Optimization-based Motion Planning and Optimal Control

Jun Zeng

Department of Mechanical Engineering
University of California, Berkeley

February 25th 2020

Overview

1. Introduction
2. Optimization-based path planning
   - General methods
   - Examples
3. Optimal control
   - Model predictive control

1. Introduction

2. Optimization-based path planning
   - General methods
   - Examples

3. Optimal control
   - Model predictive control

Where we need path planning?

How to consider a robotics research as a systematic problem? Let's think about autonomous driving:

- Policy and Strategy:
  - Global: all safe and comfortable
  - Local: need to stay on right side and lane change to another highway.
- Vision: detect and analyze neighboring objects with semantic information
- Planner: Plan a collision-free and comfortable trajectory, e.g.
  - Sample various candidates trajectories
  - Select a best one according to some criteria
  - Smooth the speed profiles to make it likely to be dynamically feasible
- Control: Track trajectories with different control laws

Planning is the bridge between high-level and low-level.

- Need to consider noise and disturbance from vision module
- Make the trajectory dynamically-feasible for the control module and other low-level ones.

The performance of path planning is generally determined by two aspects from my personal perspective:

- Refresh rate: already a huge time delay from various vision sensors, e.g. you wouldn't expect the refresh rate to be less than 10Hz for autonomous driving.
- Correctness: RRT without smoothness is generally hard to track...

Optimal Control vs Optimization-based Path Planning

- Optimal control: linear–quadratic regulator (LQR), model predictive control (MPC), sliding mode control, etc.
- Topics in optimization-based path planning:
  - Speed profile smoothing: e.g. minimum-jerk in autonomous driving.
  - Contact force: e.g. hybrid mode switch with contact force in legged robotics.
  - Energy efficiency: e.g. minimum-snap/jerk in aerial robotics.
- They are the same in mathematics: an optimization problem is formulated to generate control strategy or trajectory/robot's motions.
- They are different in the implementation: optimal control is usually holding a higher refresh rate and needs to be solved very fast, while the optimization problem in planning isn't necessarily to be solved computationally with that fast rate.

Notation

Generally, an optimal control problem or an optimization-based path planning problem could be described as follow,

$$
\begin{aligned}
u^*(t) := \arg\min_{u(.)} \int_{t_0}^{t_f} & c_t(x(t), u(t))dt \\
s.t.\ x(t_0) &= x_0 \\
\dot{x}(t) &= f(x(t), u(t)) \quad \forall t \\
x(t) &\in \mathcal{X}_{feas} \quad \forall t \quad \text{(collision-free)} \\
u(t) &\in \mathcal{U}_{feas} \quad \forall t \quad \text{(control limits)}
\end{aligned}
$$

- In the view of control: generate feasible control inputs under dynamics constraints
- In the view of planning: generate dynamically-feasible waypoints (which will be tracked with appropriate control methods).

1 Introduction

2 Optimization-based path planning
- General methods
- Examples

3 Optimal control
- Model predictive control

Mathematical methods

Generally, a optimization problem will satisfy a variety of constraints, this leads us to have different methods to solve them numerically.
Some of them have explicit solutions:

- convex quadratic programming (Convex QP)
- LQR: For continuous-system linear system $\dot{x} = Ax + Bu$ with a quadratic cost $J = \int_0^\infty (x^T Qx + u^T Ru + 2x^T Nu)dt$. The feedback control law to minimize the value of the cost is $u = -Kx$, where $K = R^{-1}(B^T P + N^T)$ and $A^T P + PA - (PB + N)R^{-1}(B^T P + N^T) + Q = 0$. Here, the control law could be solved explicitly.

Numerical methods

Other general problems don't have explicit solutions and don't guarantee existing solution but could be solved numerically:
An optimization-based path planning problem could be formulated as several types:

- Shooting methods: we 'shoot' out trajectories in different directions until we find a trajectory that has the desired boundary value.
- Collocation methods: choose a finite-dimensional space of candidate solutions (e.g. polynomials) and a number of points in the domain (called collocation points), and to select that solution which satisfies the given equation at the collocation points.

Solvers and existence of solutions

Most likely, all problems could be formulated into a dynamic programming problem and could be solved with various numerical solvers[1]:

- for convex problems:
  - software: CVX, OSQP
  - internal solvers: Gurobi, Sedumi, Mosek
- for non-convex problems:
  - interior point: IPOPT, SNOPT
  - active set method: SAS

We are more interested in the existence of the solution of the optimization. For a convex problem, we are likely to ensure at least one solution (might have several ones if it's nonlinear or non-convex cost function, our numerical solution will depends on the initial point where we do gradient descent). But for general non-convex problem, we are unable to guarantee the existence of solution.

[1] More mathematical details for solving optimization problem are presented in EE 227A/B/C.

Trajectory smoothing

One of the most applicable areas about optimization-based planning is trajectory smoothing, which intends to make trajectory generated from the sample-based methods (RRT, PRM, etc) become smoother.

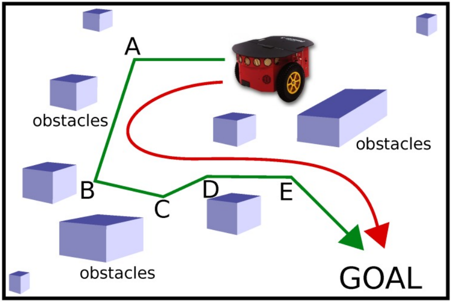


Figure: Make your trajectory smoother

Geometric interpolation

There are some traditional geometric methods for trajectory smoothing:

- Polynomial interpolation
- Bezier Curve [online playground]
- Cubic Splines
- B-Splined

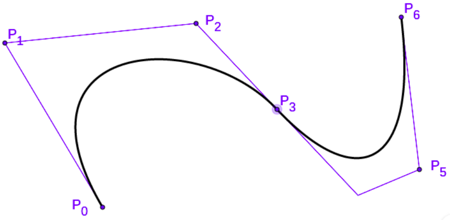
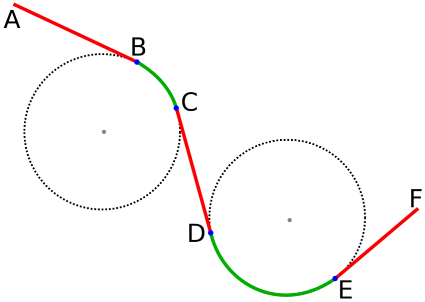


Figure: Bezier Curve

Special curves for smoothness

In industry, people prefer some special curves which are easy to be implemented with less complexity:

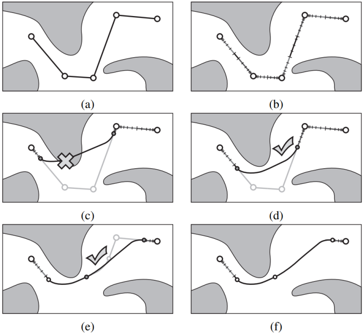
- Dubin's Curve (e.g. lane change in autonomous driving)



Smoothness doesn't necessarily guarantee dynamically-feasibility.

Optimization-based methods

The optimization-based methods could also deal with trajectory smoothing problem. For example, minimum-time for obstacle avoidance[2].



Here, only bounded acceleration is considered, dynamics is still excluded.

[2] Hauser and Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts".

Related work

- "Time Elastic Band" planner (TEB Planner) with ROS integration is introduced[3][4], where non-holonomic kinematics is considered, but roughly. [demo video in ROS]
- For autonomous driving, some special techniques are also proposed[5][6][7].

[3] Rösmann et al., "Trajectory modification considering dynamic constraints of autonomous robots".

[4] Rösmann et al., "Efficient trajectory optimization using a sparse model".

[5] Dolgov et al., "Path planning for autonomous vehicles in unknown semi-structured environments".

[6] Ziegler et al., "Trajectory planning for Bertha—A local, continuous method".

[7] Gu and Dolan, "On-road motion planning for autonomous vehicles".

There are many work about this area, the pipeline of path planning is mainly considered as trajectory generation + trajectory smooth, and the problem is formulated to consider mobile robots not holding too complex dynamics (autonomous cars, robot arm, etc).

How about generating a dynamically-feasible trajectory directly from an optimization-based problem? There are several tricky points to take into account.

- Obstacle avoidance and Safety
- Hybrid mode switch (contact force)
- Energy efficiency and smoothness (e.g. which kind of cost function we shall choose)

Obstacle avoidance

There are several ways to consider obstacle avoidance as constraints in an optimization-based problem:

- simple geometric constraints: e.g. $(x - x_{obs})^2 + (y - y_{obs})^2 \geq r^2$
  - very rough consideration: point-mass and round obstacle
  - nonlinear and non-convex constraints
  - it could be solved with modern solvers (IPOPT, etc) but doesn't guarantee a solution.

Obstacle avoidance

To make the problem become convex, an mixed-integer formulation could be introduced[8][9][10], which exploits the topological property of the collision-free space. We call it as "mixed-integer", since a list of integer variables is to used to represent the areas where the mobile robot is optimized to pass through.

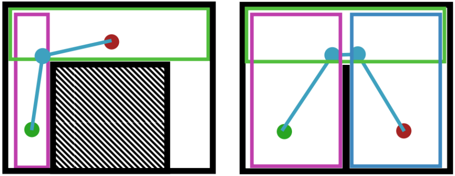


Figure: A trajectory in which each linear segment is required to remain entirely within one of the convex obstacle-free regions indicated by the colored boxes. This requirement ensures that the entire trajectory is collision-free.

[8] Richards and How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming".

[9] Deits and Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization".

[10] Deits and Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments".

Sometime we need to describe our robot and obstacles as rigid bodies, this leads us to define the distance between rigid body and obstacle avoidance constraint between them. [demo]

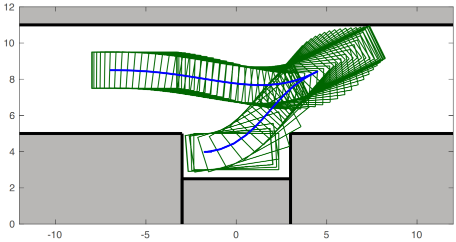


Figure: Dual variables are used to represent the collision-free constraints, see more details in[11].

[11] Zhang, Liniger, and Borrelli, "Optimization-based collision avoidance".

Obstacle avoidance

There are many other famous work:

- Sequential Convex Optimization[12]
- STOMP[13]: deployed on PR2 robot and recommended by ROS.
- CHOMP[14]: consider waypoint optimization and trajectory smoothness

[12] Schulman et al., "Motion planning with sequential convex optimization and convex collision checking".

[13] Kalakrishnan et al., "STOMP: Stochastic trajectory optimization for motion planning".

[14] Zucker et al., "Chomp: Covariant hamiltonian optimization for motion planning".

Other work

There are other famous work for special systems:

- Swarm path planning for UAV[15]
- Baidu Apollo open source planner[16]

[15] Roberge, Tarbouchi, and Labonté, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning".

[16] Fan et al., "Baidu apollo em motion planner".

Contact force

The intuition is simple: contact force is zero or distance between contact points should be zero. This formulates a complementarity constraint.

Generally, the methods is called *Optimization through contact*[17] and the problem is formulated as,

$$
\begin{aligned}
\text{find } & \ddot{q}, \lambda \\
\text{subject to } & H(q)\ddot{q} + C(q,\dot{q}) + G(q) = B(q)u + J(q)^T\lambda \\
& \phi(q) \geq 0 \\
& \lambda \geq 0 \\
& \phi(q)^T\lambda = 0
\end{aligned}
$$

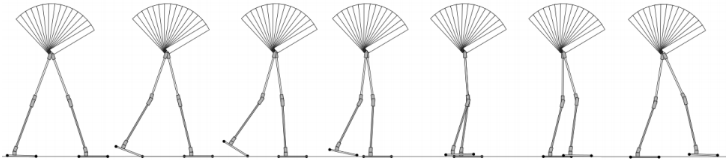
where λ represents the contact force. No longer need to consider the mode sequences.

[17] Posa, Cantu, and Tedrake, "A direct method for trajectory optimization of rigid bodies through contact".

Examples

Applicable cases:

- Manipulation with finger contact
- Legged robotics [demo]



Example

For example, the tension $T(t)$ in a massless cable could be considered as a contact force. Assume the length of cable is l_0 and the distance between two ends of the cable is $l(t)$, then we have $\phi(q) = l_0 - l(t) \geq 0$ with its complementarity constraint,

$$T(t)(l_0 - l(t)) = 0$$

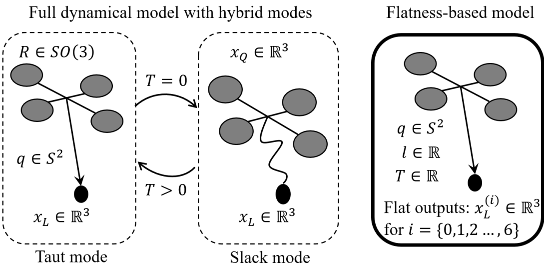


Figure: Hybrid modes switches in aerial manipulation[18]

[18] Zeng et al., "Differential Flatness based Path Planning with Direct Collocation on Hybrid Modes for a Quadrotor with a Cable-Suspended Payload".

Energy efficiency and smoothness

- A cost function to minimize jerk (third-order derivative of vehicle's position)[19].
- A cost function to minimize snap (fourth-order derivative of vehicle's position)[20].

Actually, people find that minimum jerk or snap usually performed better than those of other orders in robots for smoothness.

[19] Pattacini et al., "An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots".

[20] Mellinger and Kumar, "Minimum snap trajectory generation and control for quadrotors".

Discussion

The optimization-based path planning is still active research topic, there are many existing problems which need to be solved:

- multi-agents: e.g. ensure collision-free movements in drone swarm
- safety: e.g. guarantee safety-critical trajectory in a noisy environment (sensor noise, model uncertainty, algorithm bias, etc)
- policy: centralized/decentralized, human-robot interaction

Linear–quadratic regulator (LQR)

The LQR algorithm reduces the amount of work done by the control systems engineer to optimize the controller.

$$J^* = \min_{u_i} \sum_{k=0}^{N} x_{k+1}^T Q x_{k+1} + u_k^T R u_k$$

$$\text{s.t. } x_{k+1} = A x_k + B u_k$$

There is indeed *prediction* here, but feedback control gain is constant here.

Receding horizon control

Let's present model prediction control as follow,

$$
\begin{aligned}
J_t^*(x(t)) &= p(x_{t+N}) + \sum_{k=0}^{N-1} q(x_{t+k}, u_{t+k}) \\
\text{s.t.} x_{t+k+1} &= Ax_{t+k} + Bu_{t+k}, k = 0, ..., N-1 \\
x_{t+k} &\in \mathcal{X}, u_{t+k} \in \mathcal{U}, k = 0, ..., N-1 \\
x_{t+N} &\in \mathcal{X}_f \\
x_t &= x(t)
\end{aligned}
$$

Truncate after a finite horizon:

- N: horizon
- $p(x_{t+N})$: terminal cost which approximate the 'tail' of the cost.
- $q(x_{t+k}, u_{t+k})$: staged cost
- $\mathcal{X}_f$: Approximates the 'tail' of the constraints.

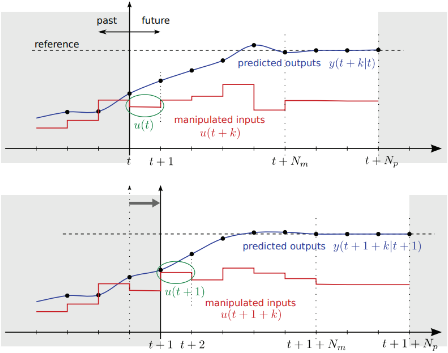


Figure: Constrained Finite Time Optimal Control (from Prof. Borrelli)

Where are the *Predictions*?

- At each sampling time, solve a constrained optimal control problem.
- Apply the optimal input only during $[t, t+1]$.
- At $t+1$, solve a CFTOC over a shifted horizon based on new state measurements.

Open-loop vs Closed-loop

Some definitions among researchers working on MPC:

- Open-loop trajectory: at each time step, we have a constrained optimal control problem, which solves a feasible trajectory with predictions.
- Closed-loop trajectory: since we solve optimal control problem at each time step, the robot will moves along a closed-loop trajectory with this control policy.

Discussions

- Terminal cost ensures the Lyapunov convergence, but it's not necessary.
- Larger horizon brings a bigger controllable set, but needs more time to calculate the optimization.
- The dynamics constraints could be nonlinear, which will make the problem as a dynamic programming problem.
- There are many variants of MPC: Robust MPC, Adaptive MPC, MPC with obstacle avoidance, etc.
- Recent work about Learning MPC takes safety into account. [demo]