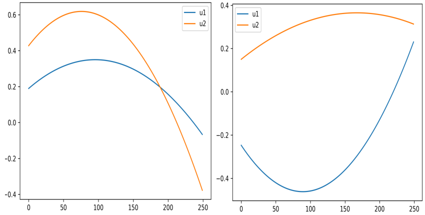
We parameterize u_i with $\theta_{0i} + \theta_{1i}t + \theta_{2i}t^2$, $i = 1, 2$.

- Initialization 1: $\theta_{0i} = \theta_{1i} = \theta_{2i} = 0.1$, $i = 1, 2$.
- initialization 2: $\theta_{0i} = \theta_{1i} = 0.1, \theta_{2i} = 0$, $i = 1, 2$.

(Two external animations)



(a) Objective in GD



(b) Control

Figure: Direct shooting simulation. Sensitive to small changes. Step size $\alpha = 1e - 5$.

Direct Shooting

Advantage:

- Effective for simple dynamics and no path constraints. e.g., launch rockets, orbit transfer, and spacecraft control.
- Small number of decision variables. Fast computation.

Issues:

- Sensitivity. Perturbations near $u(t_0)$ propagate along the trajectory.
- A single gradient evaluation requires simulating the trajectory.
- Integration accuracy decreases for complex dynamics.
- Bad heuristic can worsen the computation.

Direct Multiple Shooting

To address integration accuracy and sensitivity issues:

- Divide $[t_0, t_F]$ into multiple intervals to process.
- Parameterize $u(t)$ in each interval (in contrast with direct shooting).
- Enforce continuity.

Specifically,

- Divide $[t_0, t_F]$ into $[t_k, t_{k+1}]$, $k = 0, \ldots, K-1$.
- Parameterize $u(t)$ in each $[t_k, t_{k+1}]$ with θ_k, $k = 0, \ldots, K-1$.
- Determine state at time t_k: $\{x(t_k) := x_k\}_{k=0}^{K}$.
- Simulate $\tilde{x}(t_{k+1})$ using u_{θ_k} and x_k.
- Set $\tilde{x}(t_k) = x_k$ for each $k = 1, \ldots, K$.
- Decision variables $\langle \{x_k\}_{k=1}^{K}, \{\theta_k\}_{k=0}^{K-1} \rangle$.

Direct Multiple Shooting

- x_k are decision variables.
- Continuity error $d_k = \int f(x_k, u_k)dt - x_{k+1}$. We want $d_k = 0$.

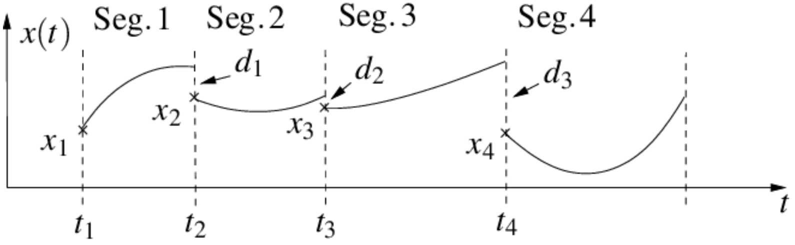


Figure: Schematic of multiple shooting methods, from (Rantil et al., 2009).

Direct Multiple Shooting

Algorithm 2: Direct Multiple Shooting.

Input: K, initial conditions $\{x_k^{(0)}\}_{k=1}^{K}$ and $\{\theta_k^{(0)}\}_{k=0}^{K-1}$;
for $n = 0, 1, \ldots$ **do**
    Integrate $\tilde{x}(t_k)$ using $u_{\theta_k^{(n)}}$ and $x_{k-1}^{(n)}$;
    Evaluate $d_k^{(n)} = \tilde{x}(t_k) - x_k^{(n)}$ and $c = \sum_{k=1}^{K} \|d_k^{(n)}\|_2^2$;
    Evaluate $J(\cdot)$ and $g(t_F, x_K^{(n)})$;
    Let $G = [J, g, c]$, evaluate $\frac{\partial G}{\partial \theta}$ and $\frac{\partial G}{\partial x}$;
    if $c < \epsilon$ **and** $g \leq 0$ **and** $\|\frac{\partial J}{\partial(\theta, x)}\| < \epsilon$ **then**
        break;
    Update $\{x_k^{(n+1)}\}_{k=1}^{K}$ and $\{\theta_k^{(n+1)}\}_{k=0}^{K-1}$ with $\frac{\partial G}{\partial x}$ and $\frac{\partial G}{\partial \theta}$;

Direct Multiple Shooting — Example

Ways to parameterize $u(t)$ in $[t_k, t_{k+1}]$:

- $u_\theta(t) = \sum_i \theta_i \psi_i(t)$.
- Constant, $u_\theta(\tau) = \theta_k,\ t_k \leq \tau < t_{k+1}$. (zero-order holder).

We use linear systems[3] as an example:

$$\dot{x} = Ax + Bu,\ x(0) = x_0 \quad \Rightarrow \quad x_k = A_d x_k + B_d u_k,\ x_k := x(t_k),$$
$$\Rightarrow \quad x_K = A_d^K x_0 + B_d u_{K-1} + \cdots + B_d^{K-1} u_0.$$

The optimization variables are $\{u_k\}_{k=0}^{K-1}$.

$$h(x_K) := h(u_0, \ldots, u_{K-1}), \quad J(u_0, \ldots, u_{K-1}) = \cdots .$$

[3] We can integrate $x(t_k)$ analytically in linear systems.

Direct Multiple Shooting

Advantages:

- Enhance robustness.
- Parallel implementation of trajectory simulations.

Issues (similar to direct shooting):

- Still need numerical integration for dynamics. Challenging for complex dynamics.
- Gradient evaluation relies on trajectory simulation.
- Hard to incorporate path constraints.
- Increased number of decision variables compared with direct shooting.

Direct Collocation

Issues for shooting and multiple shooting:

- Need an ODE solver to integrate state trajectories.
- Evaluate gradient requires integration.
- Challenging to deal with path constraints.

Integration introduces errors anyway, why not parameterize the state?

- Divide $[t_0, t_F]$ in to $[t_k, t_{k+1}]$, $k = 0, \ldots, K-1$.
- Decision variables $\{x_{k+1}, u_k\}_{k=0}^{K-1}$.
- Ensure path constraints are valid at (x_k, u_k) for each k.
- Ensure terminal constraints are valid at x_K.

Problems:

- No numerical integrator. What is the relationship between x_k and u_k?

Detour — Collocation for ODE

A **collocation method** uses a finite-dimensional candidate solution (usually polynomials) to approximate the solution of ODEs, PDEs, or integral equations. The candidate solution satisfies the given equation at a number of points called collocation points.

$$
\begin{aligned}
\dot{y}(t) = f(t, y(t)) \quad &\Rightarrow \quad y(t) = y(t_k) + \int_{t_k}^{t} f(\tau, y(\tau))d\tau, \ \ t \in [t_k, t_{k+1}], \\
&\Rightarrow \quad y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(\tau, y(\tau))d\tau..
\end{aligned}
$$

Collocation points:

- $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_N$ and $\tau_1 = t_k$, $\tau_N = t_{k+1}$.
- Values at the collocations $y(\tau_n)$, $n = 1 \ldots, N$.

Detour — Collocation for ODE

$$y(t) = y(t_k) + \int_{t_k}^{t} f(\tau, y(\tau)) d\tau, \quad t \in [t_k, t_{k+1}].$$

We use a polynomial of degree N to approximate $y(t)$ in $[t_k, t_{k+1}]$:

$$\tilde{y}(t) = a_0 + a_1(t - t_k) + \cdots + a_N(t - t_k)^N.$$

- The degree N equals to the number of collocation points.

We want to select the coefficients $\{a_n\}_{n=0}^N$ such that

- $\tilde{y}(t_k) = y(t_k)$.
- collocation constraints: $\dot{\tilde{y}}(\tau_i) = f(\tau_i, y(\tau_i)), \quad i = 1, \ldots, N$.

Direct Collocation

Back to the problem, we denote

$$\delta_k := t_{k+1} - t_k, \quad x_k := x(t_k), \quad u_k := u(t_k).$$

Forward Euler ($N = 1$):

- Collocation points: $t_0, t_2, \ldots, t_{K-1}$.

$$\begin{aligned}
\dot{x} = f(t, x, u) \quad &\Rightarrow \quad x_{k+1} - x_k = \delta_k f(t_k, x_k, u_k), \\
h(x(t), u(t)) \leq 0 \quad &\Rightarrow \quad h(x_k, u_k) \leq 0, \quad \forall k, \\
g(x(t_F)) \leq 0 \quad &\Rightarrow \quad g(x_K) \leq 0 \\
u(t) \in \mathcal{U} \quad &\Rightarrow \quad u_k \in \mathcal{U}, \quad \forall k, \\
\int_{t_0}^{t_F} L(x(t), u(t))dt \quad &\Rightarrow \quad \sum_{t=0}^{K-1} \delta_k L(x_k, u_k).
\end{aligned}$$

Direct Collocation

Backward Euler ($N = 1$):

- Collocation points: $t_1, t_2, \ldots, t_K$.

$$\dot{x} = f(t, x, u) \quad \Rightarrow \quad x_{k+1} - x_k = \delta_k f(t_{k+1}, x_{k+1}, u_{k+1}),$$

$$\int_{t_0}^{t_F} L(x(t), u(t))dt \quad \Rightarrow \quad \sum_{t=1}^{K} \delta_k L(x_k, u_k).$$

Trapezoidal collocation ($N = 2$)[4]:

- Collocation points: $t_0, t_2, \ldots, t_K$.

$$\dot{x}(t) = f(t, x(t), u(t)) \quad \Rightarrow \quad x_{k+1} - x_k = \frac{1}{2}\delta_k (f_{k+1} - f_k),$$

$$\int_{t_0}^{t_F} L(x(t), u(t))dt \quad \Rightarrow \quad \sum_{t=1}^{K} \frac{1}{2}\delta_k (L_k + L_{k+1}).$$

[4] $f_k := f(t_k, x_k, u_k), L_k := (x_k, u_k)$

Direct Collocation

Hermite–Simpson collocation ($N = 3$):

- Collocation points: $t_0, t_{\frac{1}{2}}, t_1, \ldots, t_{K-1}, t_{K-\frac{1}{2}}, t_K$.

$$\dot{x}(t) = f(t, x(t), u(t)) \quad \Rightarrow \quad \begin{cases} x_{k+1} - x_k = \frac{1}{6}\delta_k(f_k + 4f_{k+\frac{1}{2}} + f_{k+1}) \\ x_{k+\frac{1}{2}} = \frac{1}{2}(x_{k+1} + x_k) + \frac{1}{8}\delta_k(f_k - f_{k+1}) \end{cases}$$

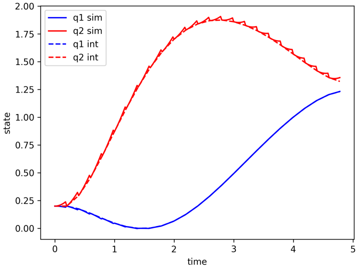
$$h(x(t), u(t)) \leq 0 \quad \Rightarrow \quad h(x_k, u_k) \leq 0, h(x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}}) \leq 0.$$

$$\int_{t_0}^{t_F} L(x(t), u(t))dt \quad \Rightarrow \quad \sum_{k=0}^{K-1} \frac{1}{6}\delta_k(L_k + L_{k+\frac{1}{2}} + L_{k+1}).$$

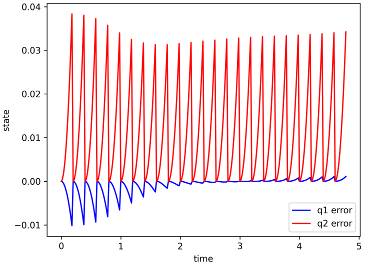
Attention: Objective approximation should be consistent with the dynamics approximation.

Direct Collocation — Example

(One external animation)



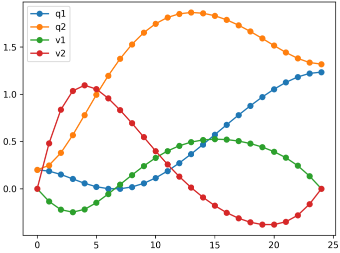
(a) Interpolation and simulated trajectory.



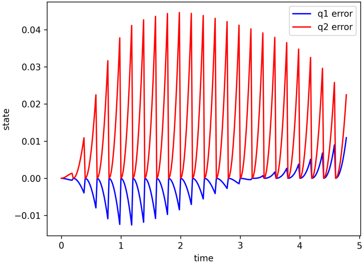
(b) Errors along the trajectory.

Figure: Forward Euler simulation results.

Direct Collocation — Example



(a) State trajectory.



(b) Errors along the trajectory.

Figure: trapezoidal collocation simulation results.

Direct Collocation

Advantages:

- Straightforward for most applications.
- Easy to handle constraints.
- Sparse gradient matrices leads to a sparse NLP.

Notes:

- In practice, Hermite-Simpson collocation gives satisfactory results.
- Higher-order collocation requires more computation, may not be necessary.
- Progressive refinement: first forward Euler, then trapezoidal, and then Hermite-Simpson.
- Initial guess matters.

Other Collocation Methods

Collocation methods can be further categorized into

- **local collocation**: select polynomials collocation points in each time interval $[t_k, t_{k+1}]$.
- **global collocation**: select polynomial and collocation points in $[t_0, t_F]$.

Orthogonal Collocation methods (local collocation)

- use zeros of certain polynomial as collocation points;
- use orthogonal polynomial as basis, such as Chebyshev polynomials and Legendre polynomials.

Outline

Indirect Methods

Recall the trajectory optimization problem:

$$\begin{aligned}\min_{u(\cdot)} \quad & J(u(\cdot)) := \Phi(x(t_F)) + \int_{t_0}^{t_F} L(\tau, x(\tau), u(\tau))d\tau \\ \text{s.t.} \quad & \dot{x}(t) = f(t, x(t), u(t)), \\ & h(t, x(t), u(t)) \leq 0, \\ & g(t_F, x(t_F)) \leq 0, \\ & u(t) \in \mathcal{U}.\end{aligned} \tag{TO}$$

We investigate optimality conditions (counterpart of KKT conditions in infinite dimensional spaces).

Indirect Methods

Define Hamiltonian $H(t, x, u, \lambda, \mu) = L + \lambda^T f + \mu^T h$,

- $\lambda(t)$ is the costate;
- $\mu(t) \geq 0$, $\nu \geq 0$ are Lagrangian multipliers for path and terminal constraints.

Optimality conditions for $\langle x, u^*, \lambda, \mu, \nu \rangle$:

$$\begin{aligned}
&\dot{x} = \frac{\partial H}{\partial \lambda}, \quad \dot{\lambda} = -\frac{\partial H}{\partial x}, \\
&u^* = \arg\min_{u \in \mathcal{U}} H, \\
&\lambda(t_F) = \frac{\partial \Phi}{\partial x(t_F)} + \nu^T \frac{\partial g}{\partial x(t_F)}, \\
&\mu^T h = 0, \quad \mu \geq 0, \quad h \leq 0, \\
&\nu^T g = 0, \quad \nu \geq 0, \quad g \leq 0.
\end{aligned} \tag{opt}$$

Indirect Methods

Notes:

- Any solution $\langle x(t), u(t), \lambda(t), \mu(t), \nu \rangle$ of the conditions is an *extramal*.
- In practice, $\arg\min_u H$ is often replaced by $\frac{\partial H}{\partial u} = 0$.
- Calculus of variations and optimal control theory: A concise introduction. (Liberzon, 2009).

Indirect methods:

- Indirect shooting.
- Indirect multiple shooting.
- Indirect collocation.

Indirect Shooting

Idea: parameterize $u_\theta(t)$ and simulate $x(t)$ and $\lambda(t)$.

- Decision variables θ, λ_0, ν.
- Use finite difference to compute gradient.

$$\begin{aligned} \nabla_u H &= 0, \\ \frac{\partial \Phi(x(t_F))}{\partial x} + \nu^T \frac{\partial g(x(t_F))}{\partial x} - \lambda(t_F) &= 0, \\ \nu^T g &= 0, \\ \nu \geq 0,\ g(x(t_F)) &\leq 0. \end{aligned} \tag{3}$$

- Equivalent to $F(z) = 0, G(z) \leq 0$. Newton's method.
- Complementarity constraints needs good initial guesses.
- Difficult to deal with path constraints because $\mu(t)$ is a trajectory.
- Usually unstable, good heuristic is required.

Indirect Multiple Shooting

Idea:

- Divide $[t_0, t_F]$ into K intervals $[t_k, t_{k+1}]$.
- Decision variables $\theta, \{x_k\}_{k=1}^K, \{\lambda_k\}_{k=0}^K$.
- Parameterize u_θ and simulate x and λ in each interval.

$$\tilde{x}_k = \int \frac{\partial H}{\partial \lambda}(u_\theta, x_k)dt, \quad \tilde{\lambda}_k = \int -\frac{\partial H}{\partial x}(u_\theta, \lambda_k)dt.$$

- Enforce continuity.

$$\tilde{x}_k - x_{k+1} = 0, \quad \tilde{\lambda}_k - \lambda_{k+1} = 0. \tag{4}$$

We solve (3) + (4) in multiple shooting methods.

- Stability is improved.
- Still difficult to handle path constraints.

Indirect Collocation

Idea:

- Parameterize u_θ, $x(t)$ and $\lambda(t)$. Decision variables $\{x_k, u_k, \lambda_k\}_{k=0}^{K}$.
- Apply collocation conditions and ensure constraints

We use trapezoidal collocation as an example[5].

$$
\begin{aligned}
\dot{x} = \frac{\partial H}{\partial \lambda} \quad &\Rightarrow \quad x_{k+1} - x_k = \frac{1}{2}\delta_k\left(\frac{\partial H_k}{\partial \lambda} + \frac{\partial H_{k+1}}{\partial \lambda}\right), \\
\dot{\lambda} = -\frac{\partial H}{\partial x} \quad &\Rightarrow \quad \lambda_{k+1} - \lambda_k = \frac{1}{2}\delta_k\left(-\frac{\partial H_k}{\partial x} - \frac{\partial H_{k+1}}{\partial x}\right), \\
\nabla_u H = 0 \quad &\Rightarrow \quad \nabla_u H_k = 0, \quad \forall k, \\
g(x(t), u(t)) \leq 0 \quad &\Rightarrow \quad g(x_k, u_k) \leq 0, \quad \forall k, \\
h(x(t_F)) \leq 0 \quad &\Rightarrow \quad h(x_K) \leq 0, \\
\mu^T g = 0,\ \nu^T h = 0 \quad &\Rightarrow \quad \mu_k^T g(x_k, u_k) = 0,\ \mu^T h(x_K) = 0, \quad \forall k, \\
\mu(t) \geq 0, \nu \geq 0 \quad &\Rightarrow \quad \mu_k \geq 0, \nu \geq 0, \quad \forall k.
\end{aligned} \tag{5}
$$

[5] We denote $H_k := H(x_k, u_k, \lambda_k, \mu_k, \nu)$.

Indirect Collocation

We want to solve

$$F(z) = 0, \quad G(z) \leq 0.$$

- Complementarity constraints are inherently combinatorial. Gradient based methods is hard to explore new solutions.
- Mixed integer programming or relaxed conditions $\mu^T g \leq \epsilon, \nu^T h \leq \epsilon$.
- Solving (5) requires good initialization. Otherwise easy to diverge.

Outline

Practical Issues

Initialization:

- Both direct/indirect methods rely on initial guesses.
- Initialization for direct methods is easier to construct.
- Indirect methods are more sensitive to initialization and easier to diverge for bad initialization.

Combination of direct/indirect methods:

- Indirect methods generate more accurate solution (if converge) than direct methods.
- Use direct methods to initialize indirect methods.

Practical Issues

Mesh refinement in collocation:

- Solve collocation problems on a sequence of collocation meshes.
- Subsequent meshes have more points and (or) higher-order collocation methods.

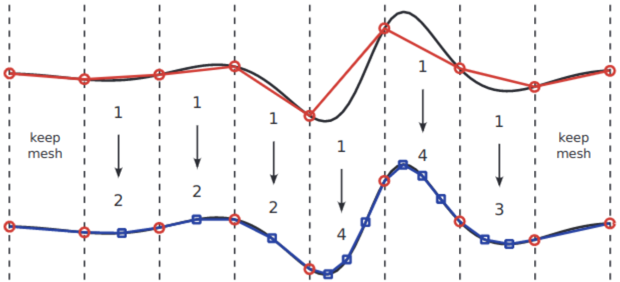


Figure: Schematic of mesh refinement, from (Kelly, 2017)

Outline

Other Methods

Model-based methods:

- Dynamic Programming (DP).
- Differential Dynamic Programming (DDP).
- Iterative Linear Quadratic Regulator (iLQR).
- Genetic Algorithms (other optimization methods)

Model-free methods:

- Reinforcement learning. Reward engineering

We use discrete systems to illustrate the methodology. For continuous systems, we discretize it and then apply the methods.

Dynamic Programming

We look for a stationary policy $\pi_t : \mathcal{U} \to \mathcal{X}$ of the problem:

$$\begin{aligned} \min_{u} \quad & \sum_{t=0}^{T} \gamma^t L(x_t, u_t) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \quad x_0 \text{ given.} \end{aligned}$$

Bellman equation[6] (fixed point equation):

$$V_t^{\pi_t}(x) = L(x, \pi_t(x)) + \gamma V_{t+1}^{\pi} \left(f(x, \pi_t(x)) \right), \quad \forall x.$$

When $T \to \infty$, π^t and V^{π_t} become stationary. For optimal policy:

$$V^*(x) = \min_u \left[L(x, u) + \gamma V^*(f(x, u)) \right], \quad \forall x.$$

[6] For continuous-time systems, the Bellman equation is a PDE.

Dynamic Programming

Common methods for DP:

- Backward computation, Riccati equation.
- Value iteration.
- Policy iteration.

Differential Dynamic Programming

DDP iteratively perturb value functions on a nominal trajectory $\langle \{\bar{x}_t\}_{t=0}^{T}, \{\bar{u}_t\}_{t=0}^{T-1} \rangle$ to generate new controls (David, 1966).

$$\begin{aligned} \min_{u} \quad & \Phi(x_T) + \sum_{t=0}^{T-1} L(x_t, u_t) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t). \end{aligned}$$

The value function V_t satisfies

$$V_t(x) = \min_{u} [L(x, u) + V_{t+1}(f(x, u))], \quad \forall x.$$

Now we perturb x by δx. We want to find the perturbed $\tilde{u} = u + \delta u$ as new controls.

Differential Dynamic Programming

We let

$$
\begin{aligned}
Q_t(\delta x, \delta u) &= L(x+\delta x, u+\delta u) + V_{t+1}(f(x+\delta x, u+\delta u)) \\
&\quad - L(x,u) - V_{t+1}(f(x,u)) \\
&\approx \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix},
\end{aligned}
$$

where $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{ux}$ are partial derivatives evaluated at $(\bar{x}_t, \bar{u}_t)$.

We have

$$
\delta u^* = \arg\min_{\delta u} Q_t(\delta x, \delta u) = -Q_{uu}^{-1}(Q_u + Q_{ux}\delta x) = k_t + K_t \delta x.
$$

Differential Dynamic Programming

Algorithm 3: Differential Dynamic Programming

Input: Nominal trajectory $\langle \bar{x}, \bar{u} \rangle$;
while *True* **do**
    `// Backward pass`
    for $t = T-1, \ldots, 0$ **do**
        Evaluate K_t, k_t at $(\bar{x}_t, \bar{u}_t)$;
    `// Forward pass`
    $x_0 = \bar{x}_0$;
    for $t = 0, \ldots, T-1$ **do**
        $u_t \leftarrow \bar{u}_k + k_t + K_t x_t$;
        $x_{t+1} \leftarrow f(x_t, u_t)$;
    $\langle \bar{x}, \bar{u} \rangle \leftarrow \langle x, u \rangle$;

Iterative Linear Quadratic Regulator

iLQR iteratively solves a QP on a nominal trajectory $\langle \{\bar{x}_t\}_{t=0}^{T}, \{\bar{u}_t\}_{t=0}^{T-1} \rangle$ to obtain new controls.

Let $x = \bar{x} + \delta x$, $u = \mp \delta u$. Linearize at $\langle \bar{x}, \bar{u} \rangle$:

$$
\begin{aligned}
\min_{\delta u} \quad & \delta x_T^T Q_T \delta x_T + \delta q_T^T \delta x_T \\
& + \sum_{t=0}^{T-1} x_t^T Q_t x_t + u_t^T R_t u_t + x_t^T S_t u_t + q_t^T x_t + r_t^T u_t \\
\text{s.t.} \quad & \delta x_{t+1} = A_t \delta x_t + B_t \delta u_t,
\end{aligned}
$$

- $q_t = \nabla_x L(\bar{x}_t, \bar{u}_t), r_t = \nabla_u L(\bar{x}, \bar{u})$.
- $Q_t = \nabla_{xx}^2 L(\bar{x}_t, \bar{u}_t), S_t = \nabla_{xu} L(\bar{x}_t, \bar{u}_t), R_t = \nabla_{uu} L(\bar{x}_t, \bar{u}_t)$.
- $A = \nabla_x f(x_t, u_t), B = \nabla_u f(x_t, u_t)$.

Summary

We have briefly introduced

- applications of trajectory optimization in robotics;
- numerical methods for solving continuous-time optimal control problems;
- common model-based methods for optimal control (discrete systems).

Ideas and tricks:

- The idea of parameterization and ODE approximation.
- Euler forward discretization is not the only option.
- First direct then indirect.

Additional Resources

Some useful and powerful solvers:

- MATLAB `FMINCON` (various methods)
- `scipy.optimize` (various methods, SQP for nonlinear programming)
- IPOPT (large scale nonlinear programming, interior point method)
- SNOPT (large scale sparse nonlinear programming, SQP)
- MOSEK (SDP, SOCP, convex optimization)
- Gurobi (convex optimization, mixed integer programming)
- CPLEX (convex optimization, mixed integer programming)

Some library for trajectory optimization:

- MATLAB `MPC`
- PSOPT (C++ interface)
- OpenOCL (inactive since 2019)