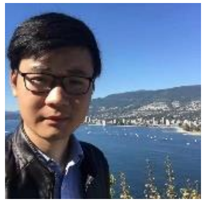
深蓝学院
shenlanxueyuan.com

Lecture 6

SOFT AND HARD CONSTRAINED TRAJECTORY OPTIMIZATION



主讲人 Fei Gao

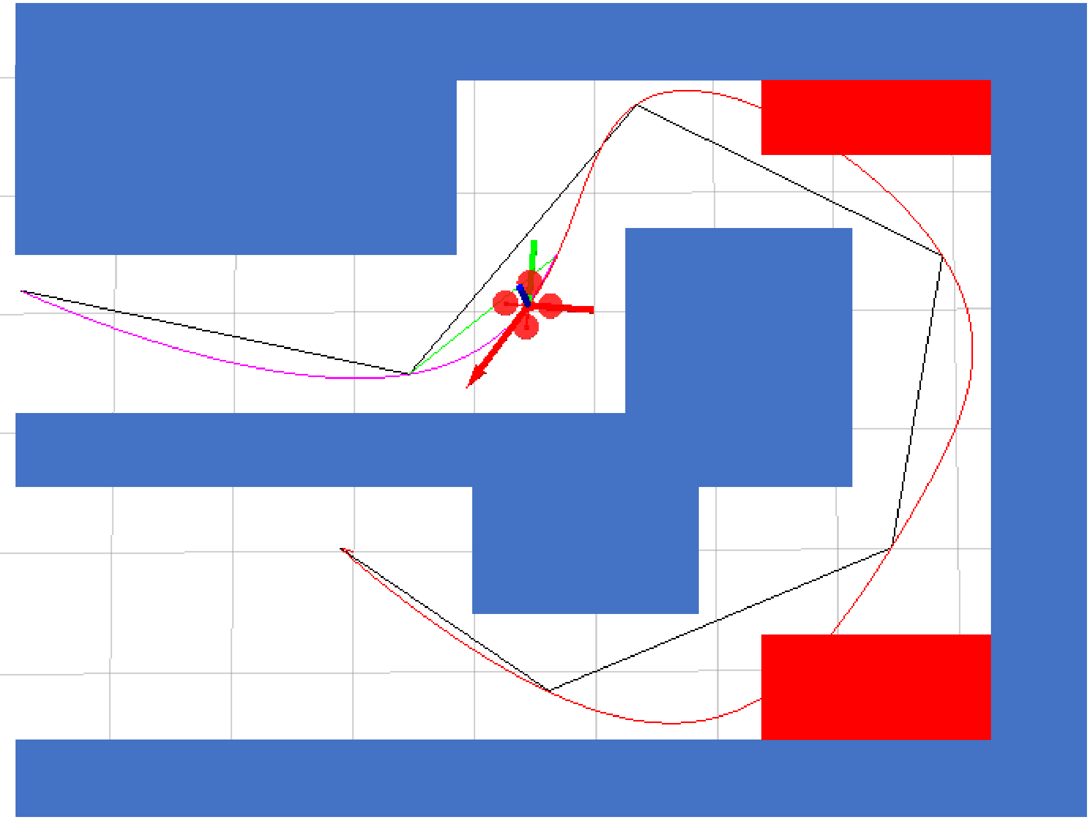
Ph.D. in Robotics
Hong Kong University of Science and Technology
Assistant Professor，Zhejiang University

Outline

1. Introduction
2. Soft-constrained Optimization
3. Hard-constrained Optimization
4. Case Study
5. Homework

Introduction

Minimum snap trajectory optimization

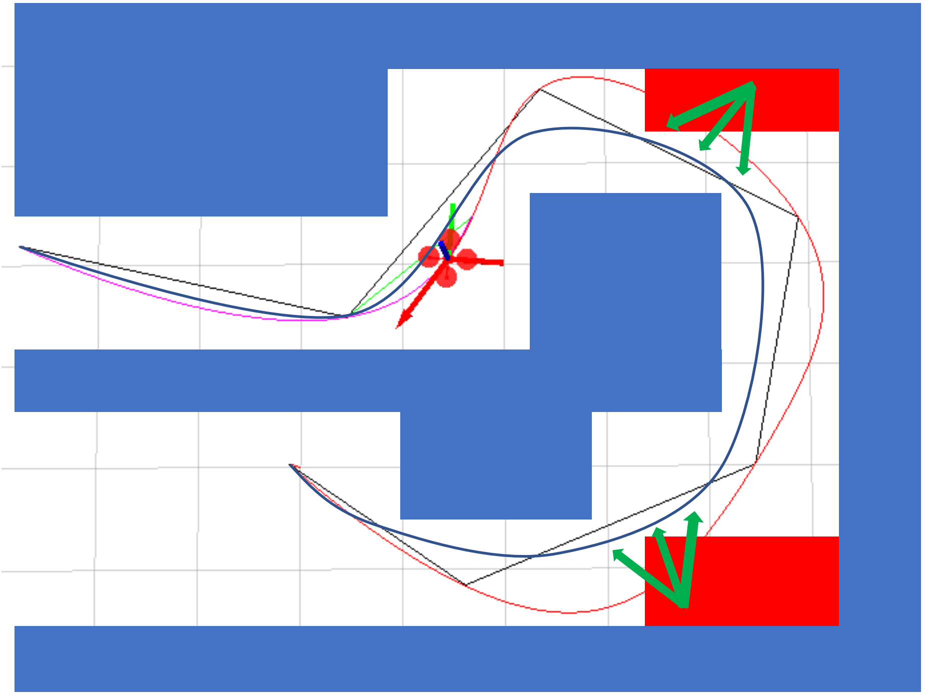


- We only constrain intermediate waypoints of the trajectory should pass.
- Computational cheap and easy to implement.
- No constraints on the trajectory itself.
- The “overshoot” of the trajectory unavoidable.

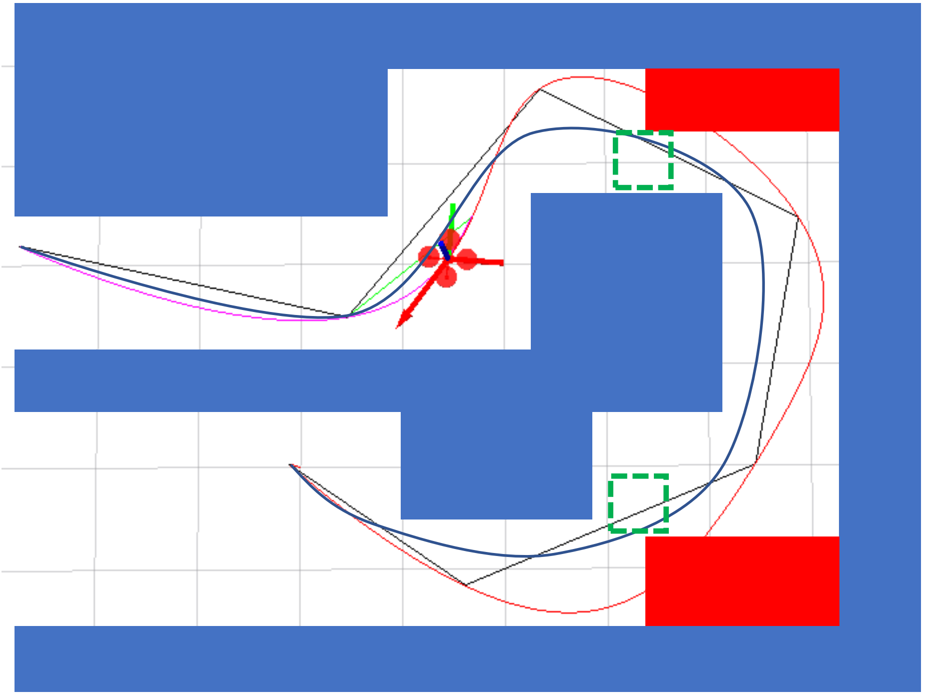
The basic minimum snap framework is good for smooth curve generation, not for collision avoidance.

Minimum snap with safety constraints

Adding forces



Adding bounds



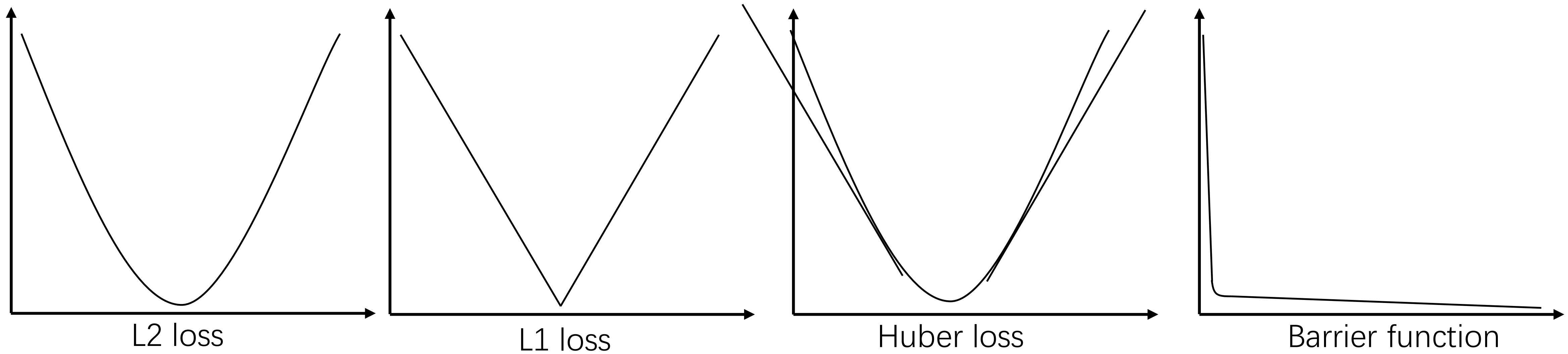
Hard/ Soft constraints

$$\begin{aligned} \min \quad & f(x) \\ s.t. \quad & \boxed{\begin{aligned} & g_i(x) = c_i, \quad i = 1, \cdots, n \quad \text{Equality constraints} \\ & h_j(x) \geq d_j, \quad j = 1, \cdots, n \quad \text{Inequality constraints} \end{aligned}} \end{aligned}$$

Required to be strictly satisfied.

$$\min \quad f(x) + \boxed{\lambda_1 \cdot g(x) + \lambda_2 \cdot h(x)}$$

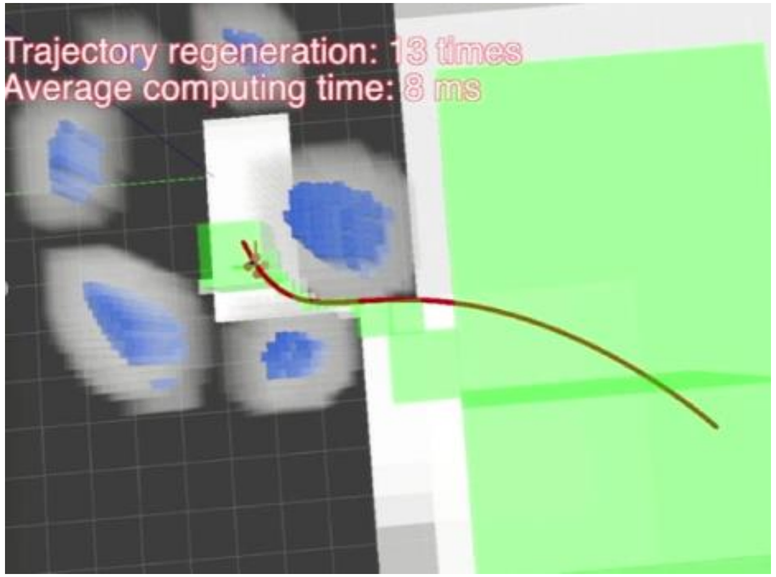
- Penalty terms / loss functions.
- Constraints which are preferred but not strictly required.
- Various kinds of loss functions.

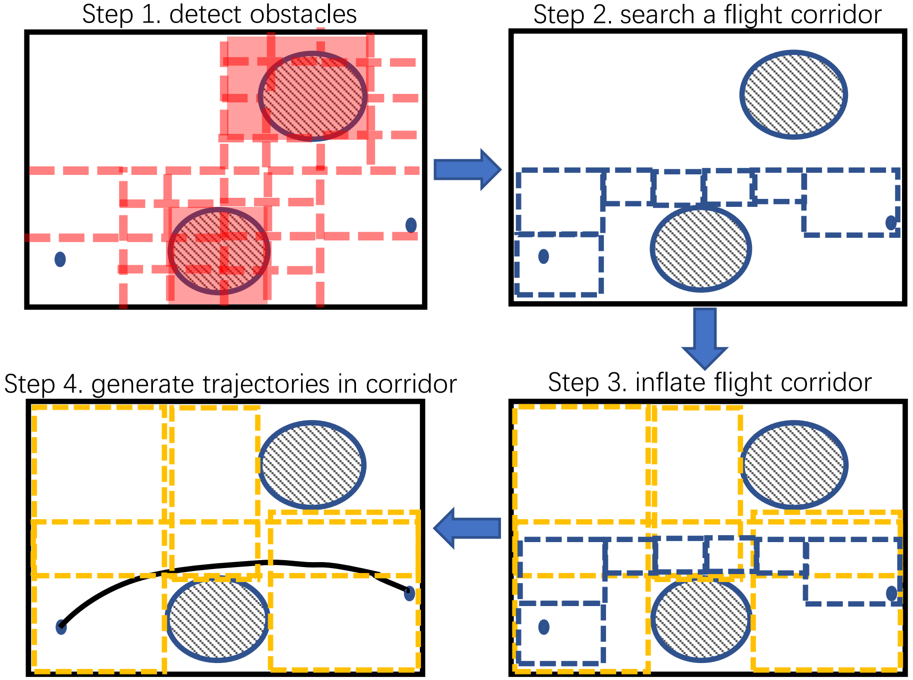


Hard-constrained Optimization

Corridor-based Trajectory Optimization

Corridor-based Smooth Trajectory Generation





- Differential flatness property

$$\{x, y, z, \dot{x}, \dot{y}, \dot{z}, \emptyset, \theta, \varphi, p, q, r\} \rightarrow \{x, y, z, \varphi\}$$

- Piecewise polynomial trajectory

$$f_\mu(t) = \begin{cases} \sum_{j=0}^{N} p_{1j}(t - T_0)^j & T_0 \leq t \leq T_1 \\ \sum_{j=0}^{N} p_{2j}(t - T_1)^j & T_0 \leq t \leq T_1 \\ \vdots & \vdots \\ \sum_{j=0}^{N} p_{Mj}(t - T_{M-1})^j & T_0 \leq t \leq T_1 \end{cases}$$

- Cost function (minimum jerk)

$$J = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k} \right)^2 dt$$

- Boundary constrains
- Continuity constrains
- Safety constrains

$$\begin{aligned} \min \ & \mathbf{p}^T \mathbf{H} \mathbf{p} \\ \text{s.t.} \ & \mathbf{A}_{eq} \mathbf{p} = \mathbf{b}_{eq} \\ & \mathbf{A}_{lq} \mathbf{p} \leq \mathbf{b}_{lq} \end{aligned}$$

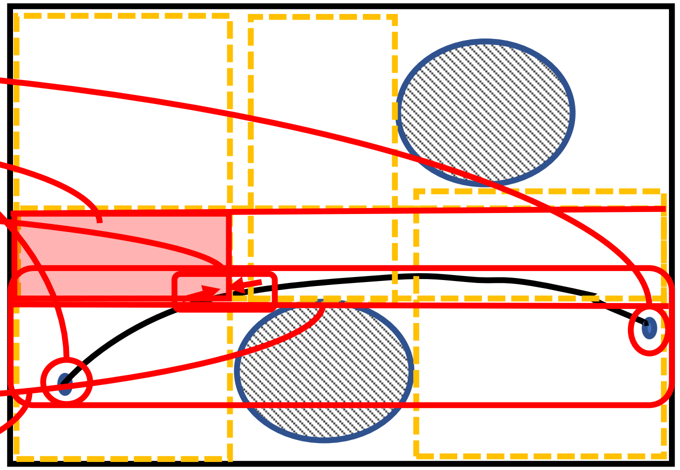
Quadratic Program

Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, **Jing Chen** et al.

Problem formulation

- Instant linear constraints:
  - Start, goal state constraint ($\mathbf{Ap} = \mathbf{b}$)
  - Transition point constraint ($\mathbf{Ap} = \mathbf{b}, \mathbf{Ap} \leq \mathbf{b}$)
  - Continuity constraint($\mathbf{Ap_i} = \mathbf{Ap_{i+1}}$)

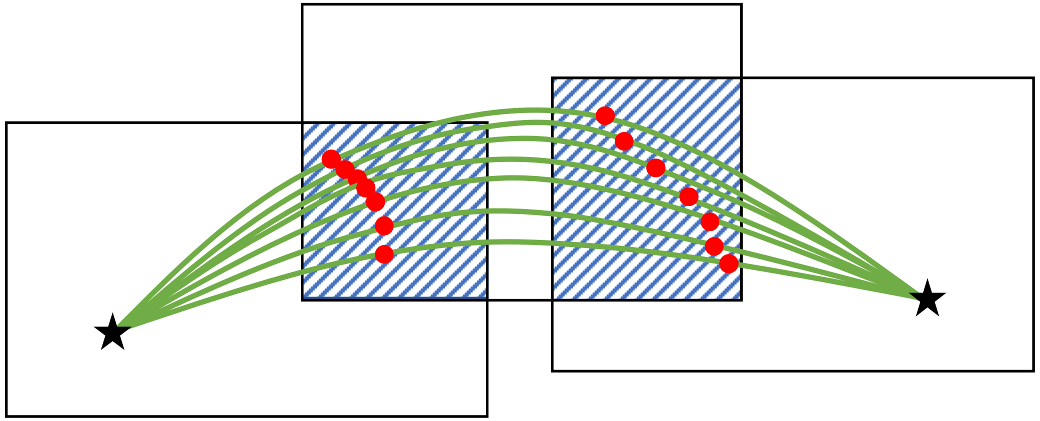
- Interval linear constraints:
  - Boundary constraint ($\mathbf{A}(t)\mathbf{p} \leq \mathbf{b}, \forall t \in [t_l, t_r]$)
  - Dynamic constraint ($\mathbf{A}(t)\mathbf{p} \leq \mathbf{b}, \forall t \in [t_l, t_r]$)
    - Velocity constraints
    - Acceleration constraints

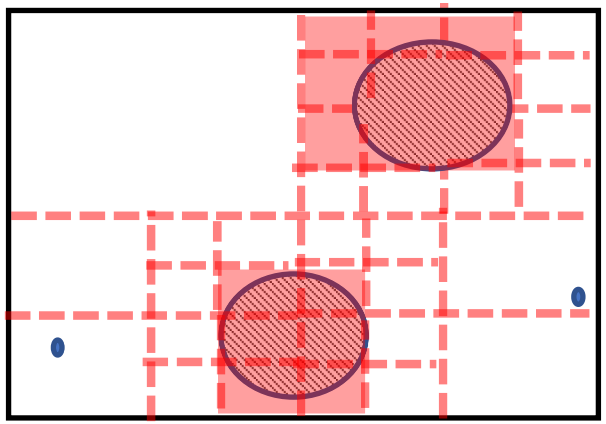


Advantages

Many advantages:

- Efficiency: path search in the reduced graph, convex optimization in the corridor are efficient.
- High quality: corridor provides large optimization freedom.





$$\min\ \mathbf{p}^T\mathbf{H}\mathbf{p}$$
$$\text{s.t.}\quad \mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq}$$
$$\mathbf{A}_{lq}\mathbf{p} \le \mathbf{b}_{lq}$$

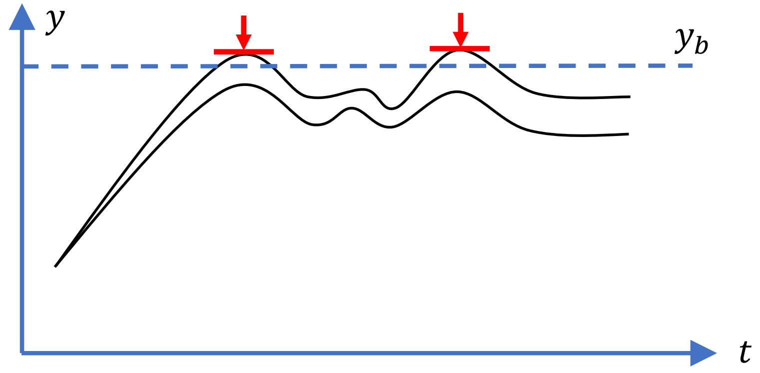
Quadratic Program

Disadvantages

Problem:

All constraints are enforced on piecewise joint points only, how to guarantee they are active along all the trajectory ?

- Iteratively check extremum and add extra constraints.



- Iterative solving is time consuming.
- If strictly no feasible solution meets all constraints. We have to run 10 iterations to determine the status of the solution ?

- Checking extremum is yet another polynomial root finding problem.
  - Up to quartic function, it's easy.
  - Higher order polynomials need numerical solutions.

$$f(x) = ax^4 + bx^3 + cx^2 + dx + e$$

Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, **Jing Chen** et al.

Polynomial roots finding

Matlab function “roots”

For a general polynomial function: $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$,
It has the **companion matrix**:

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}$$

The characteristic polynomial $\det(xI - A)$ of A is the polynomial $p(x)$.

roots

Polynomial roots

Syntax

```
r = roots(p)
```

Description

`r = roots(p)` returns the roots of the polynomial represented by `p` as a column vector. Input `p` is a vector containing `n+1` polynomial coefficients, starting with the coefficient of x^n. A coefficient of `0` indicates an intermediate power that is not present in the equation. For example, `p = [3 2 -2]` represents the polynomial $3x^2 + 2x - 2$.

The `roots` function solves polynomial equations of the form $p_1 x^n + \ldots + p_n x + p_{n+1} = 0$. Polynomial equations contain a single variable with nonnegative exponents.

Algorithms

The `roots` function considers `p` to be a vector with `n+1` elements representing the `n`th degree characteristic polynomial of an `n`-by-`n` matrix, `A`. The roots of the polynomial are calculated by computing the eigenvalues of the companion matrix, `A`.

```
A = diag(ones(n-1,1),-1);
A(1,:) = -p(2:n+1)./p(1);
r = eig(A)
```

The results produced are the exact eigenvalues of a matrix within roundoff error of the companion matrix, `A`. However, this does not mean that they are the exact roots of a polynomial whose coefficients are within roundoff error of those in `p`.

Bezier Curve Optimization

Trajectory basis changing

- Use **Bernstein** polynomial basis.

Bézier curve

- Change to basis of the trajectory from monomial polynomial to Bernstein polynomial

$$P_j(t) = p_j^0 + p_j^1 t + p_j^2 t^2 + \cdots + p_j^n t^n$$

$$B_j(t) = c_j^0 b_n^0(t) + c_j^1 b_n^1(t) + \cdots + c_j^n b_n^n(t) = \sum_{i=0}^{n} c_j^i b_n^i(t)$$

$$b_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}$$

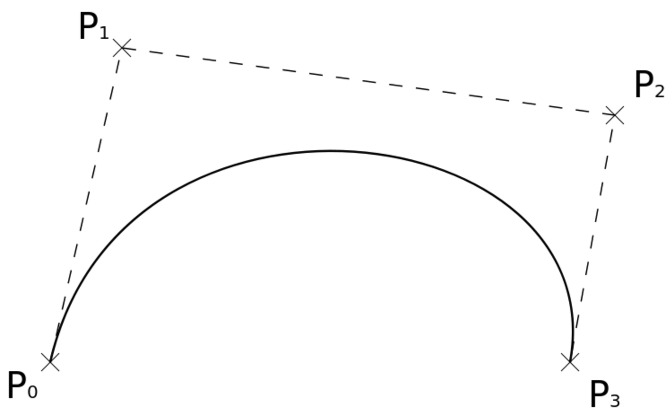
- Bézier curve is just a special polynomial, it can be mapped to monomial polynomial by: $p = M \cdot c$. And all previous derivations still hold.

For 6 order: $$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -6 & 6 & 0 & 0 & 0 & 0 & 0 \\ 15 & -30 & 15 & 0 & 0 & 0 & 0 \\ -20 & 60 & -60 & 20 & 0 & 0 & 0 \\ 15 & -60 & 90 & -60 & 15 & 0 & 0 \\ -6 & 30 & -60 & 60 & -30 & 6 & 0 \\ 1 & -6 & 15 & -20 & 15 & -6 & 1 \end{bmatrix}$$

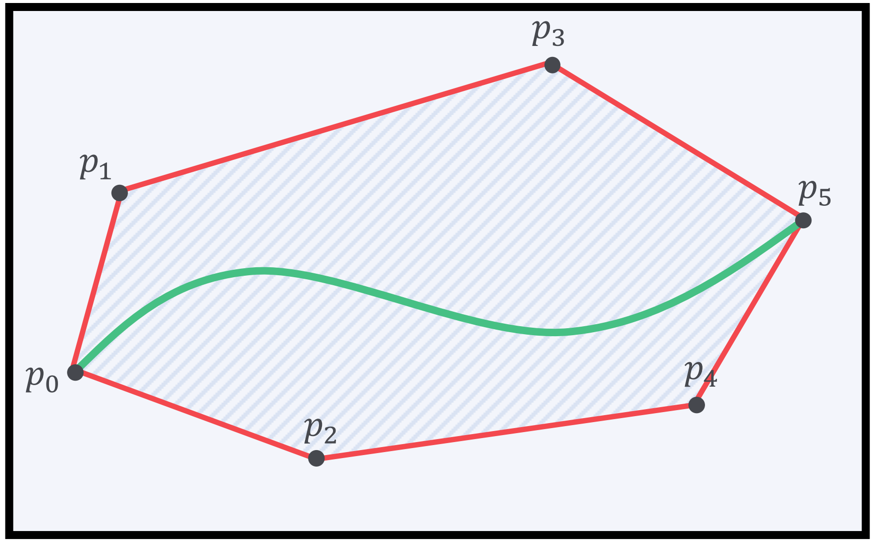
Trajectory basis changing

Properties:

- **Endpoint interpolation.** The Bezier curve always starts at the first control point, ends at the last control point, and never pass any other control points.
- **Convex hull.** The Bezier curve $B(t)$ consists of a set of control points ci are entirely confined within the convex hull defined by all these control points.
- **Hodograph.** The derivative curve $B'(t)$ of a Bezier curve $B(t)$ is called as hodograph, and it is also a Bezier curve with control points defined by $n \cdot (c_{i+1} - c_i)$, where n is the degree.
- **Fixed time interval**. A Bezier curve is always defined on [0,1]

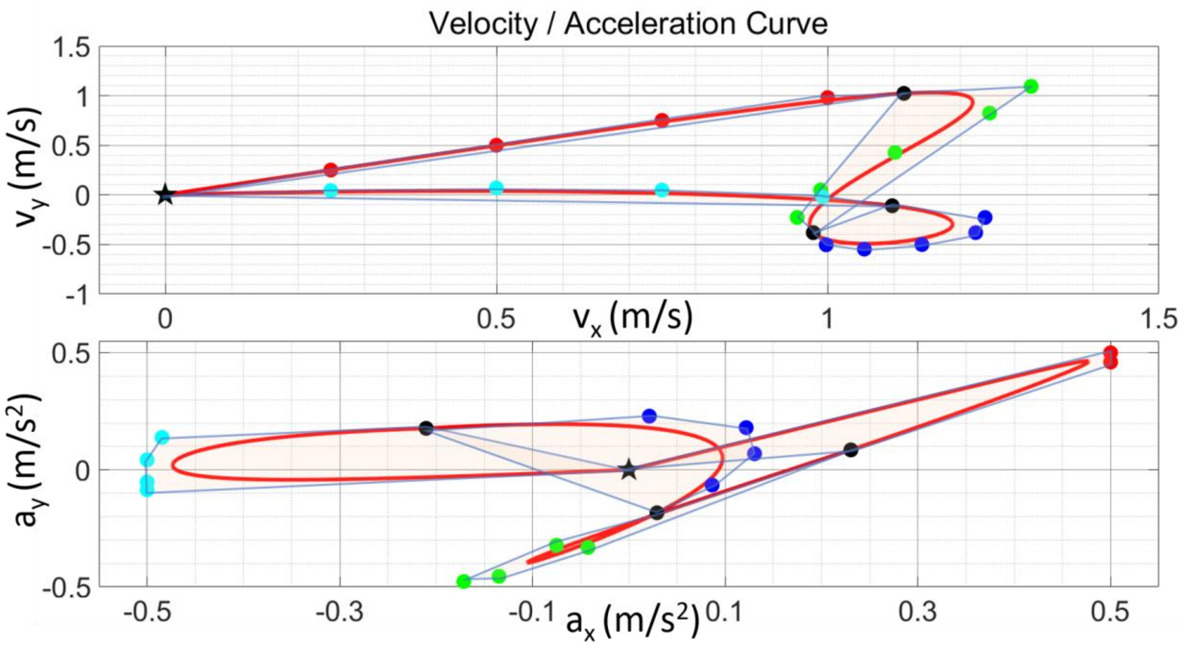


Convex hull property



- The flight corridor consists of convex polygons.
- Each cube corresponds to a piece of Bezier curve.
- Control points of this curve are enforced inside the polygon.
- The trajectory is entirely inside the convex hull of all points.

The trajectory is entirely inside the flight corridor.



Trajectory Generation Formulation

Define higher order (l^{th}) control points:

$$a_{\mu j}^{0,i} = c_{\mu j}^{i}, a_{\mu j}^{l,i} = \frac{n!}{(n-l)!} \cdot (a_{\mu j}^{l-1,i+1} - a_{\mu j}^{l-1,i}), \quad l \geq 1$$

- Boundary Constraints:

$$\boxed{a_{\mu j}^{l,0} \cdot s_j^{(1-l)} = d_{\mu j}^{(l)}}$$

- Continuity Constraints:

$$\boxed{a_{\mu j}^{\phi,n} \cdot s_j^{(1-\phi)} = a_{\mu,j+1}^{\phi,0} \cdot s_{j+1}^{(1-\phi)}, \quad a_{\mu j}^{0,i} = c_{\mu j}^{i}.}$$

- Safety Constraints:

$$\boxed{\beta_{\mu j}^{-} \leq c_{\mu j}^{i} \leq \beta_{\mu j}^{+}, \quad \mu \in \{x, y, z\}, \quad i = 0, 1, 2, ..., n,}$$

- Dynamical Feasibility Constraints:

$$\boxed{\begin{aligned} & v_m^{-} \leq n \cdot (c_{\mu j}^{i} - c_{\mu j}^{i-1}) \leq v_m^{+}, \\ & a_m^{-} \leq n \cdot (n-1) \cdot (c_{\mu j}^{i} - 2c_{\mu j}^{i-1} + c_{\mu j}^{i-2})/s_j \leq a_m^{+} \end{aligned}}$$

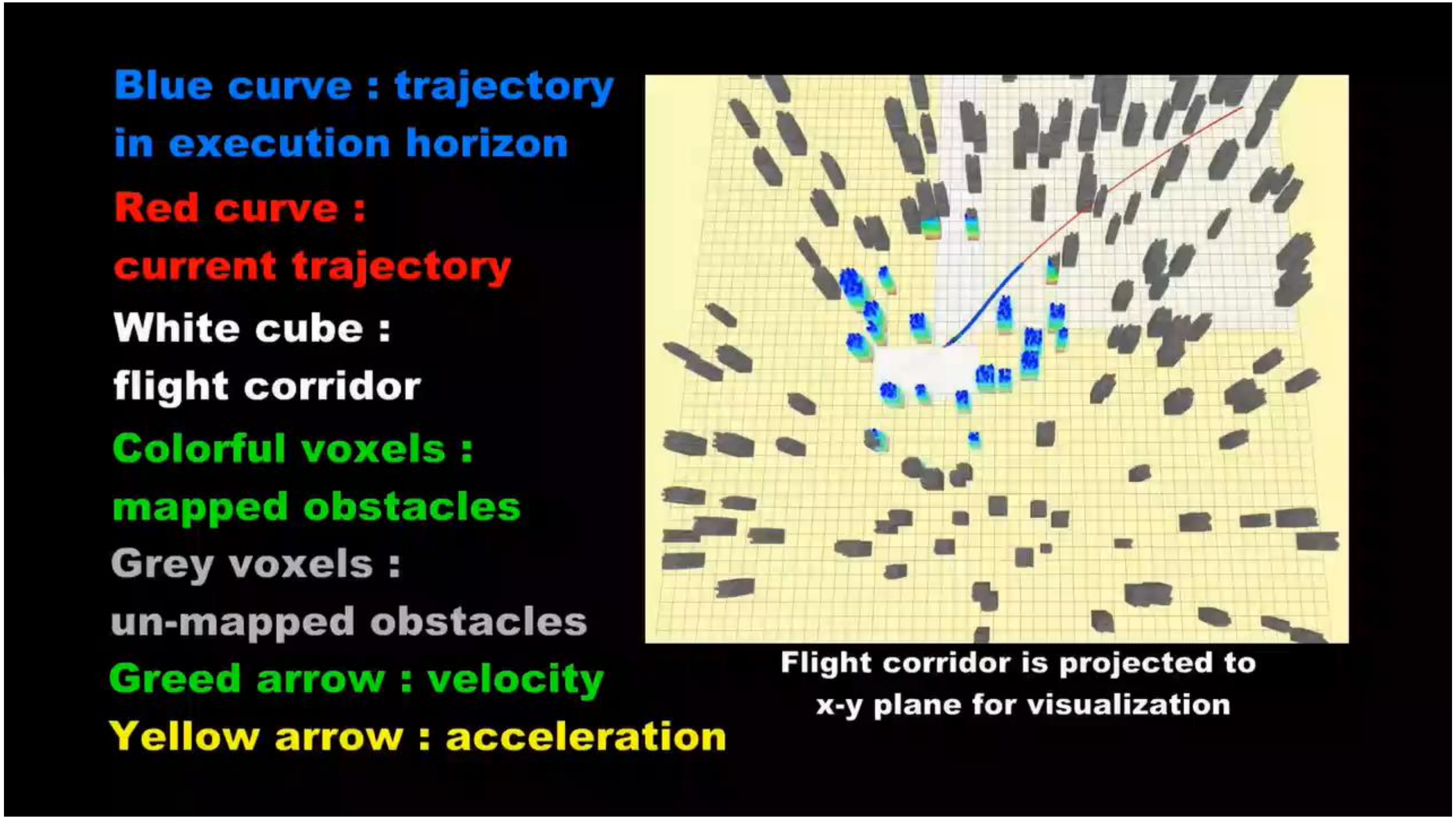
Stack all of these

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{Q}_o \mathbf{c} \\ \text{s.t.} \quad & \boxed{\mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq},} \\ & \boxed{\mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie},} \\ & \boxed{\mathbf{c}_j \in \Omega_j, \quad j = 1, 2, ..., m,} \end{aligned}$$

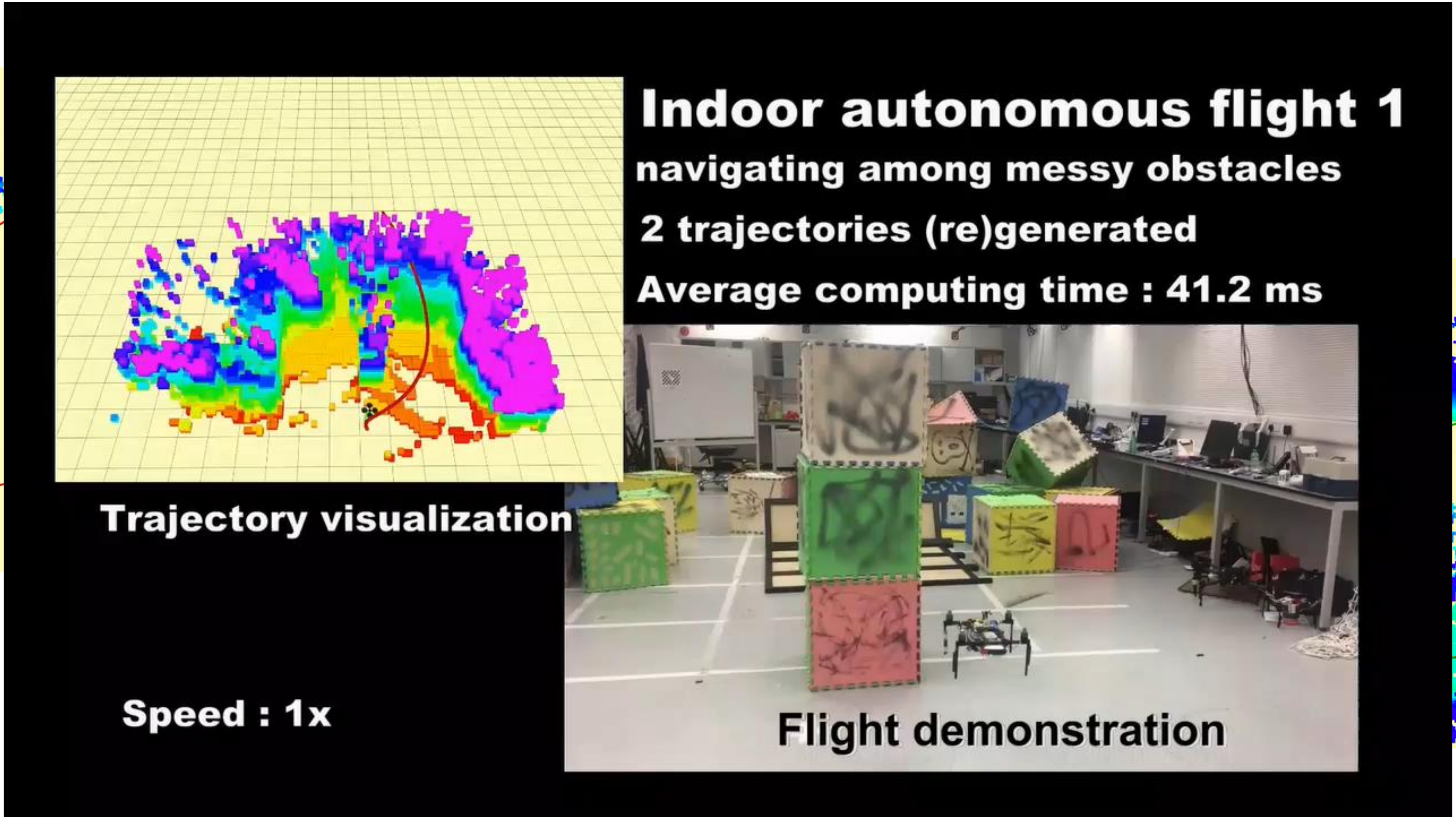
We only solve this program once to determine whether there is a qualified trajectory exists.

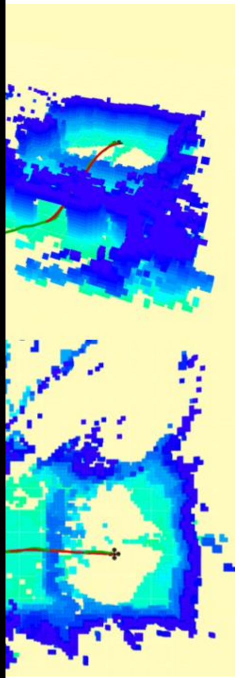
A typical convex QP formulation.

Simulation results



Experimental results



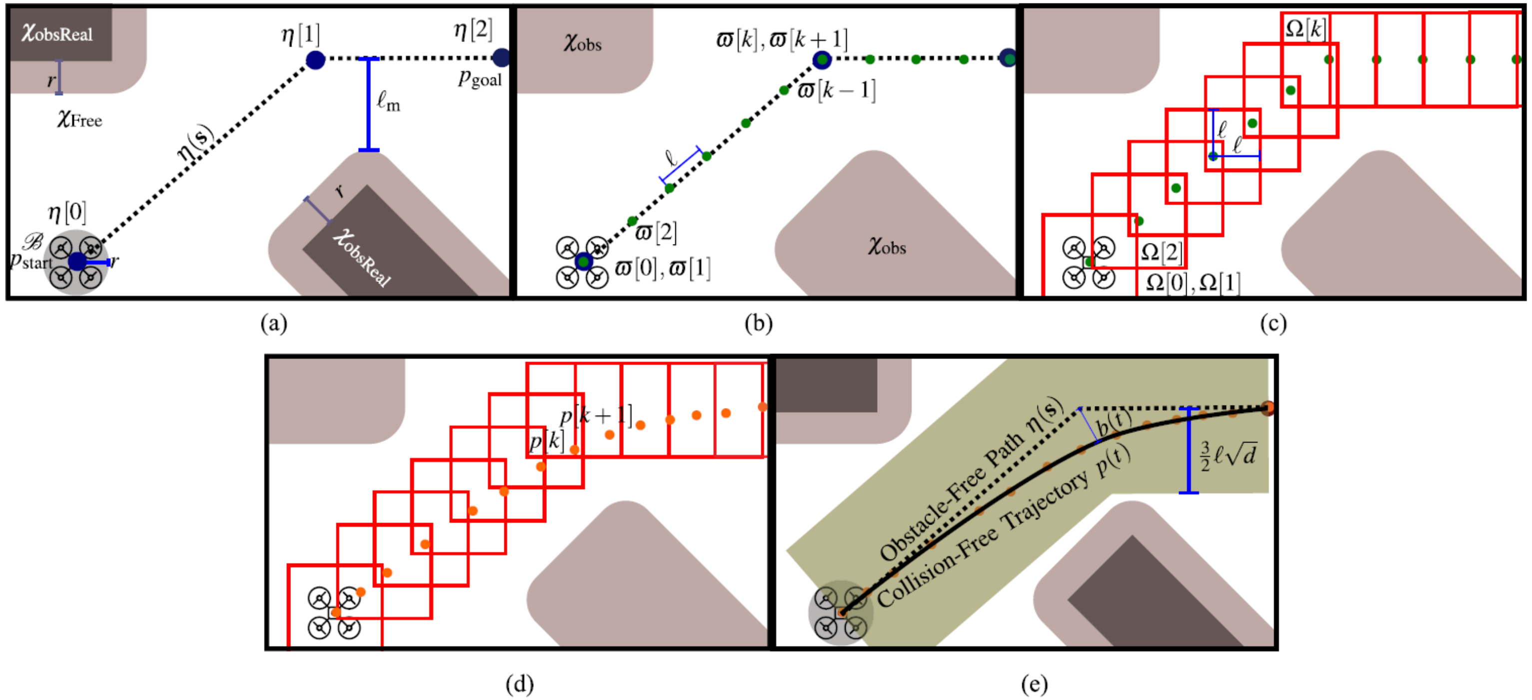


Online Safe Trajectory Generation For Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial, **Fei Gao** et al.
Source code released at: https://github.com/HKUST-Aerial-Robotics/Btraj ⬅ A complete UAV online motion planning framework

Other Options

Dense constraints

- Adding numerous constraints at discrete time ticks.
- Piecewise-constant accelerations at each tick.
- QP program solution.



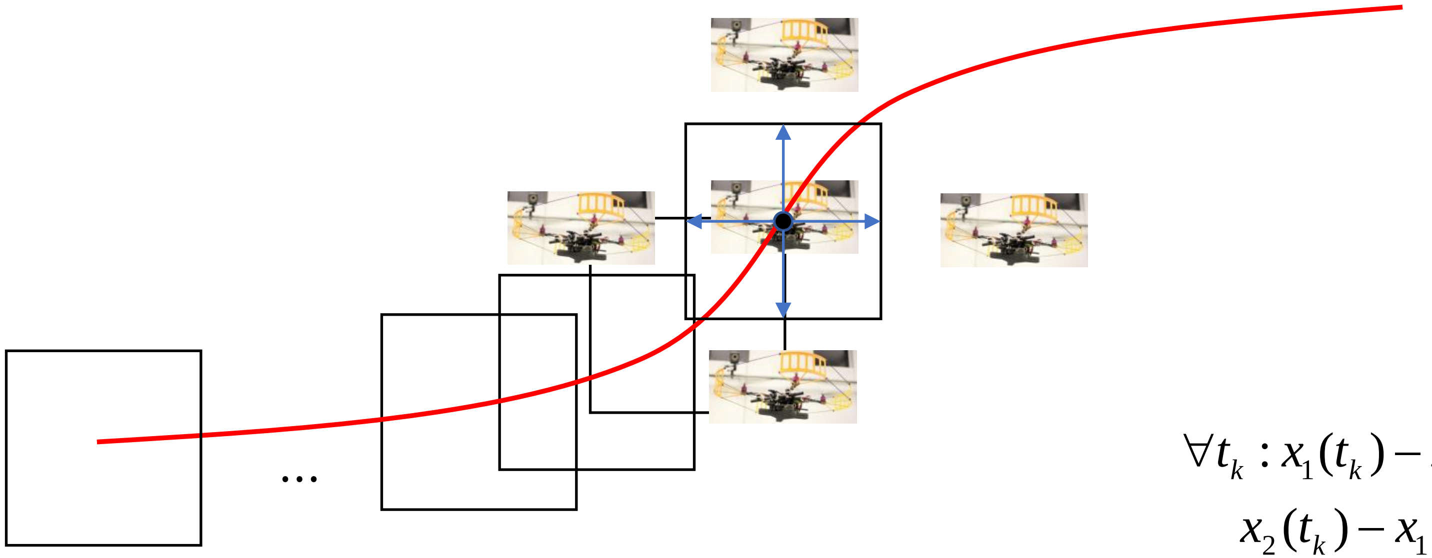
- Always generates over-conservative trajectories.
- Too many constraints, the computational burden is high.

A hybrid method for online trajectory planning of mobile robots in cluttered environments, **L Campos-Macías** et. al

Dense constraints

A hybrid method for online trajectory planning of mobile robots in cluttered environments, **L Campos-Macías** et. al

Mixed integer optimization



$$\forall t_k : x_1(t_k) - x_2(t_k) \le d_x$$
$$or \quad x_2(t_k) - x_1(t_k) \le d_x$$
$$or \quad y_1(t_k) - y_2(t_k) \le d_y$$
$$or \quad y_2(t_k) - y_1(t_k) \le d_y$$
$$or \quad z_1(t_k) - z_2(t_k) \le d_z$$
$$or \quad z_2(t_k) - z_1(t_k) \le d_z$$

- Mixed-integer QP.
- ‘Big M’ method.
- Super slow.

$$\forall t_k : x_1(t_k) - x_2(t_k) - c_0 \cdot M \le d_x$$
$$x_2(t_k) - x_1(t_k) - c_1 \cdot M \le d_x$$
$$y_1(t_k) - y_2(t_k) - c_2 \cdot M \le d_y$$
$$y_2(t_k) - y_1(t_k) - c_3 \cdot M \le d_y$$
$$z_1(t_k) - z_2(t_k) - c_4 \cdot M \le d_z$$
$$z_2(t_k) - z_1(t_k) - c_5 \cdot M \le d_z$$
$$\sum_{i=0}^{5} c_i = 5, \; M = 100000$$
$$c_i \in \{0,1\}$$

Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams, **D. Mellinger** et al.

Dense constraints

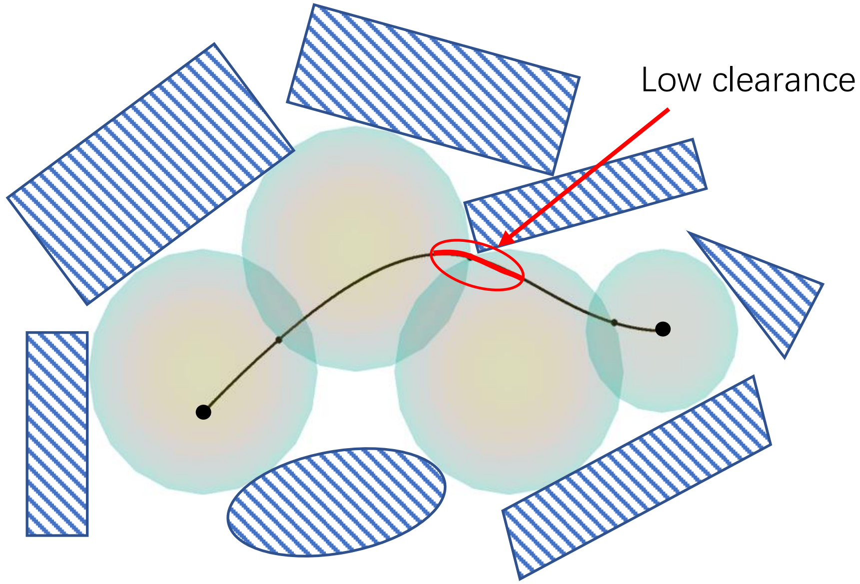
Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams, **D. Mellinger** et al.

Soft-constrained Optimization

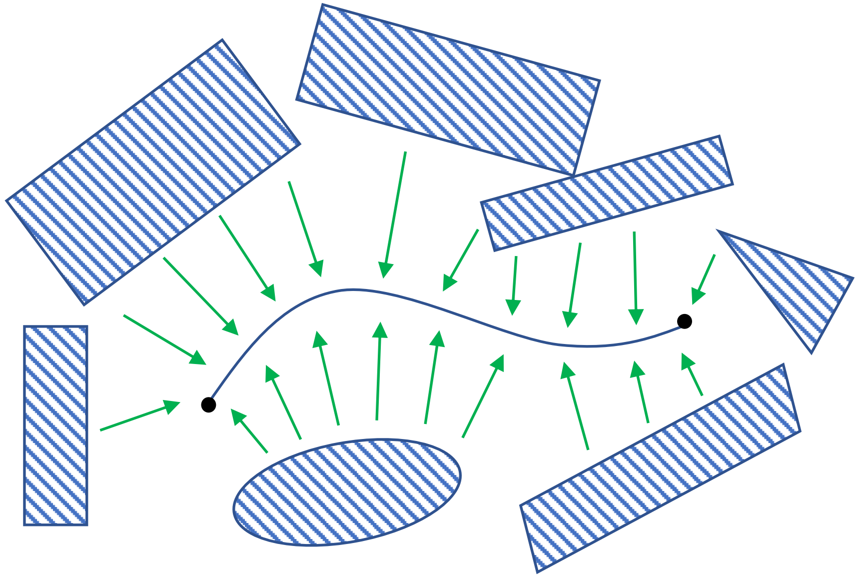
Distance-based Trajectory Optimization

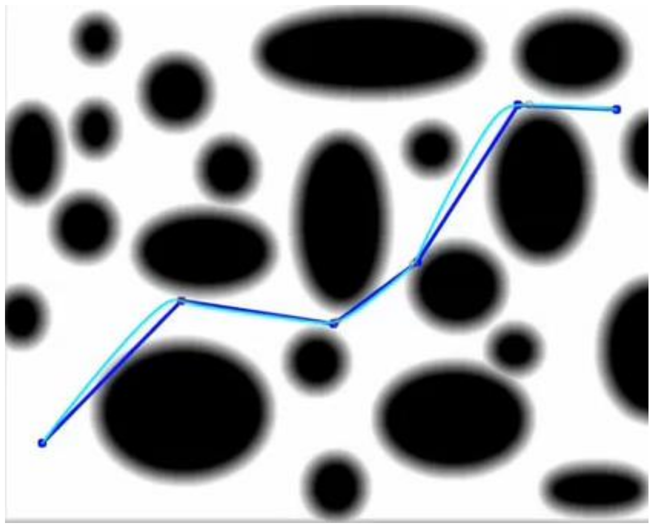
Motivation

Hard-constrained methods



Soft-constrained methods





Vison-based drone:

- Limited sensing range and quality
- Noisy depth estimation

Hard-constrained method:

- Treat all free space equally
- Solution space is sensitive to noise

Problem formulation

- Differential flatness property

$$\{x, y, z, \dot{x}, \dot{y}, \dot{z}, \emptyset, \theta, \varphi, p, q, r\} \rightarrow \{x, y, z, \varphi\}$$

- Piecewise polynomial trajectory

$$f_\mu(t) = \begin{cases} \sum_{j=0}^{N} p_{1j}(t - T_0)^j & T_0 \le t \le T_1 \\ \sum_{j=0}^{N} p_{2j}(t - T_1)^j & T_0 \le t \le T_1 \\ \vdots & \vdots \\ \sum_{j=0}^{N} p_{Mj}(t - T_{M-1})^j & T_0 \le t \le T_1 \end{cases}$$

- Objective function

$$J = J_s + J_c + J_d$$

$$= \underbrace{\lambda_1 J_1}_{\text{Smoothness cost}} + \underbrace{\lambda_2 J_2}_{\text{Collision cost}} + \underbrace{\lambda_3 J_3}_{\text{Dynamical cost}}$$

- Smoothness cost: minimum snap formulation

$$J_s = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k} \right)^2 dt$$

$$= \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^T \boldsymbol{C}^T \boldsymbol{M}^{-T} \boldsymbol{Q} \boldsymbol{M}^{-1} \boldsymbol{C} \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^T \begin{bmatrix} \boldsymbol{R}_{FF} & \boldsymbol{R}_{FP} \\ \boldsymbol{R}_{PF} & \boldsymbol{R}_{PP} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}$$

- Collision cost: penalize on the distance to nearest obstacle

$$J_c = \int_{T_0}^{T_M} c\big(p(t)\big) ds$$

$$= \sum_{k=0}^{T/\delta t} c\big(p(T_k)\big) \|v(t)\| \delta t, \; T_k = T_0 + k\delta t$$

- Dynamical Cost: penalize on the velocity and acceleration where exceeds limits (similar to collision term).

Objective/Jacobian evaluation

- Smoothness cost: minimum snap formulation

$$J_s = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k} \right)^2 dt$$

$$= \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^T \boldsymbol{C}^T \boldsymbol{M}^{-T} \boldsymbol{Q} \boldsymbol{M}^{-1} \boldsymbol{C} \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^T \begin{bmatrix} \boldsymbol{R}_{FF} & \boldsymbol{R}_{FP} \\ \boldsymbol{R}_{PF} & \boldsymbol{R}_{PP} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}$$

- The Jacobian with respect to free derivatives $\boldsymbol{d}_{p\mu}$ is:

$$\frac{\alpha J_s}{\alpha \boldsymbol{d}_{p\mu}} = 2\boldsymbol{d}_F^T \boldsymbol{R}_{FP} + 2\boldsymbol{d}_P^T \boldsymbol{R}_{PP}$$

- Collision cost: penalize on the distance to nearest obstacle

$$J_c = \int_{T_0}^{T_M} \boxed{c(p(t))} ds$$

Distance penalty at a point along the trajectory

$$= \sum_{k=0}^{T/\delta t} c(p(T_k)) \|v(t)\| \delta t, \; T_k = T_0 + k\delta t$$

- The Jacobian with respect to free derivatives $\boldsymbol{d}_{p\mu}$ is:

$$\frac{\alpha J_c}{\alpha \boldsymbol{d}_{p\mu}} = \sum_{k=0}^{T/\delta t} \left\{ \nabla_\mu c(p(T_k)) \|v\| \boldsymbol{F} + c(p(T_k)) \frac{v_\mu}{\|v\|} \boldsymbol{G} \right\} \delta t \, , \mu \in \{x, y, z\}$$

L_{dp} is the right block of matrix $M^{-1}C$ which corresponds to the free derivatives on the μ axis $d_{p\mu}$.

$$F = TL_{dp}, \qquad G = TV_m L_{dp}.$$

$\nabla_\mu c(\cdot)$ is the gradient in μ axis of the collision cost.

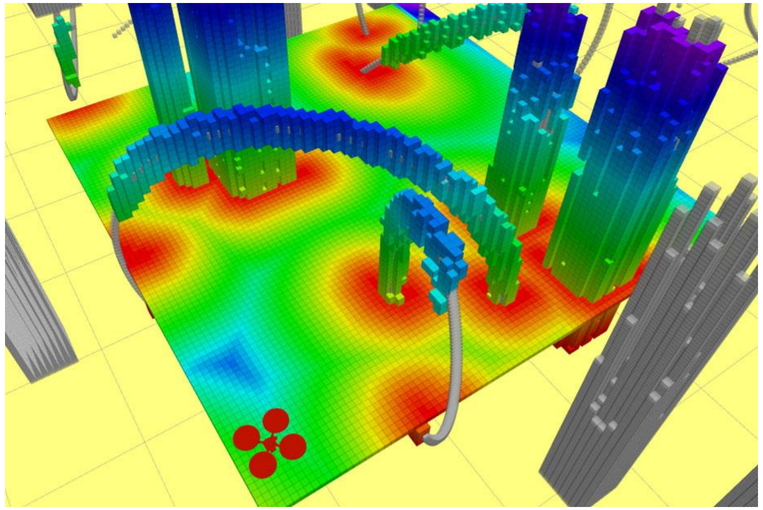
V_m maps the coefficients of the position to the coefficients of the velocity. $\boldsymbol{T} = [T_k^0, T_k^1, \cdots, T_k^n]$

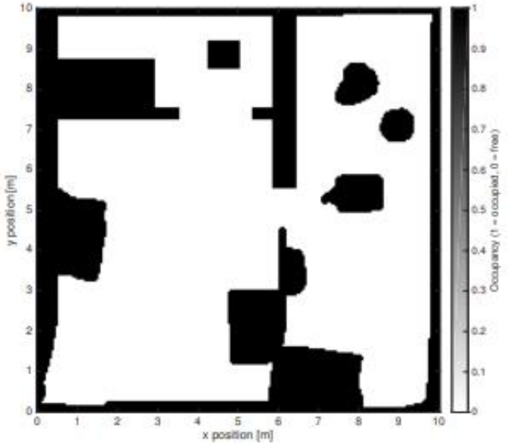
$$\mathbf{H}_o = \left[\frac{\partial^2 f_o}{\partial \mathbf{d}_{P_x}^2}, \frac{\partial^2 f_o}{\partial \mathbf{d}_{P_y}^2}, \frac{\partial^2 f_o}{\partial \mathbf{d}_{P_z}^2} \right],$$

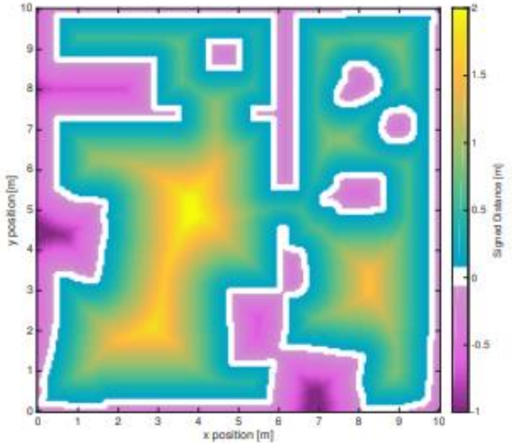
$$\frac{\partial^2 f_o}{\partial \mathbf{d}_{P_\mu}^2} = \sum_{k=0}^{\tau/\delta t} \Big\{ \mathbf{F}^T \nabla_\mu c(p(\mathcal{T}_k)) \frac{v_\mu}{\|v\|} \mathbf{G} + \mathbf{F}^T \nabla_\mu^2 c(p(\mathcal{T}_k)) \|v\| \mathbf{F}$$

$$+ \mathbf{G}^T \nabla_\mu c(p(\mathcal{T}_k)) \frac{v_\mu}{\|v\|} \mathbf{F} + \mathbf{G}^T c(p(\mathcal{T}_k)) \frac{v_\mu^2}{\|v\|^3} \mathbf{G} \Big\} \delta t, \qquad (10)$$

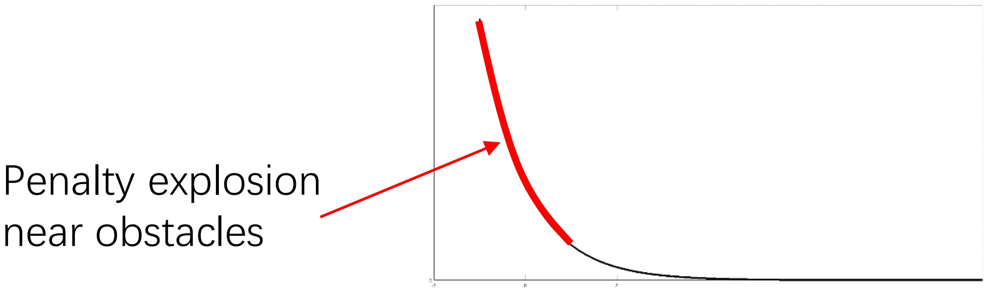
Euclidean signed distance field (ESDF)



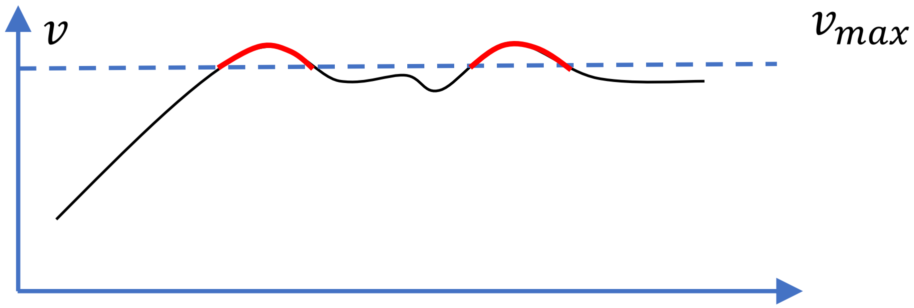




Use **exponential** function as cost function c, to prevent trajectory from be near to obstacle



- Dynamical Cost



Numerical optimization

$$\text{minimize } f(x)$$

- Produce sequence of points $x^{(k)} \in dom\, f, k = 0,1, \ldots$with

$$f\left(x^{(k)}\right) \rightarrow p^{\star}$$

- Can be interpreted as iterative methods for solving optimality condition

$$\nabla f(x^{\star}) = 0$$

Descent method

$$x^{k+1} = x^k + t^k \Delta x^k \text{ with } f\left(x^{k+1}\right) < f(x^k)$$

- Other notations: $x^+ = x + t\Delta x,\ x \coloneqq x + t\Delta x$
- Δx is the step, or search direction; t is the step size, or step length

---

General descent method

given a starting point $x \in dom\ f$.

repeat

1. Determine a descent direction Δx.
2. Line search. Choose a step size $t > 0$.
3. Update. $x \coloneqq x + t\Delta x$.

until stopping criterion is satisfied.

Line search types

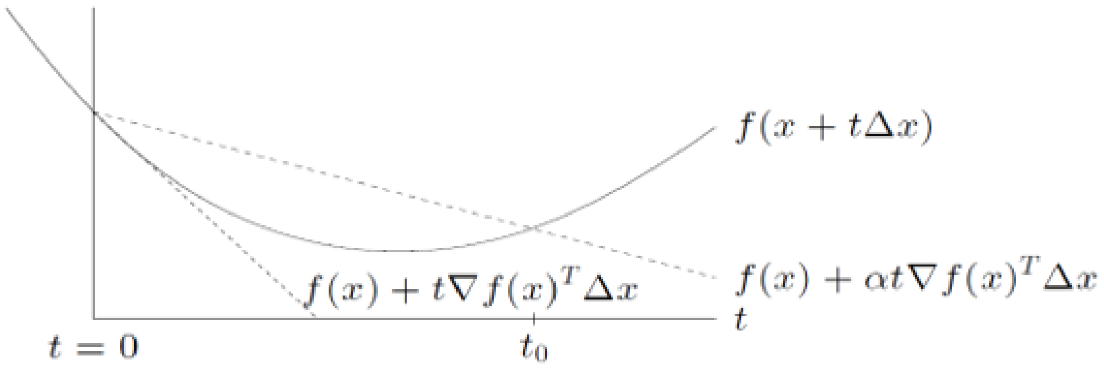
exact line search: $t = argmin_{t>0} f(x + t\Delta x)$

backtracking line search (with parameters $\alpha \in (0, 1/2), \beta \in (0,1)$)

- starting at $t = 1$, repeat $t := \beta t$ until

$$f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^T \Delta x$$

- graphical interpretation: backtrack until $t \leq t_0$



First-order method

Gradient descent method

General descent method with $\Delta x = -\nabla f(x)$

---

given a starting point $x \in dom\ f$.

Repeat

1. $\Delta x \coloneqq -\nabla f(x)$.
2. Line search. Choose step size t via exact or backtracking line search.
3. Update. $x \coloneqq x + t\Delta x$.

until stopping criterion is satisfied.

- Stopping criterion usually of the form $||\nabla f(x)||_2 \leq \epsilon$

Second-order method

Newton method

$Taylor\ expansion\ at\ x^{(k)}$, and second − order similarity $\Delta x = -\nabla^2 f(x)^{-1} \nabla f(x)$

---

given a starting point $x \in dom\ f$.

Repeat

1. $\Delta x := -\nabla^2 f(x)^{-1} \nabla f(x)$.
2. Line search. $t = argmin_{t>0} f(x^{(k)} - t\nabla^2 f(x)^{-1} \nabla f(x))$..
3. Update. $x := x + t\Delta x$.

until stopping criterion is satisfied.

- Stopping criterion usually of the form $\left||\nabla f(x)|\right|_2 \leq \epsilon$
- $\nabla^2 f(x)$ is the Hessian matrix of the $f(x)$ at x
- For Gauss-newton: $\nabla^2 f(x) \approx J_f^T J_f$, J_f is Jacobi matrix, $\Delta x = -(J_f^T J_f)^{-1}\ J_f^T$

Second-order method

Levenberg-Marquardt method

Improvement of Gauss-newton method: $\Delta x = -(J_f^T J_f + \lambda I)^{-1} J_f^T$

---

given a starting point $x \in dom\, f, start\ \lambda_0 > 0$.

Repeat

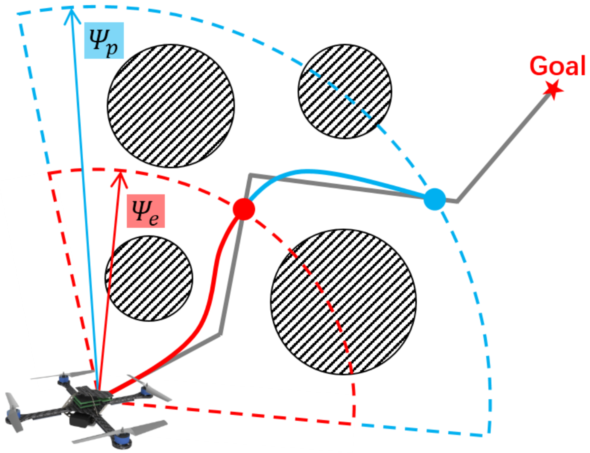
1. $\Delta x = -(J_f^T J_f + \lambda I)^{-1} J_f^T$.
2. Update. λ , the updating is controlled by the gain ratio
3. Update. $x := x + \Delta x$.

until stopping criterion is satisfied.

- Stopping criterion usually of the form $||\nabla f(x)||_2 \leq \epsilon$
- Whenλ->0, LM method -> Gauss-newton method
- Whenλ->∞, LM method -> Gradient descent method

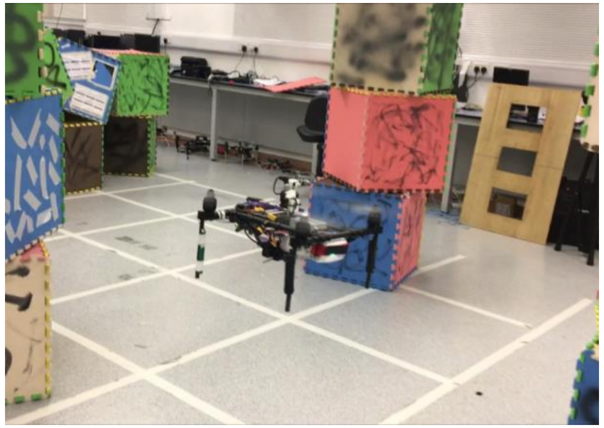
Planning strategy

- **Receding horizon re-planning** ψ_p : *planning horizon* ψ_e : *execution horizon*



- Get a global plan, search at the beginning of the navigation, or initialize as a straight line.
- Find a local path a local target by using the front-end.
- Generate a local trajectory by using the back-end.
- Track the trajectory within the execution horizon

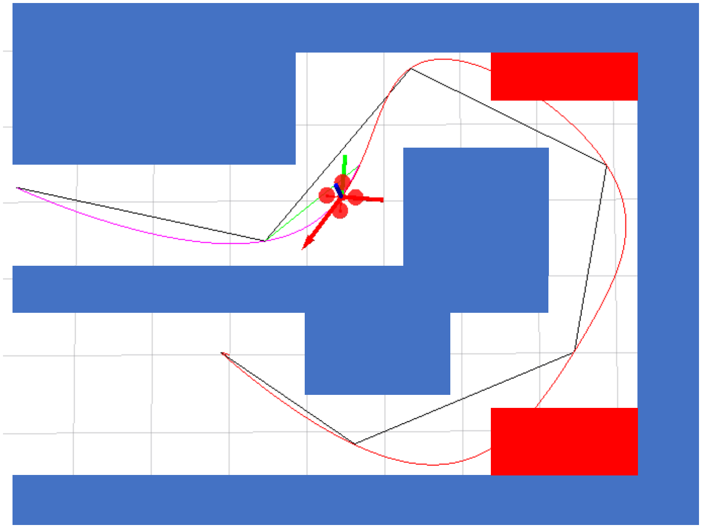
- **Exploration strategy**



- Activated when outlier in mapping module blocks the way.
- Generate safe but short trajectories in nearby regions.
- Hover and observe.

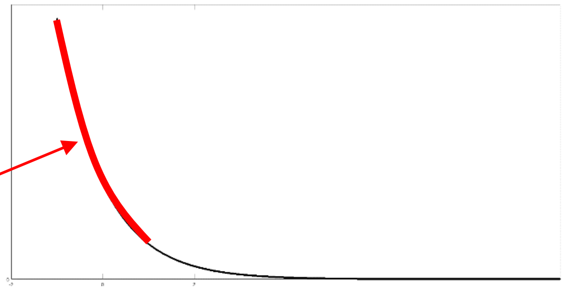
Planning strategy

- Initialize as minimum snap trajectory
  - Good smoothness.
  - Collision may cost by overshoot. Unsafe initial value.



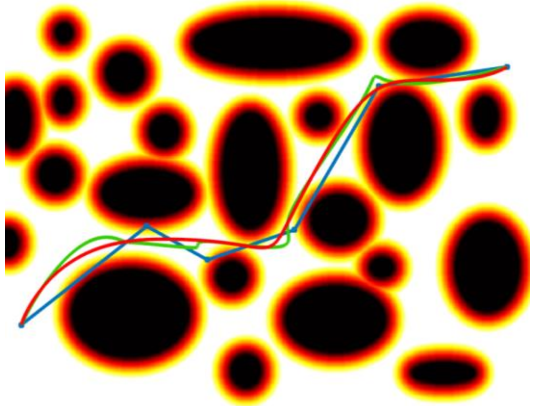
- Initialize as straight-line trajectory following the path
  - Poor smoothness.
  - Collision free. Safe initial value.
  - Given collision-free initial trajectory, safety is achieved by using a **exponential penalty function** on collision cost.

Penalty explosion near obstacles

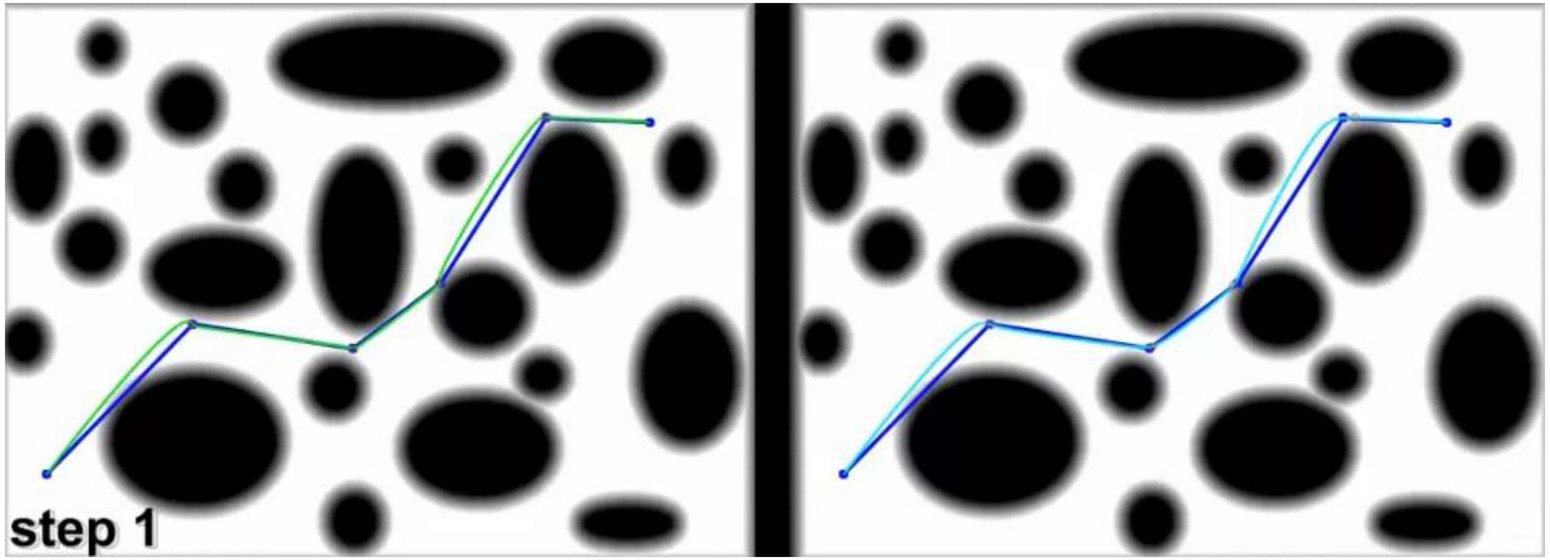


Optimization strategy

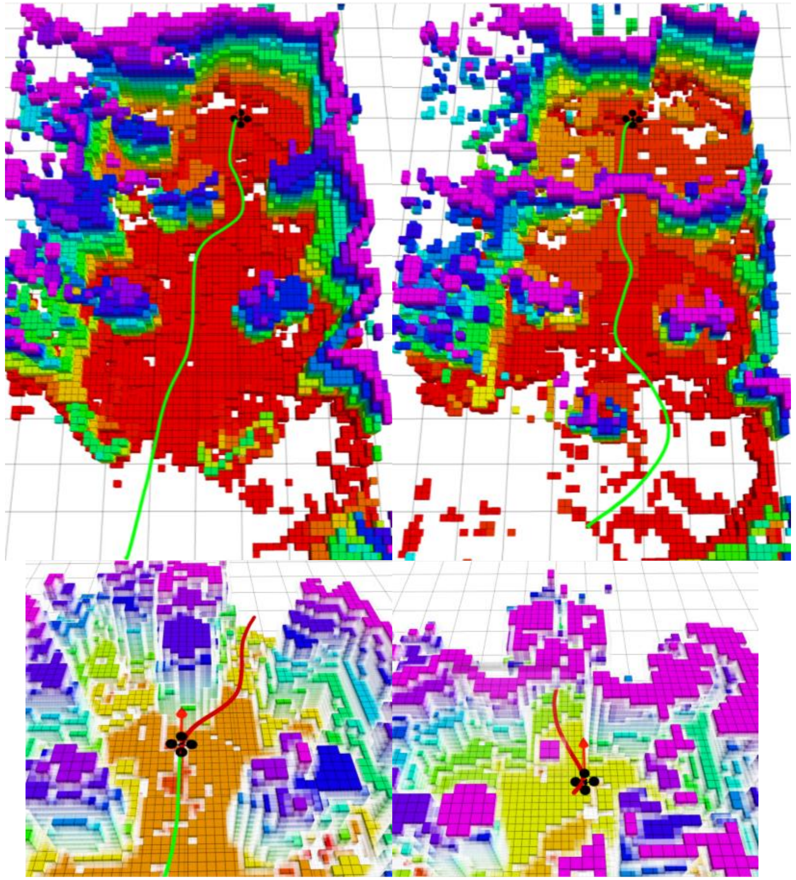
- two-step optimization

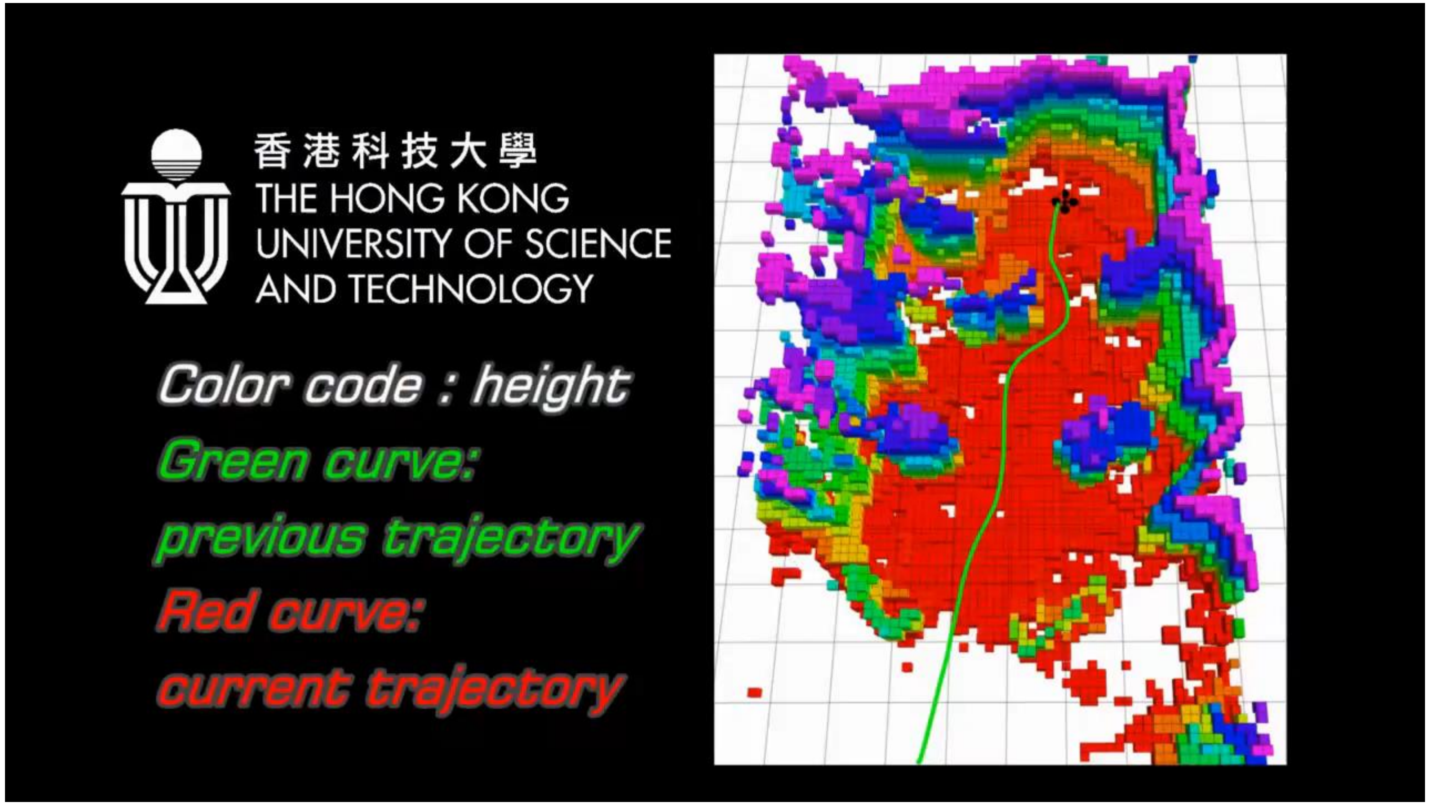


- Optimize the trajectory with collision cost only.
- Re-allocate time and re-parametrize the trajectory.
- Optimize the objective with a smoothness term and dynamical penalty term added.

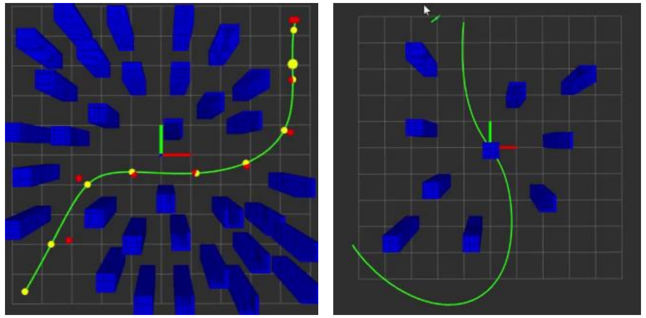


Results





Source code released at:
https://github.com/HKUST-Aerial-Robotics/grad_traj_optimization
A tool for local UAV trajectory optimization



Gradient-Based Online Safe Trajectory Generation for Quadrotor Flight in Complex Environments, **Fei Gao** et al.

Case Study

Fast Planner

Framework

Kinodynamic path searching

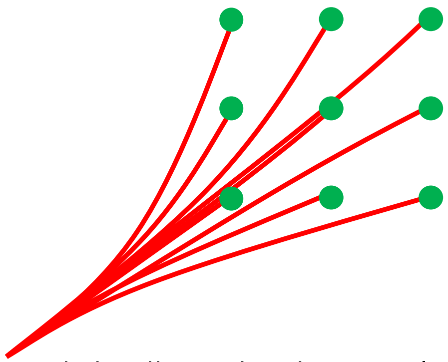
+ B-Spline trajectory optimization

+ time adjustment

B-Spline trajectory optimization

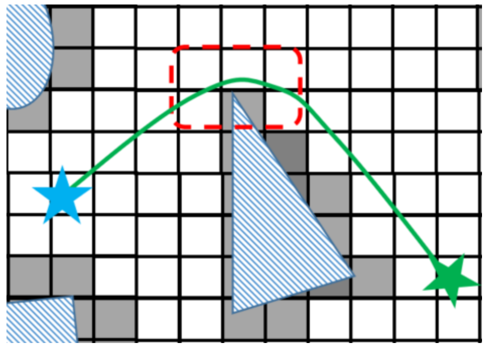
- Limitations of initial path

a) Suboptimality



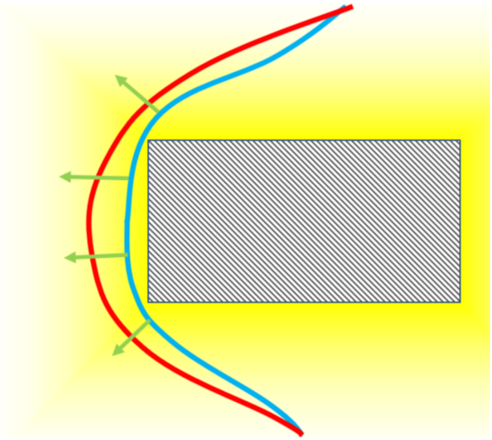
only search in discretized control space

b) Low clearance / high collision risk



some segments may be very close to obstacles

- Gradient-based trajectory optimization
  - Improve smoothness
  - Use Euclidean distance field (EDF) to improve clearance



B-Spline trajectory optimization

- Drawbacks of previous gradient-based methods：
  - Expensive computation of collision cost

$$f_{collision} = \int_{T_1}^{T_2} c\big(p(t)\big)ds = \int_{T_1}^{T_2} c\big(p(t)\big)||v(t)||dt = \boxed{\sum_{k=0}^{(T_2-T_1)/\delta t}} c\big(p(T_k)\big)||v(t)||\delta t$$

small δt for accuracy $\rightarrow$ large $(T_2-T_1)/\delta t$ $\rightarrow$ high complexity

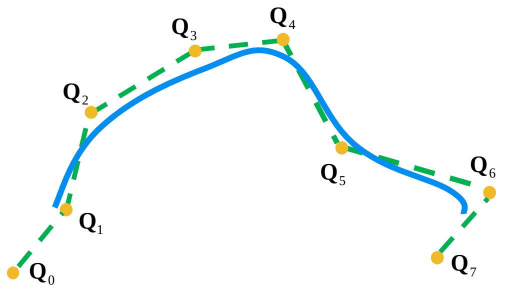
  - Also so for dynamic feasibility costs

$$f_{vel} = \sum_{\mu\in\{x,y,z\}} \int_{T_1}^{T_2} c_v\left(v_\mu(t)\right) ds = \sum_{\mu\in\{x,y,z\}} c_v\left(v_\mu(t)\right) \|a_\mu(t)\|\, dt$$

$$f_{acc} = \sum_{\mu\in\{x,y,z\}} \int_{T_1}^{T_2} c_a\left(a_\mu(t)\right) ds = \sum_{\mu\in\{x,y,z\}} c_a\left(a_\mu(t)\right) \|j_\mu(t)\|\, dt$$

B-Spline trajectory optimization

- Solution: B-Spline trajectory representation
  - Uniform B-Spline



Determined by its degree p_b, time interval Δt, and a set of $\{Q_0, Q_1, \ldots, Q_N\}$.

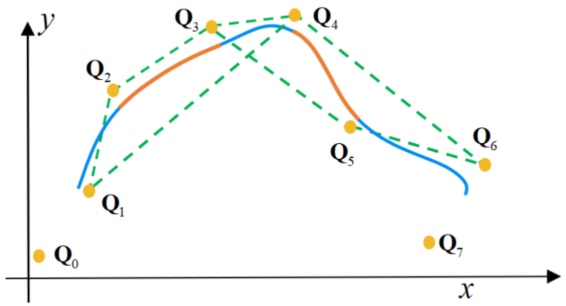
$$t \in [t_i, t_{i+1}) \xrightarrow{\text{normalize}} s(t) = (t - t_i)/\Delta t$$

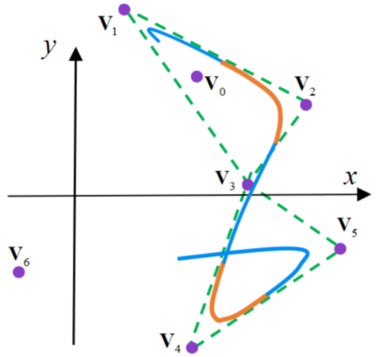
$$p(s(t)) = \begin{pmatrix} 1 \\ s \\ s^2 \\ \cdot \\ \cdot \\ \cdot \\ s^{p_b} \end{pmatrix}^T M_{p_b+1} \begin{pmatrix} Q_{i-p_b} \\ Q_{i-p_b+1} \\ Q_{i-p_b+2} \\ \cdot \\ \cdot \\ \cdot \\ Q_i \end{pmatrix}$$

  - Advantages

a) Convex hull property: simplify safety & dynamic constraints
b) Continuity: no need of constraints at segments joints

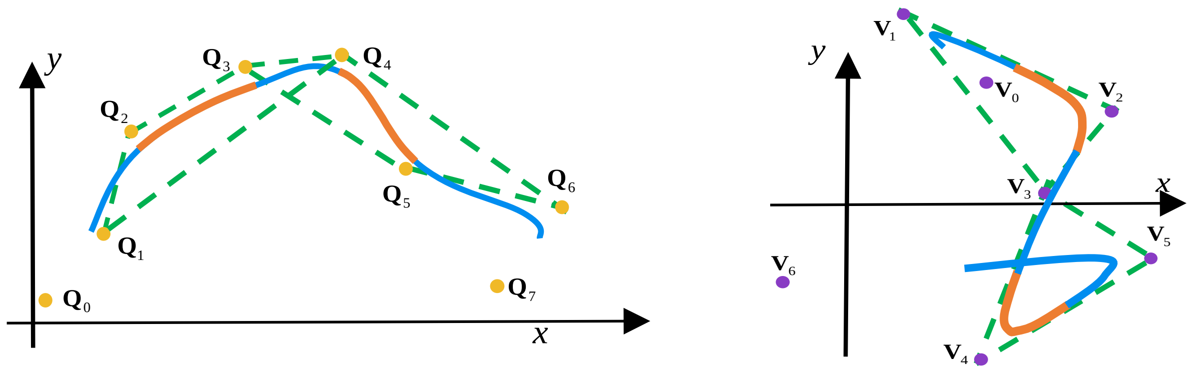
$\Rightarrow$ Speed up optimization





B-Spline trajectory optimization

- Convex hull property



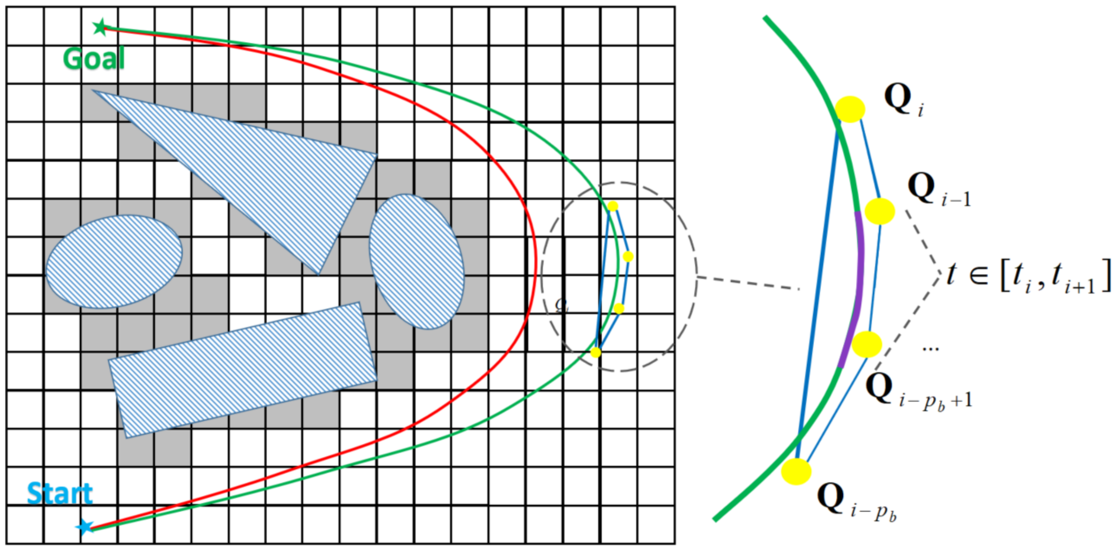
Each segment of a B-spline is **bounded** by the corresponding convex hull of control points (left), and so is the derivative (right).

- Simplify safety & dynamic constraints

Convex hulls are within feasible space $\xrightarrow{\text{convex hull property}}$ The whole trajectory is feasible!

B-Spline trajectory optimization

- Ensure safety by convex hull



Push control points away from obstacles → Convex hulls are pushed to safe space → The whole trajectory is bounded in safe region

B-Spline trajectory optimization

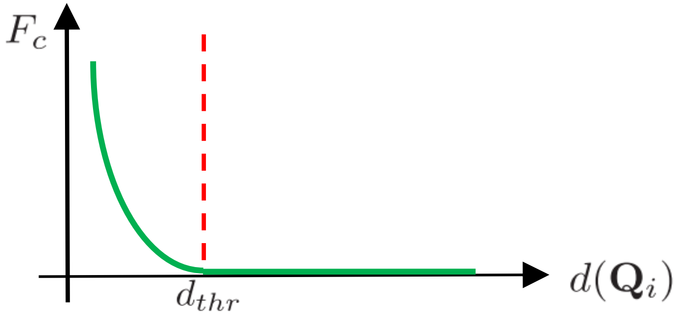
- Problem formulation $\quad f_{total} = \lambda_1 f_s + \lambda_2 f_c + \lambda_3 (f_v + f_a)$
  - Smoothness - elastic band cost function

$$f_s = \sum_{i=p_b-1}^{N-p_b+1} \left\| \underbrace{(\mathbf{Q}_{i+1} - \mathbf{Q}_i)}_{F_{i+1,i}} - \underbrace{(\mathbf{Q}_{i-1} - \mathbf{Q}_i)}_{F_{i-1,i}} \right\|^2$$

  - Safety - potential function in EDF

$$f_c = \sum_{i=p_b}^{N-p_b} F_c(d(\mathbf{Q}_i))$$

$$F_c(d(\mathbf{Q}_i)) = \begin{cases} (d(\mathbf{Q}_i) - d_{thr})^2 & d(\mathbf{Q}_i) \le d_{thr} \\ 0 & d(\mathbf{Q}_i) > d_{thr} \end{cases}$$



  - Dynamic feasibility

$$f_v = \sum_{\mu \in \{x,y,z\}} \sum_{i=p_b}^{N-p_b} F_v(V_{i\mu}), \quad f_a = \sum_{\mu \in \{x,y,z\}} \sum_{i=p_b-2}^{N-p_b} F_a(A_{i\mu})$$

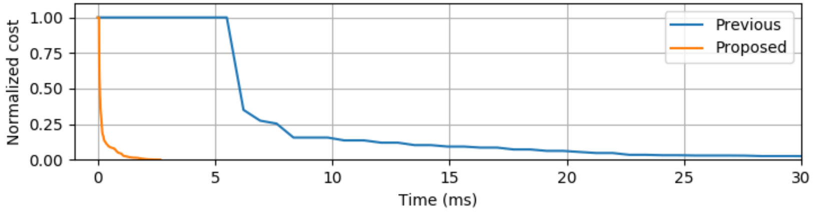
$$F_v(v_\mu) = \begin{cases} \left(v_\mu^2 - v_{\max}^2\right)^2 & v_\mu^2 > v_{\max}^2 \\ 0 & v_\mu^2 \le v_{\max}^2 \end{cases}$$

Solve this non-linear optimization by the L-BFGS algorithm

Results

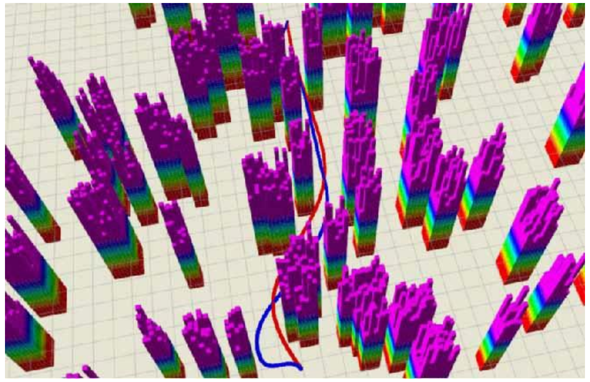
- Comparisons

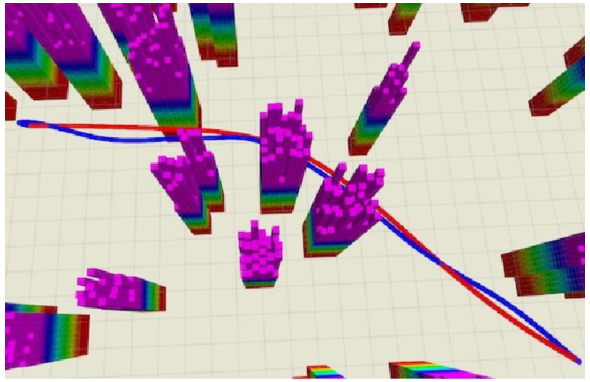
a) Faster convergence



b) Higher trajectory quality

	Integral of Jerk^2 (m^2/s^5)			Comp. Time(s)
	Mean	Max	Std	
Previous [1]	43.913	181.495	18.394	0.010
Proposed	**35.932**	**131.913**	**13.118**	0.001

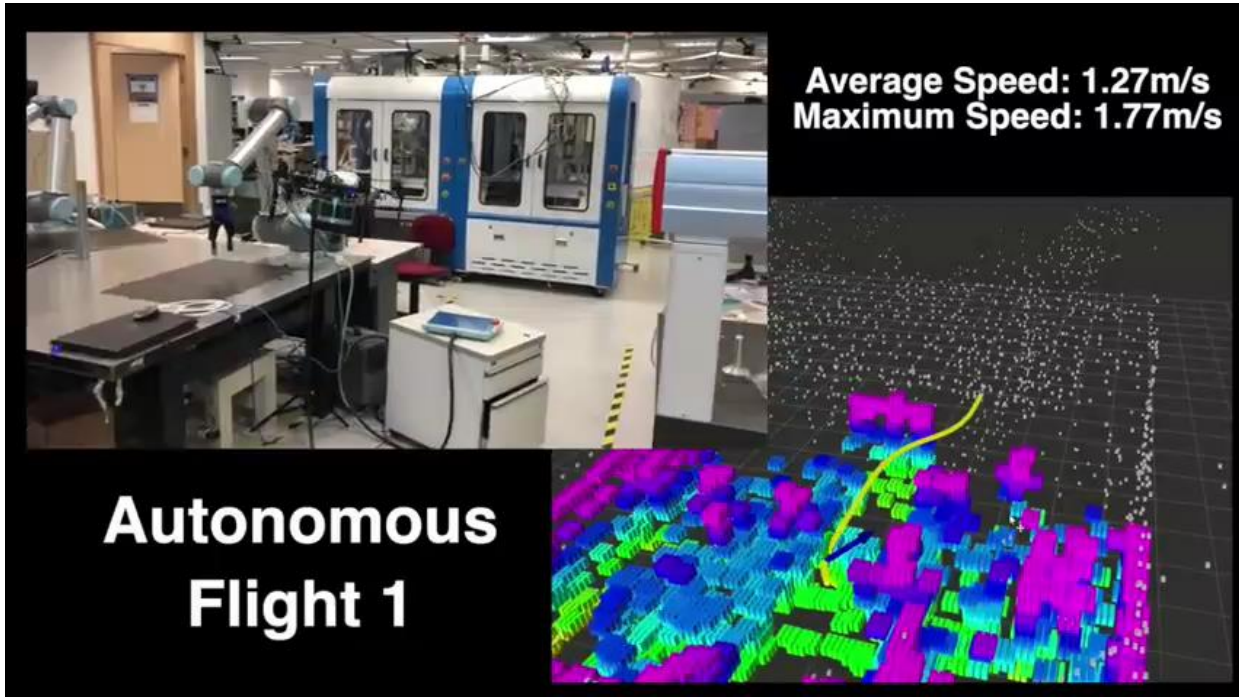


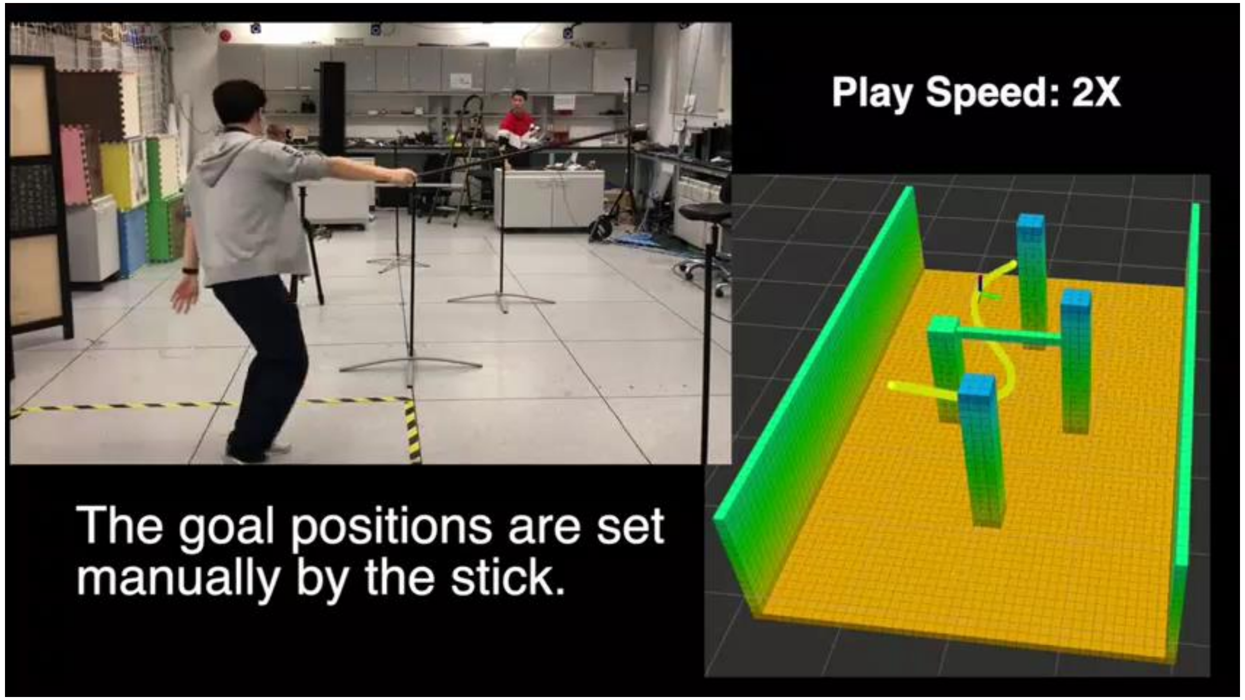


Trajectories generated by the proposed (red) and previous [1] (blue) method. Computation time for proposed method is 10 times shorter!

Gradient-Based Online Safe Trajectory Generation for Quadrotor Flight in Complex Environments, **Fei Gao** et al.

Results


Average Speed: 1.27m/s
Maximum Speed: 1.77m/s
Autonomous
Flight 1

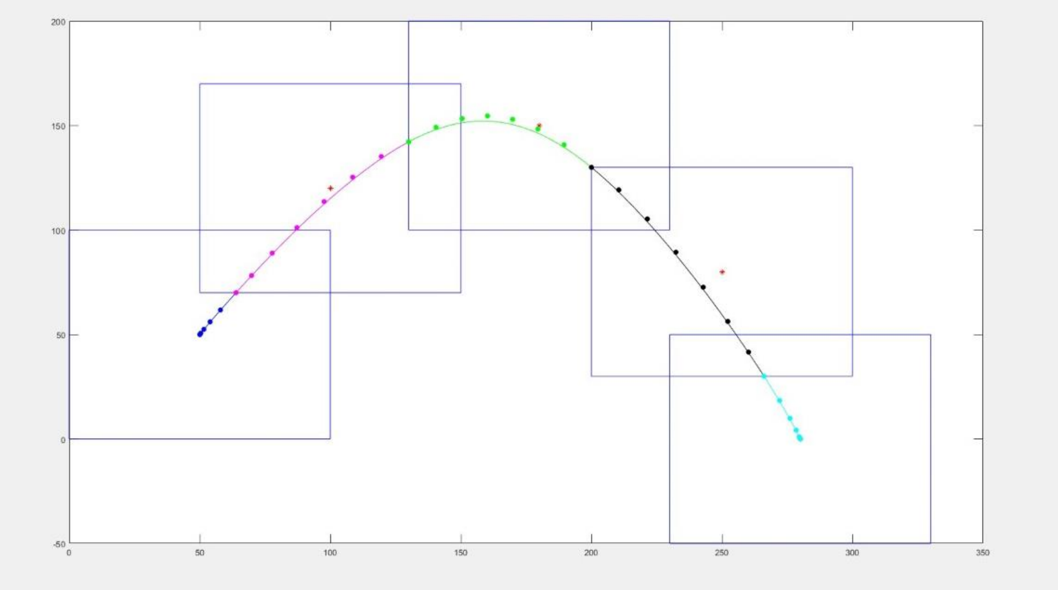

Play Speed: 2X
The goal positions are set manually by the stick.

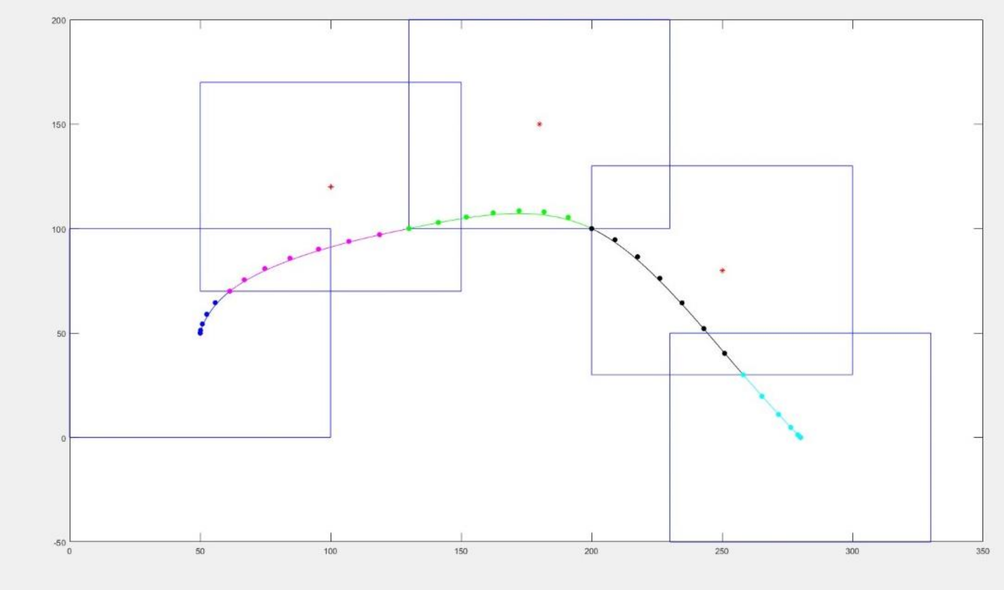
Homework

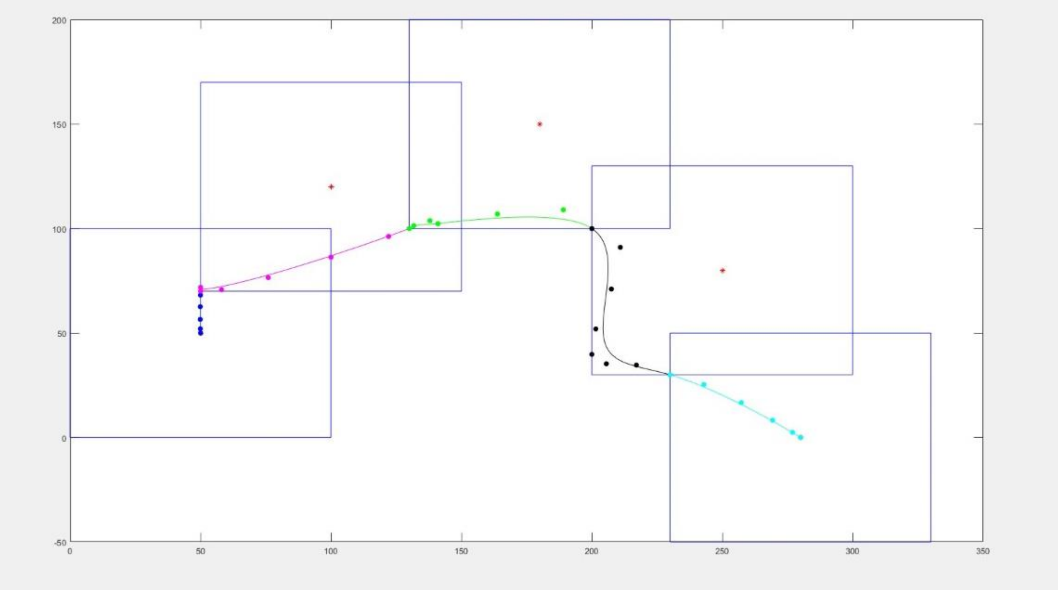
Homework

- In matlab, write a corridor-constrained piecewise Bezier curve generation.
- The conversion between Bezier to monomial polynomial is given.
- The corridor is pre-defined.
- Only position needs to be constrained.
- TA provides a video tutorial.

Homework


200
150
100
50
0
-50
0
50
100
150
200
250
300
350


200
150
100
50
0
-50
0
50
100
150
200
250
300
350


200
150
100
50
0
-50
0
50
100
150
200
250
300
350

Thanks for Listening!