

## Lecture 5

# MINIMUM SNAP TRAJECTORY GENERATION



主讲人 Fei Gao

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





# Outline



1. Introduction



2. Minimum Snap Optimization



3. Closed-form Solution to Minimum Snap



4. Implementation Details



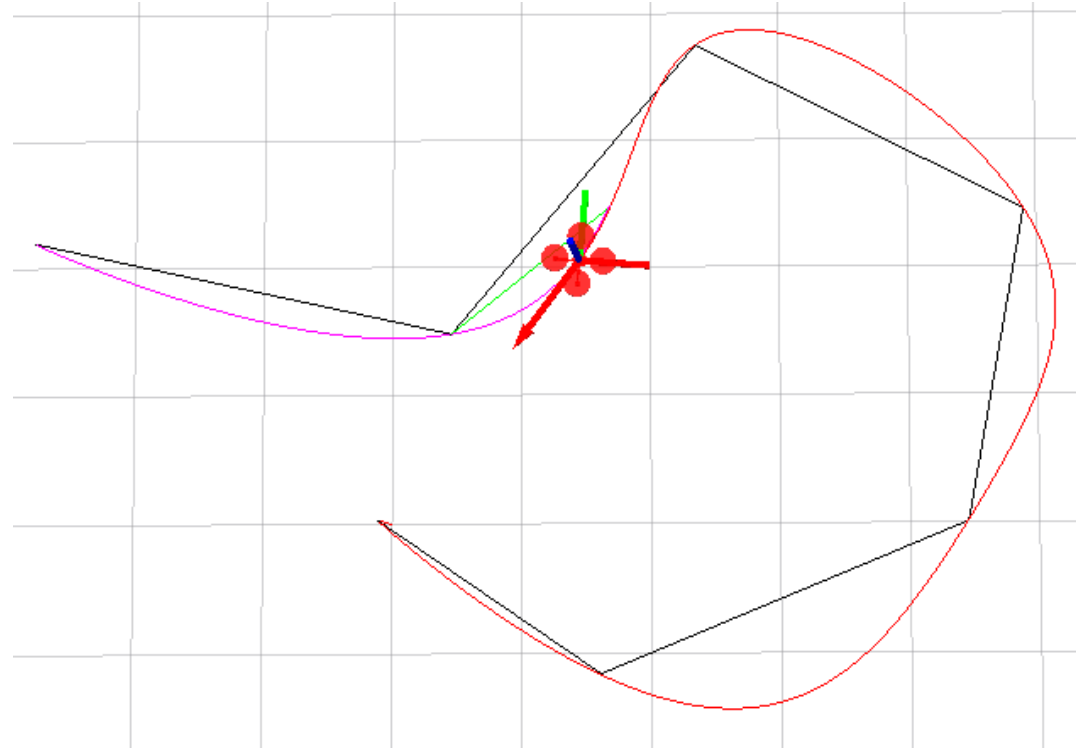
5. Homework

# Introduction



# Why smooth trajectory

- Good for autonomous moving.
- Velocity/higher order dynamics can't change immediately.
- The robot should not stop at turns.
- Save energy.

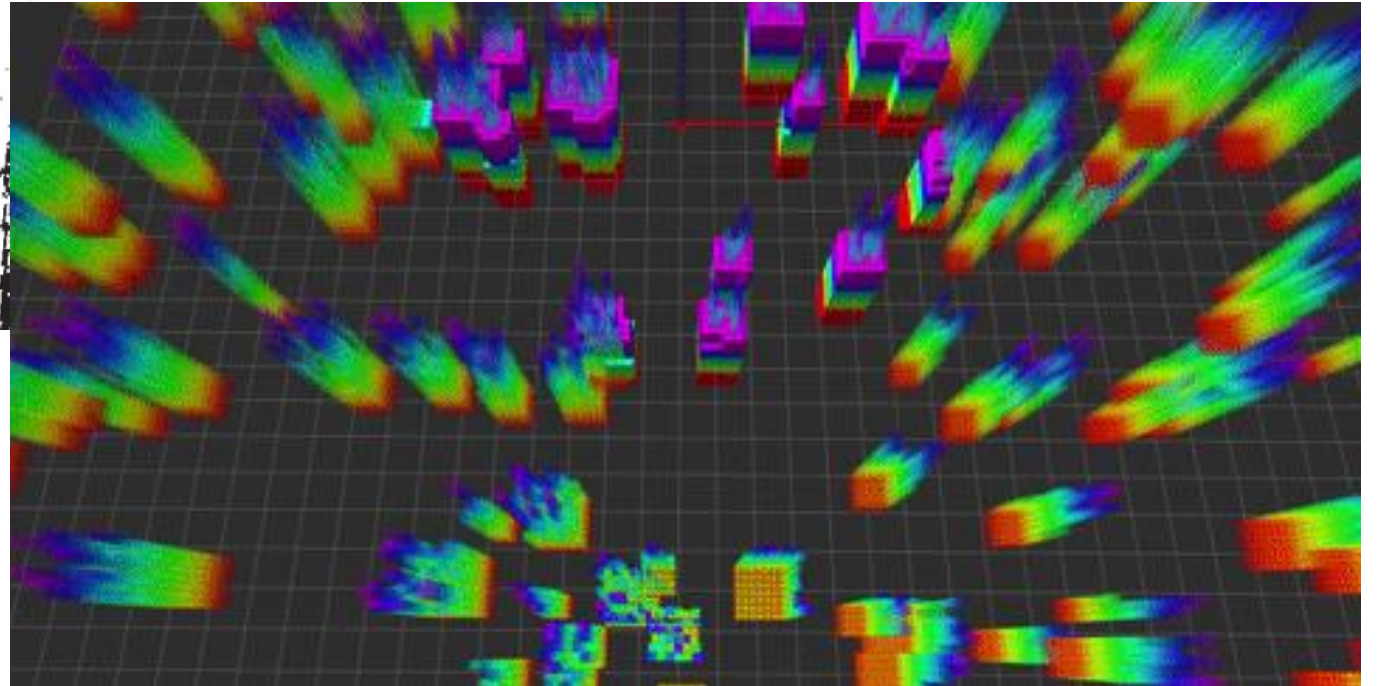
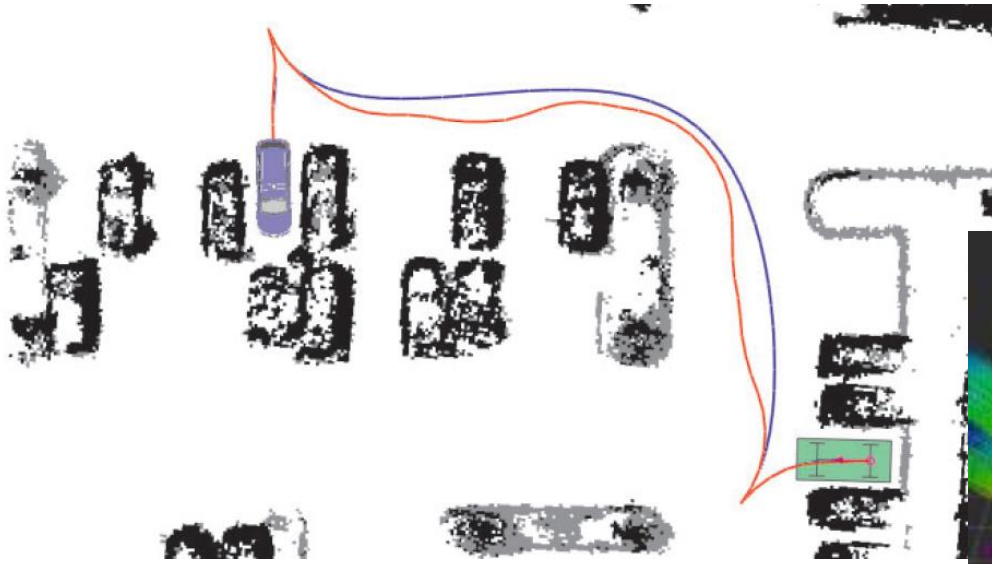




# Why trajectory generation/optimization

**Ask:** We have the front-end (path finding?), why a back-end necessary?

**Ask:** The front-end is kinodynamic feasible, why a back-end necessary?

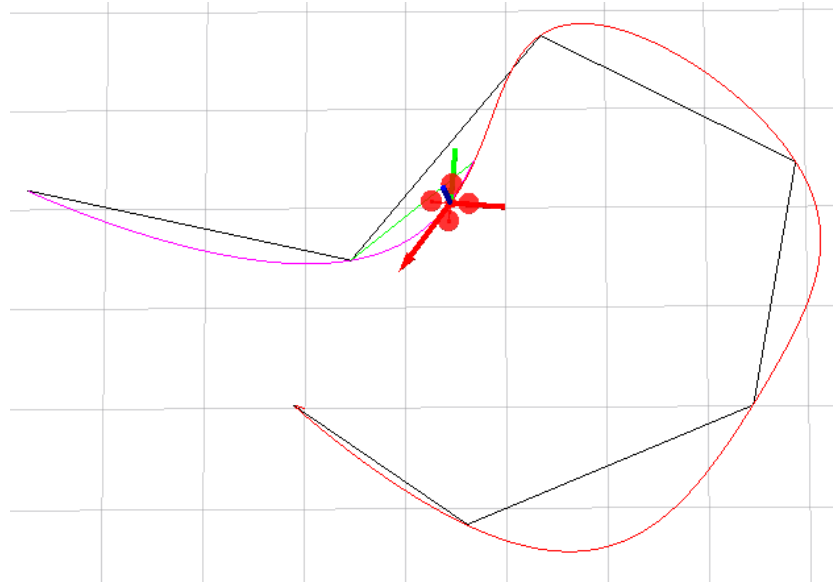


- Path quality.
- Time efficiency.



# Smooth trajectory generation

- Boundary condition: start, goal positions (orientations)
- Intermediate condition: waypoint positions (orientations)
  - Waypoints can be found by path planning ( $A^*$ ,  $RRT^*$ , etc.)
  - Introduced in previous 3 lectures
- Smoothness criteria
  - Generally translates into minimizing rate of change of "input"



# Minimum Snap Optimization

# Differential Flatness





# Differential Flatness

The states and the inputs of a quadrotor can be written as algebraic functions of four carefully selected flat outputs and their derivatives

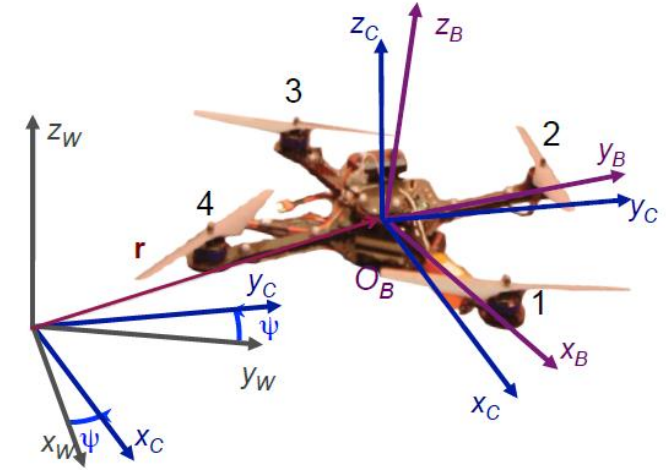
- Enables automated generation of trajectories
- Any smooth trajectory in the space of flat outputs (with reasonably bounded derivatives) can be followed by the under-actuated quadrotor
- A possible choice:
  - $\sigma = [x, y, z, \psi]^T$
- Trajectory in the space of flat outputs:
  - $\sigma(t) = [T_0, T_M] \rightarrow \mathbb{R}^3 \times SO(2)$



# Quadrotor dynamics

- Quadrotor states  
Position, orientation, linear velocity, angular velocity

$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$



- Nonlinear dynamics

Newton Equation: 
$$m\ddot{\mathbf{p}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \mathbf{u}_1$$

Euler Equation: 
$$\mathbf{I} \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \mathbf{I} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} \begin{matrix} \mathbf{u}_2 \\ \mathbf{u}_3 \\ \mathbf{u}_4 \end{matrix}$$



# Differential Flatness

- **Quadrotor states**

- Position, orientation, linear velocity, angular velocity

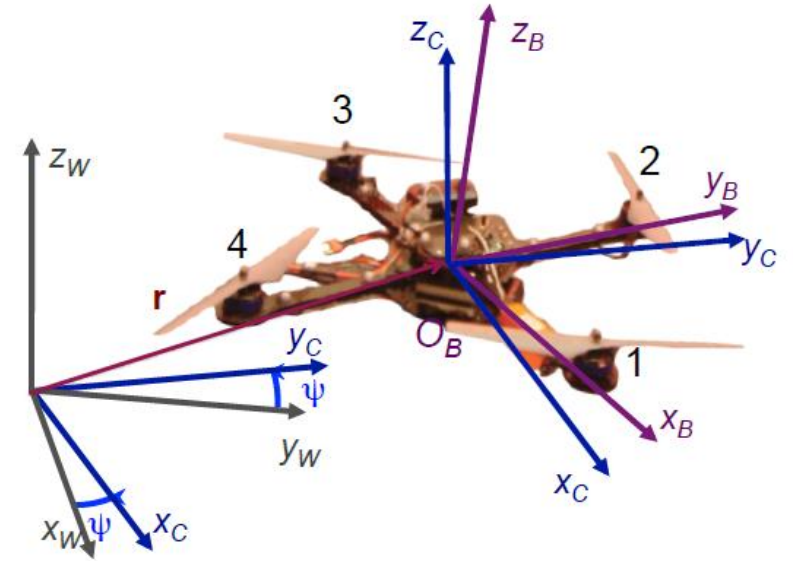
$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

Body angular velocity  
viewed in the body frame

- Equation of motions:

$$m\ddot{\mathbf{p}} = -mg\mathbf{z}_W + u_1\mathbf{z}_B.$$

$$\boldsymbol{\omega}_B = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad \dot{\boldsymbol{\omega}}_B = \mathbf{I}^{-1} \left[ -\boldsymbol{\omega}_B \times \mathbf{I} \cdot \boldsymbol{\omega}_B + \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \right]$$



- Position, velocity, and acceleration are simply derivatives of the flat outputs



# Differential Flatness

- **Orientation**

- Quadrotor state:

$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

- From the equation of motion:

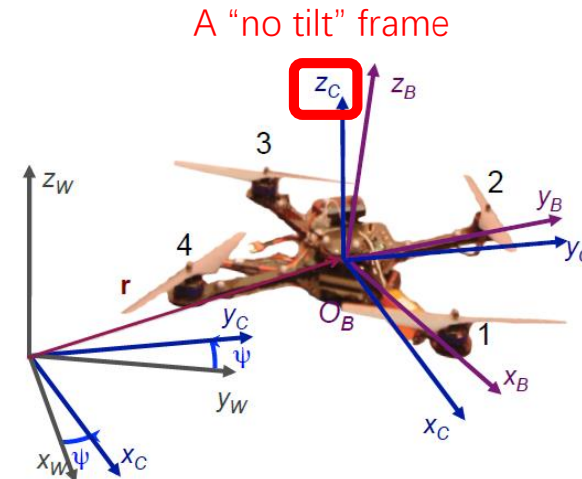
$$\mathbf{z}_B = \frac{\mathbf{t}}{\|\mathbf{t}\|}, \mathbf{t} = [\ddot{\sigma}_1, \ddot{\sigma}_2, \ddot{\sigma}_3 + g]^T$$

- Define the yaw vector (Z-X-Y Euler):

$$\mathbf{x}_C = [\cos \sigma_4, \sin \sigma_4, 0]^T$$

- Orientation can be expressed in terms of flat outputs

$$\mathbf{y}_B = \frac{\mathbf{z}_B \times \mathbf{x}_C}{\|\mathbf{z}_B \times \mathbf{x}_C\|}, \quad \mathbf{x}_B = \mathbf{y}_B \times \mathbf{z}_B \quad \mathbf{R}_B = [\mathbf{x}_B \ \mathbf{y}_B \ \mathbf{z}_B]$$



$$\sigma = [x, y, z, \psi]^T$$



# Differential Flatness

- **Angular velocity**

- Quadrotor state:

$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

- Take the derivative of the equation of motion

$$m\ddot{\mathbf{p}} = -mg\mathbf{z}_W + u_1\mathbf{z}_B.$$



$$m\dot{\mathbf{a}} = \dot{u}_1\mathbf{z}_B + \boxed{\boldsymbol{\omega}_{BW}} \times u_1\mathbf{z}_B$$

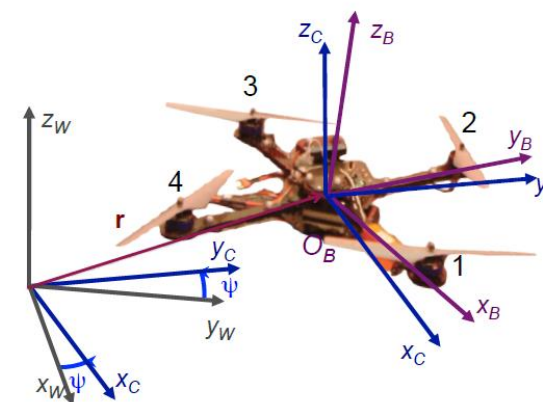
Body angular velocity  
viewed in the world frame

- Quadrotors only have vertical thrust:

$$\dot{u}_1 = \mathbf{z}_B \cdot m\dot{\mathbf{a}}$$

- We have:

$$\mathbf{h}_\omega = \boldsymbol{\omega}_{BW} \times \mathbf{z}_B = \frac{m}{u_1} (\dot{\mathbf{a}} - (\mathbf{z}_B \cdot \dot{\mathbf{a}})\mathbf{z}_B).$$





# Differential Flatness

- **Angular velocity**

- Quadrotor state:

$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

- We have:

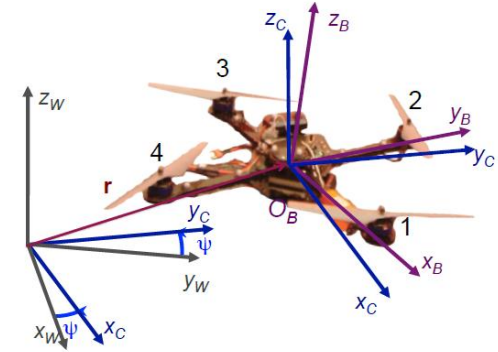
$$\mathbf{h}_\omega = \boldsymbol{\omega}_{BW} \times \mathbf{z}_B = \frac{m}{u_1} (\dot{\mathbf{a}} - (\mathbf{z}_B \cdot \dot{\mathbf{a}})\mathbf{z}_B).$$

- We know that:

$$\boldsymbol{\omega}_{BW} = \omega_x \mathbf{x}_B + \omega_y \mathbf{y}_B + \omega_z \mathbf{z}_B$$

- Angular velocities along  $x_B$  and  $y_B$  directions can be found as:

$$\omega_x = -\mathbf{h}_\omega \cdot \mathbf{y}_B, \quad \omega_y = \mathbf{h}_\omega \cdot \mathbf{x}_B$$





# Differential Flatness

- Angular velocity

- Quadrotor state:

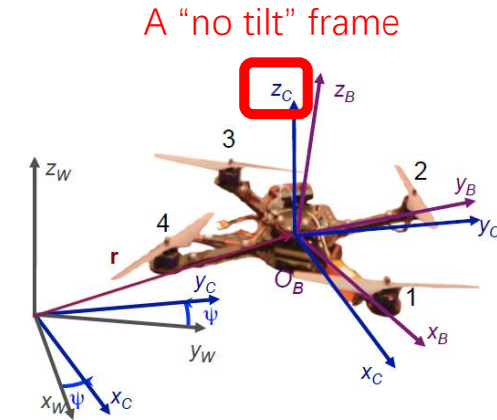
$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

- We have:

$$\mathbf{h}_\omega = \boldsymbol{\omega}_{BW} \times \mathbf{z}_B = \frac{m}{u_1} (\dot{\mathbf{a}} - (\mathbf{z}_B \cdot \dot{\mathbf{a}}) \mathbf{z}_B).$$

- Since  $\boldsymbol{\omega}_{BW} = \boldsymbol{\omega}_{BC} + \boldsymbol{\omega}_{CW}$ , where  $\boldsymbol{\omega}_{BC}$  has no  $\mathbf{z}_B$  component:

$$\omega_z = \boldsymbol{\omega}_{BW} \cdot \mathbf{z}_B = \boldsymbol{\omega}_{CW} \cdot \mathbf{z}_B = \dot{\psi} \mathbf{z}_W \cdot \mathbf{z}_B.$$





# Differential Flatness

- **Summary**

- Quadrotor state:

$$\mathbf{X} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

- Flat outputs:

- $\boldsymbol{\sigma} = [x, y, z, \psi]^T$

- Position, velocity, acceleration:

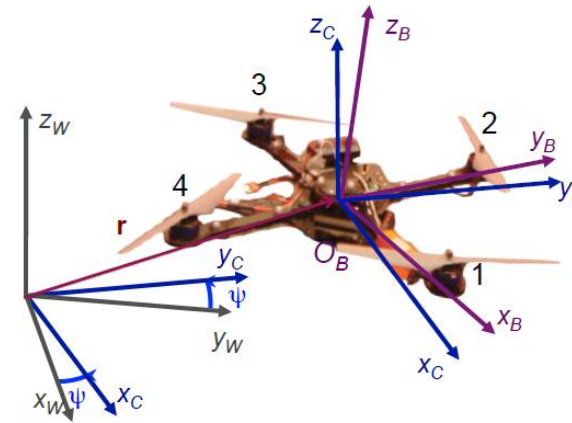
- Derivatives of flat outputs

- Orientation:

$$\mathbf{x}_C = [\cos\sigma_4, \sin\sigma_4, 0]^T \longrightarrow \mathbf{R}_B = [\mathbf{x}_B \ \mathbf{y}_B \ \mathbf{z}_B]$$

- Angular velocity:

$$\omega_x = -\mathbf{h}_\omega \cdot \mathbf{y}_B, \quad \omega_y = \mathbf{h}_\omega \cdot \mathbf{x}_B, \quad \omega_z = \dot{\psi} \mathbf{z}_W \cdot \mathbf{z}_B$$

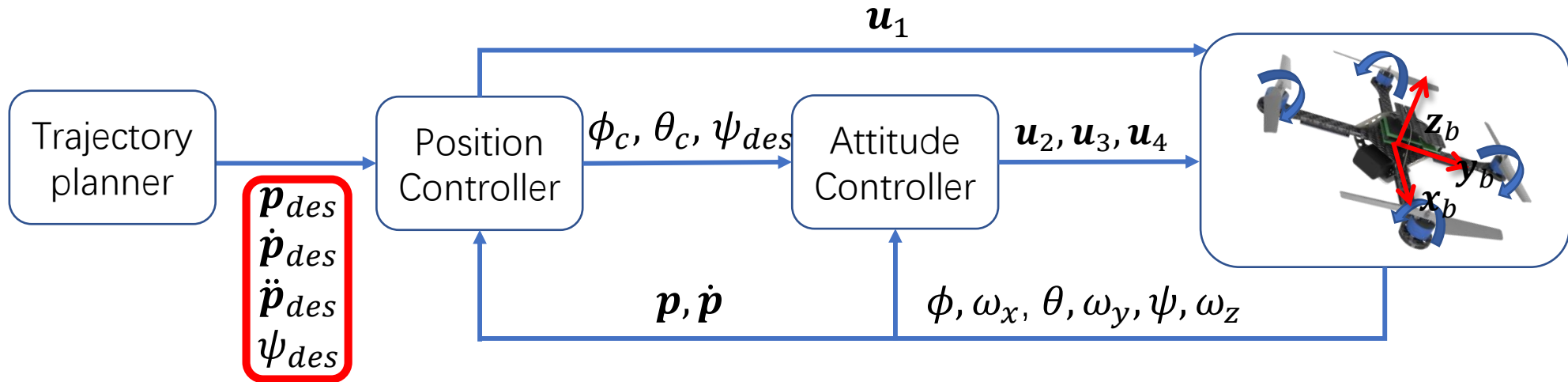


Done. You should only remember the planning of a UAV can be done in  $x, y, z, \psi$ .





## Close the planning-control loop



Nonlinear dynamics

Newton Equation: 
$$m\ddot{\mathbf{p}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}$$

Euler Equation: 
$$\mathbf{I} \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \mathbf{I} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$



# Polynomial Trajectories

- Flat outputs:
  - $\sigma = [x, y, z, \psi]^T$
- Trajectory in the space of flat outputs:
  - $\sigma(t) = [T_0, T_M] \rightarrow \mathbb{R}^3 \times SO(2)$
- Polynomial functions can be used to specify trajectories in the space of flat outputs
  - Easy determination of smoothness criterion with polynomial orders
  - Easy and closed form calculation of derivatives
  - Decoupled trajectory generation in three dimensions

# Minimum-snap

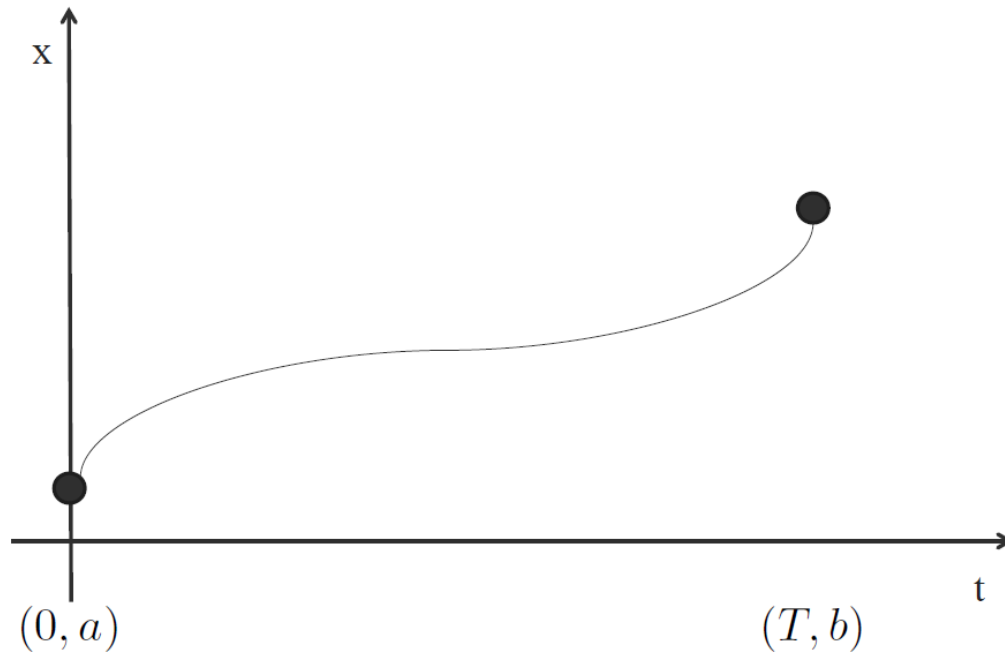


# Smooth 1D Trajectory

Just a simple BVP


- Design a trajectory  $x(t)$  such that:
  - $x(0) = a$
  - $x(T) = b$

} Boundary condition
- Smoothness ensured by parametrization Smoothness criteria





# Smooth 1D Trajectory

- 5<sup>th</sup> order polynomial trajectory:
  - $x(t) = p_5t^5 + p_4t^4 + p_3t^3 + p_2t^2 + p_1t + p_0$   Trajectory parametrization
- Boundary conditions      No intermediate condition

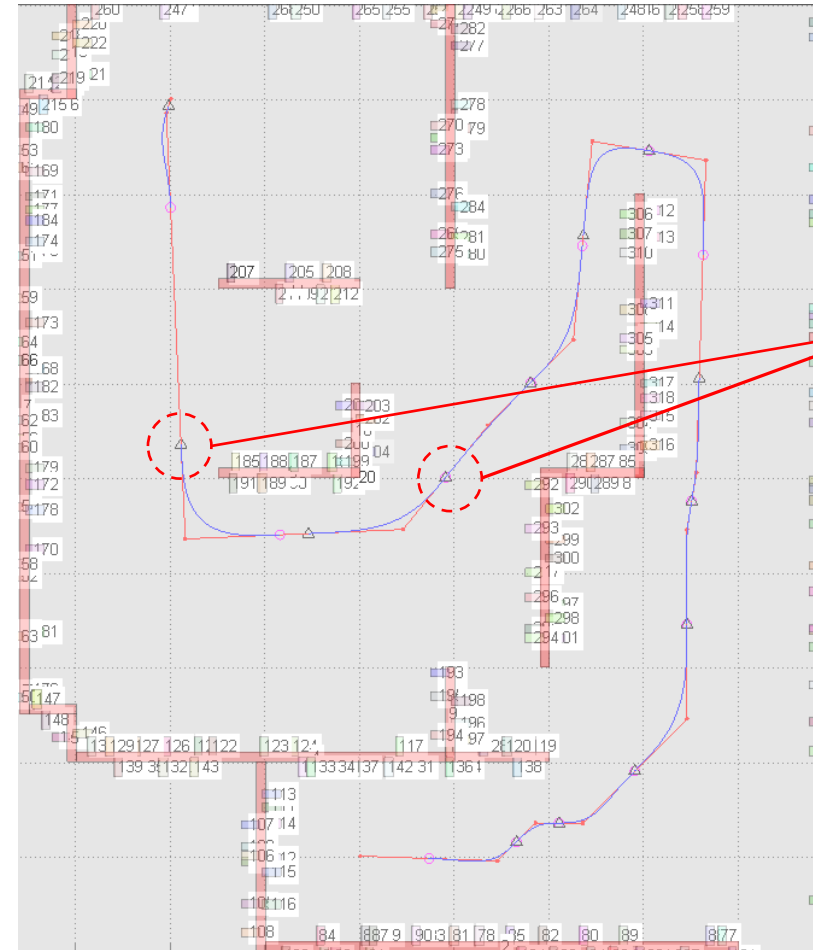
	Position	Velocity	Acceleration
t = 0	a	0	0
t = T	b	0	0

- Solve:

$$\begin{bmatrix} a \\ b \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix}$$

# Smooth Multi-Segment Trajectory

- Smooth the corners of straight line segments.
- Preferred constant velocity motion at  $v$ .
- Preferred zero acceleration.
- Requires special handling of short segments.



Intermediate conditions

## Smooth Multi-Segment 1D Trajectory

- Generate each 5<sup>th</sup> order polynomial independently:
  - $x(t) = p_5 t^5 + p_4 t^4 + p_3 t^3 + p_2 t^2 + p_1 t + p_0$
- Boundary conditions

	Position	Velocity	Acceleration
t = 0	a	$v_0$	0
t = T	b	$v_T$	0

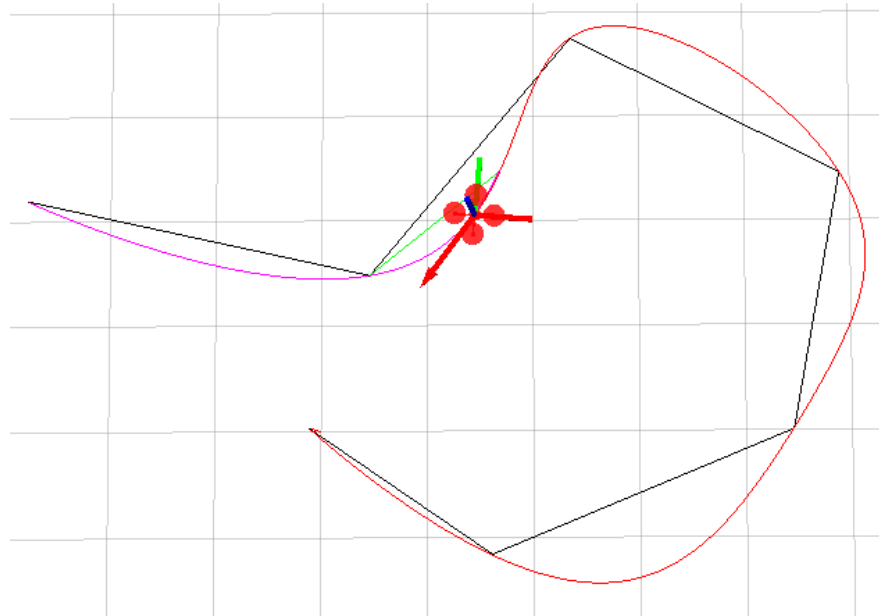
- Solve:

$$\begin{bmatrix} a \\ b \\ v_0 \\ v_T \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix}$$



# Smooth 3D Trajectory

- Boundary condition: start, goal positions (orientations)
- Intermediate condition: waypoint positions (orientations)
  - Waypoints can be found by path planning (A\*, RRT\*, etc)
  - Introduced in previous 3 lectures
- Smoothness criterion
  - Generally translates into minimizing rate of change of “input”







# Optimization-based Trajectory Generation

- Explicitly minimize certain derivatives in the space of flat outputs
- Quadrotor dynamics

Derivative	Translation	Rotation	Thrust
0	Position		
1	Velocity		
2	Acceleration	Rotation	
3	Jerk	Angular Velocity	Thrust
4	Snap	Angular Acceleration	Differential Thrust



# Optimization-based Trajectory Generation

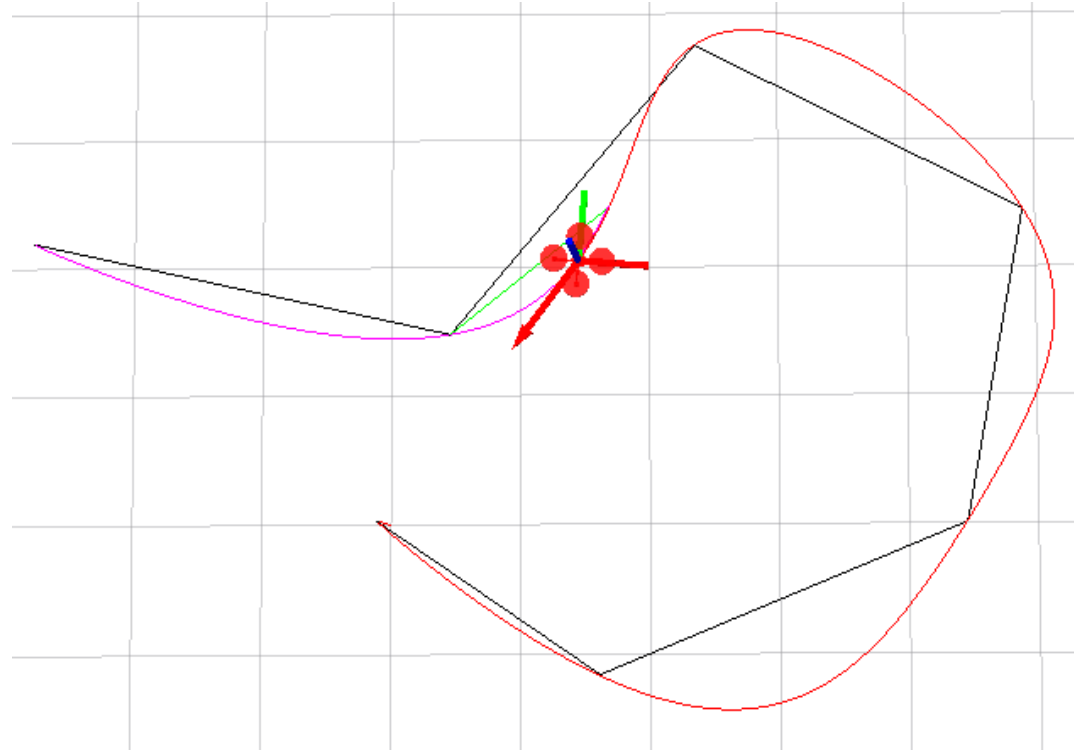
- Explicitly minimize certain derivatives in the space of flat outputs
  - Minimum jerk: minimize angular velocity, good for visual tracking
  - Minimum snap: minimize differential thrust, saves energy

Derivative	Translation	Rotation	Thrust
0	Position		
1	Velocity		
2	Acceleration	Rotation	
3	Jerk	Angular Velocity	Thrust
4	Snap	Angular Acceleration	Differential Thrust



# Optimization-based Trajectory Generation

- Multi-segment minimum snap trajectory



# Minimum Snap Trajectory Generation

Formulation:

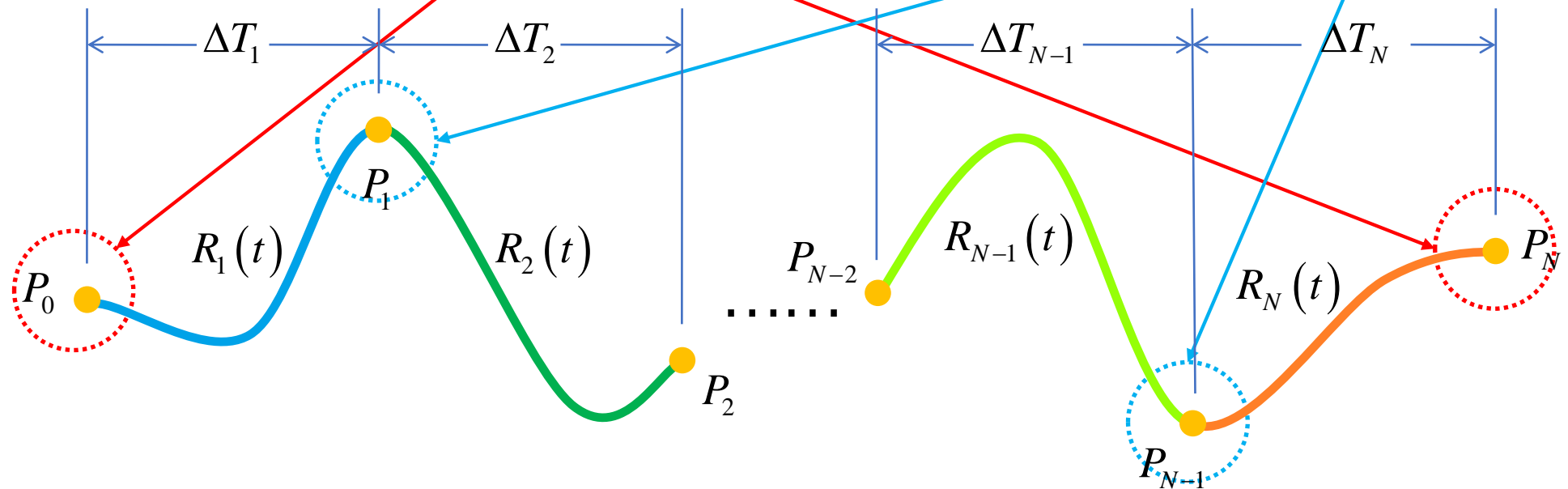
$$f(t) = \begin{cases} f_1(t) \doteq \sum_{i=0}^N p_{1,i} t^i & T_0 \leq t \leq T_1 \\ f_2(t) \doteq \sum_{i=0}^N p_{2,i} t^i & T_1 \leq t \leq T_2 \\ \vdots & \vdots \\ f_M(t) \doteq \sum_{i=0}^N p_{M,i} t^i & T_{M-1} \leq t \leq T_M \end{cases}$$

- Each segment is a polynomial.
- No need to fix the order, but keep the same order make this problem simpler.
- **Time durations** for each segment must be known!

# Minimum Snap Trajectory Generation

Constraints:

- Derivative constraints: 
$$\begin{cases} f_j^{(k)}(T_{j-1}) &= x_{0,j}^{(k)} \\ f_j^{(k)}(T_j) &= x_{T,j}^{(k)} \end{cases}$$
- Continuity constraints: 
$$f_j^{(k)}(T_j) = f_{j+1}^{(k)}(T_j)$$



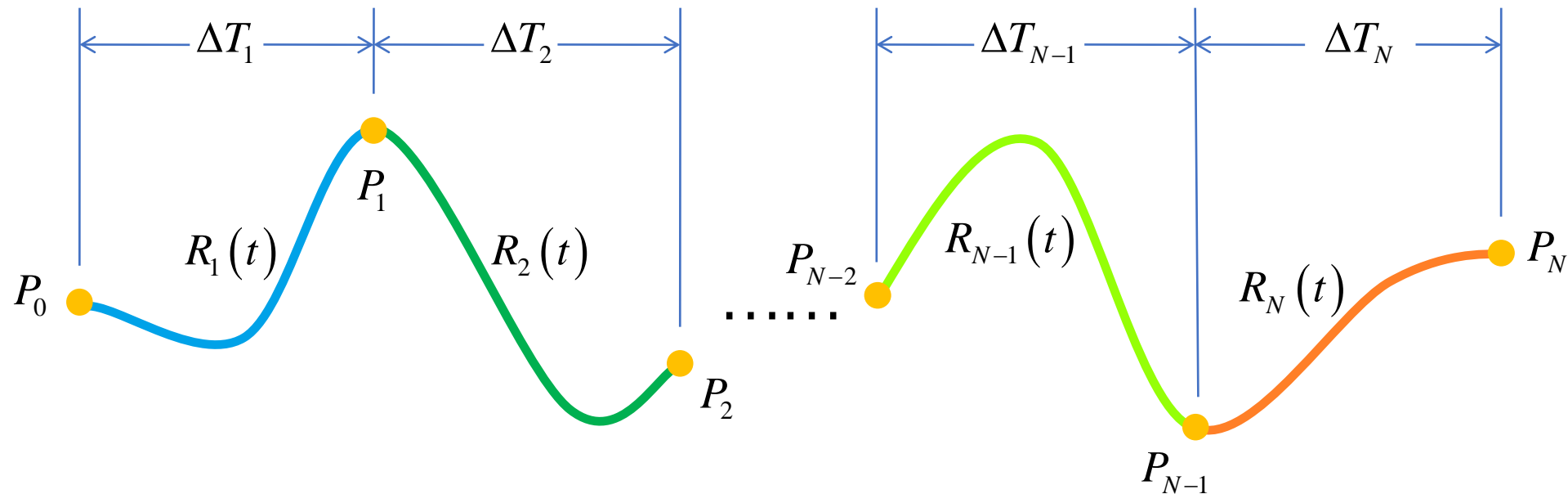
# Minimum Snap Trajectory Generation

How to determine the trajectory order?

- Ensure smoothness at an order.
- Ensure continuity at an order.
- Minimize control input at an order.

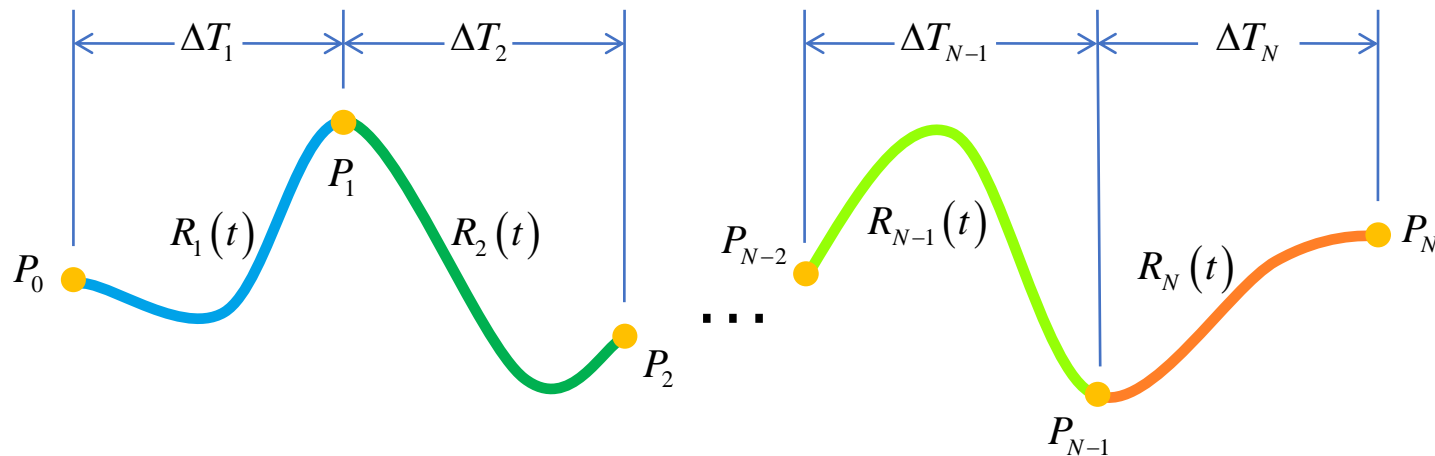
Smoothness means its derivative is continuous!

This three items are **not** coupled!



# Minimum Snap Trajectory Generation

- Minimum degree polynomial to ensure smoothness for one-segment trajectory:
  - Minimum jerk:  $N = 2 * 3(\text{jerk}) - 1 = 5$
  - Minimum snap:  $N = 2 * 4(\text{snap}) - 1 = 7$
- Minimum degree polynomial to ensure smoothness for k-segment trajectory:



Minimum jerk:

Constraints num:  $3 + 3 + (k-1) = k + 5$

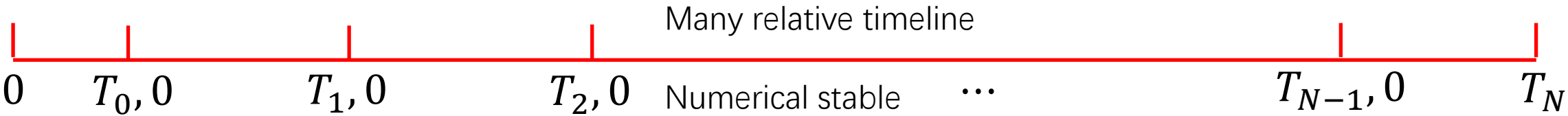
Unknowns num:  $(N+1) * k$

$$(N + 1) * k = k + 5 \quad N = \frac{5}{k}$$

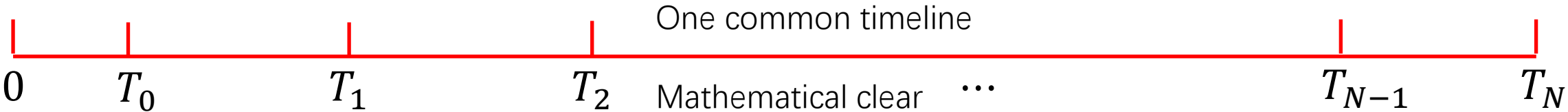
# Minimum Snap Trajectory Generation

Note different timeline

✓ Implementation, homework



✓ Problem formulation, lecture slides





# Minimum Snap Trajectory Generation

- Cost function for one polynomial segment:

$$f(t) = \sum_i p_i t^i$$

$$\Rightarrow f^{(4)}(t) = \sum_{i \geq 4} i(i-1)(i-2)(i-3)t^{i-4}p_i$$

$$\Rightarrow \left(f^{(4)}(t)\right)^2 = \sum_{i \geq 4, l \geq 4} i(i-1)(i-2)(i-3)l(l-1)(l-2)(l-3)t^{i+l-8}p_i p_l$$

$$\Rightarrow J(T) = \int_{T_{j-1}}^{T_j} \left(f^{(4)}(t)\right)^2 dt = \sum_{i \geq 4, l \geq 4} \frac{i(i-1)(i-2)(i-3)l(l-1)(l-2)(l-3)}{i+l-7} (T_j^{i+l-7} - T_{j-1}^{i+l-7}) p_i p_l$$

$$\Rightarrow J(T) = \int_{T_{j-1}}^{T_j} \left(f^{(4)}(t)\right)^2 dt$$

$$= \begin{bmatrix} \vdots \\ p_i \\ \vdots \end{bmatrix}^T \begin{bmatrix} \vdots & & \\ \dots & \frac{i(i-1)(i-2)(i-3)l(l-1)(l-2)(l-3)}{i+l-7} T^{i+l-7} & \dots \\ \vdots & & \end{bmatrix} \begin{bmatrix} \vdots \\ p_l \\ \vdots \end{bmatrix}$$

$$\Rightarrow J_j(T) = \mathbf{p}_j^T \mathbf{Q}_j \mathbf{p}_j \text{ Minimize this!}$$

# Minimum Snap Trajectory Generation

- Derivative constraint for one polynomial segment
  - Also models waypoint constraint ( $0^{th}$  order derivative)

$$\begin{aligned}
 f_j^{(k)}(T_j) &= x_j^{(k)} \\
 \Rightarrow \sum_{i \geq k} \frac{i!}{(i-k)!} T_j^{i-k} p_{j,i} &= x_{T,j}^{(k)} \\
 \Rightarrow \begin{bmatrix} \dots & \frac{i!}{(i-k)!} T_j^{i-k} & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \end{bmatrix} &= x_{T,j}^{(k)} \\
 \Rightarrow \begin{bmatrix} \dots & \frac{i!}{(i-k)!} T_j^{i-k} & \dots \\ \dots & \frac{i!}{(i-k)!} T_j^{i-k} & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \end{bmatrix} &= \begin{bmatrix} x_{0,j}^{(k)} \\ x_{T,j}^{(k)} \end{bmatrix} \\
 \Rightarrow \mathbf{A}_j \mathbf{p}_j &= \mathbf{d}_j
 \end{aligned}$$

$$x(t) = p_5 t^5 + p_4 t^4 + p_3 t^3 + p_2 t^2 + p_1 t + p_0$$

$$x(0) = \dots, x(T) = \dots$$

$$\dot{x}(0) = \dots, \dot{x}(T) = \dots$$

$$\ddot{x}(0) = \dots, \ddot{x}(T) = \dots$$

$$\vdots$$

$$\begin{aligned}
 p_0 &= \dots, \\
 p_5 T^5 + p_4 T^4 + p_3 T^3 + p_2 T^2 + p_1 T + p_0 &= \dots
 \end{aligned}$$

$$[T^5, T^4, T^3, T^2, T, 1] \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \dots$$

# Minimum Snap Trajectory Generation

- Continuity constraint between two segments:
  - Ensures continuity between trajectory segments when no specific derivatives are given

$$\begin{aligned} f_j^{(k)}(T_j) &= f_{j+1}^{(k)}(T_j) \\ \Rightarrow \sum_{i \geq k} \frac{i!}{(i-k)!} T_j^{i-k} p_{j,i} - \sum_{l \geq k} \frac{l!}{(l-k)!} T_j^{l-k} p_{j+1,l} &= 0 \\ \Rightarrow \left[ \dots \quad \frac{i!}{(i-k)!} T_j^{i-k} \quad \dots \quad - \frac{l!}{(l-k)!} T_j^{l-k} \quad \dots \right] \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \\ p_{j+1,l} \\ \vdots \end{bmatrix} &= 0 \\ \Rightarrow [\mathbf{A}_j \quad -\mathbf{A}_{j+1}] \begin{bmatrix} \mathbf{p}_j \\ \mathbf{p}_{j+1} \end{bmatrix} &= 0 \end{aligned}$$

# Minimum Snap Trajectory Generation

- Constrained quadratic programming (QP) formulation:

$$\begin{aligned} \min \quad & \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} \\ \text{s. t.} \quad & \mathbf{A}_{eq} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \mathbf{d}_{eq} \end{aligned}$$

It's a typical **convex optimization** program.



# **Aggressive Quadrotor Part II**

**Daniel Mellinger and Vijay Kumar**  
**GRASP Lab, University of Pennsylvania**

# Convex Optimization



# Convex function and convex set

## Convex function

- A function  $f: R^n \rightarrow R$  is said to be **convex** if the domain,  $\text{dom } f$ , is convex and for any  $x, y \in \text{dom } f$  and  $0 \leq \theta \leq 1$ ,

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

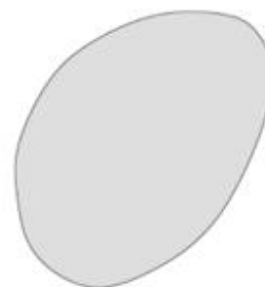


- $f$  is strictly convex if the inequality is strict for  $0 < \theta < 1$ .
- $f$  is concave if  $-f$  is convex.

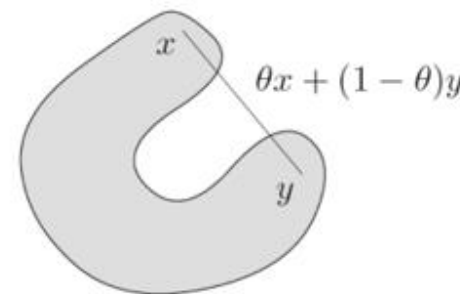
## Convex set

- A set  $C \subseteq R^n$  is said to be **convex** if the line segment between any two points is in the set: for any  $x, y \in C$  and  $0 \leq \theta \leq 1$ ,

$$\theta x + (1 - \theta)y \in C.$$



convex



non-convex



# Convex optimization

- Optimization problem in standard form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \quad i = 1, \dots, m \\ & h_i(x) = 0 \quad i = 1, \dots, p \end{array}$$

$x \in R^n$  is the optimization variable

$f_0: R^n \rightarrow R$  is the objective function

$f_i: R^n \rightarrow R, i = 1, \dots, m$  are inequality constraint functions

$h_i: R^n \rightarrow R, i = 1, \dots, p$  are equality constraint functions

- Convex optimization problem in standard form:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \quad i = 1, \dots, m \\ & Ax = b \end{array}$$

where  $f_0, f_1, \dots, f_m$  are convex and equality constraints are affine.

- **Local and global optima:** any locally optimal point of a convex problem is globally optimal.
- Most problems are not convex when formulated.
- Reformulating a problem in convex form is an art, there is no systematic way.





# Convex optimization

- Optimization problem in standard form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \quad i = 1, \dots, m \\ & h_i(x) = 0 \quad i = 1, \dots, p \end{array}$$

$x \in R^n$  is the optimization variable

$f_0: R^n \rightarrow R$  is the objective function

$f_i: R^n \rightarrow R, i = 1, \dots, m$  are inequality constraint functions

$h_i: R^n \rightarrow R, i = 1, \dots, p$  are equality constraint functions

- Convex optimization problem in standard form:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \quad i = 1, \dots, m \\ & Ax = b \end{array}$$

where  $f_0, f_1, \dots, f_m$  are convex and equality constraints are affine.

- **Local and global optima:** any locally optimal point of a convex problem is globally optimal.
- Most problems are not convex when formulated.
- Reformulating a problem in convex form is an art, there is no systematic way.



# Disciplined convex optimization programs

## Linear Programming (LP)

$$\begin{array}{ll}\underset{x}{\text{minimize}} & c^T x + d \\ \text{subject to} & Gx \leq h \\ & Ax = b\end{array}$$

- Convex problem: affine objective and constraint functions.

## Quadratically Constrained QP (QCQP)

$$\begin{array}{ll}\underset{x}{\text{minimize}} & (1/2)x^T P_0 x + q_0^T x + r_0 \\ \text{subject to} & (1/2)x^T P_i x + q_i^T x + r_i \leq 0 \quad i = 1, \dots, m \\ & Ax = b\end{array}$$

- Convex problem (assuming  $P_i \in S^n \geq 0$ ): convex quadratic objective and constraint functions.

## Quadratic Programming (QP)

$$\begin{array}{ll}\underset{x}{\text{minimize}} & (1/2)x^T P x + q^T x + r \\ \text{subject to} & Gx \leq h \\ & Ax = b\end{array}$$

- Convex problem (assuming  $P \in S^n \geq 0$ ): convex quadratic objective and affine constraint functions.

## Second-Order Cone Programming (SOCP)

$$\begin{array}{ll}\underset{x}{\text{minimize}} & f^T x \\ \text{subject to} & \|A_i x + b_i\| \leq c_i^T x + d_i \quad i = 1, \dots, m \\ & Fx = g\end{array}$$

- Convex problem: linear objective and second-order cone constraints
- For  $A_i$  row vector, it reduces to an LP.
- For  $c_i = 0$ , it reduces to a QCQP.
- More general than QCQP and LP.

# Closed-form Solution to Minimum Snap



## Decision variable mapping

- Direct optimization of polynomial trajectories is numerically unstable
- A change of variable that instead optimizes segment endpoint derivatives is preferred
- We have  $\mathbf{M}_j \mathbf{p}_j = \mathbf{d}_j$ , where  $\mathbf{M}_j$  is a mapping matrix that maps polynomial coefficients to derivatives

$$J = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} \quad J = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}_M \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}_M \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}$$



# Decision variable mapping

$$J = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} \quad + \quad \mathbf{M}_j \mathbf{p}_j = \mathbf{d}_j$$



$$J = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}_M \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}_M \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}$$

$$x(t) = p_5 t^5 + p_4 t^4 + p_3 t^3 + p_2 t^2 + p_1 t + p_0$$

$$x'(t) = 5p_5 t^4 + 4p_4 t^3 + 3p_3 t^2 + 2p_2 t + p_1$$

$$x''(t) = 20p_5 t^3 + 12p_4 t^2 + 6p_3 t + 2p_2$$

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix}$$



## Fixed and free variable separation

- Use a selection matrix  $\mathbf{C}$  to separate free ( $\mathbf{d}_P$ ) and constrained ( $\mathbf{d}_F$ ) variables
  - Free variables : derivatives unspecified, only enforced by continuity constraints

$$\mathbf{C}^T \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} \quad \longrightarrow \quad J = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \underbrace{\mathbf{C}\mathbf{M}^{-T}\mathbf{Q}\mathbf{M}^{-1}\mathbf{C}^T}_{\mathbf{R}} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{R}_{FF} & \mathbf{R}_{FP} \\ \mathbf{R}_{PF} & \mathbf{R}_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}$$

- Turned into an unconstrained quadratic programming that can be solved in closed form:

$$J = \mathbf{d}_F^T \mathbf{R}_{FF} \mathbf{d}_F + \mathbf{d}_F^T \mathbf{R}_{FP} \mathbf{d}_P + \mathbf{d}_P^T \mathbf{R}_{PF} \mathbf{d}_F + \mathbf{d}_P^T \mathbf{R}_{PP} \mathbf{d}_P$$

$$\mathbf{d}_P^* = -\mathbf{R}_{PP}^{-1} \mathbf{R}_{FP}^T \mathbf{d}_F$$



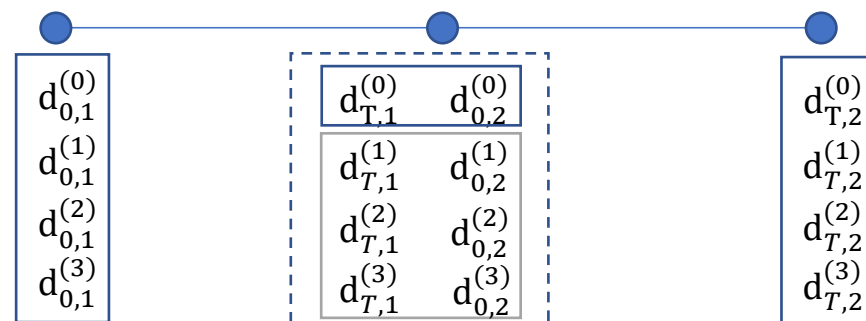
# Build the selection matrix

$d_{0,i}^{(k)}$   
 Index of derivative  
 Index of segment  
 Time index

Fixed derivatives: fixed start, goal state, and intermediate positions

Free derivatives: all derivatives at intermediate connections.

$$\begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} = \mathbf{C}^T \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}$$



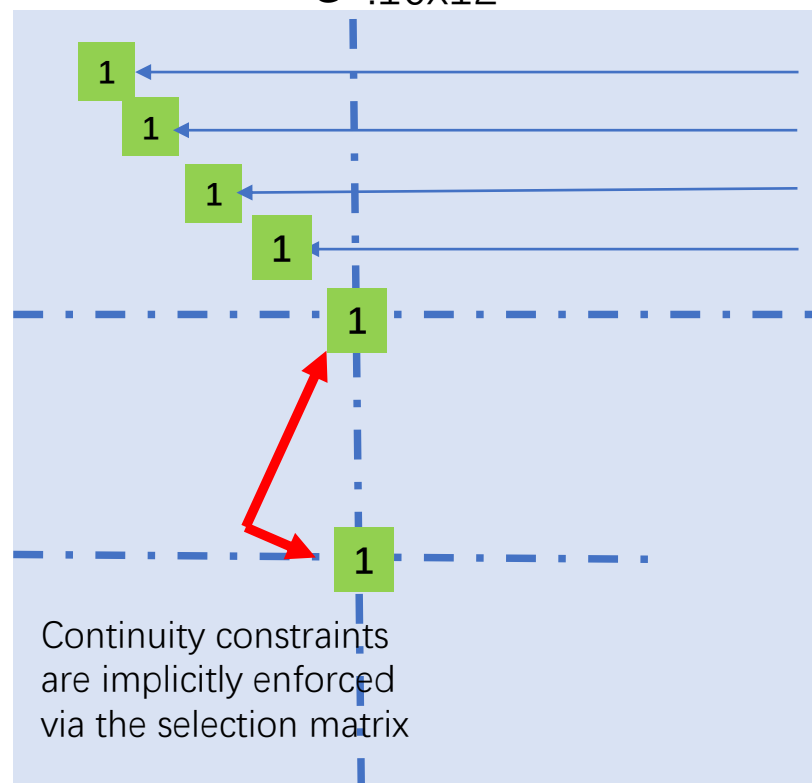
16x1

$\mathbf{C}^T: 16 \times 12$

12x1

$d_{0,1}^{(0)}$   
 $d_{0,1}^{(1)}$   
 $d_{0,1}^{(2)}$   
 $d_{0,1}^{(3)}$   
 $d_{T,1}^{(0)}$   
 $d_{T,1}^{(1)}$   
 $d_{T,1}^{(2)}$   
 $d_{T,1}^{(3)}$   
 $d_{0,2}^{(0)}$   
 $d_{0,2}^{(1)}$   
 $d_{0,2}^{(2)}$   
 $\vdots$

=

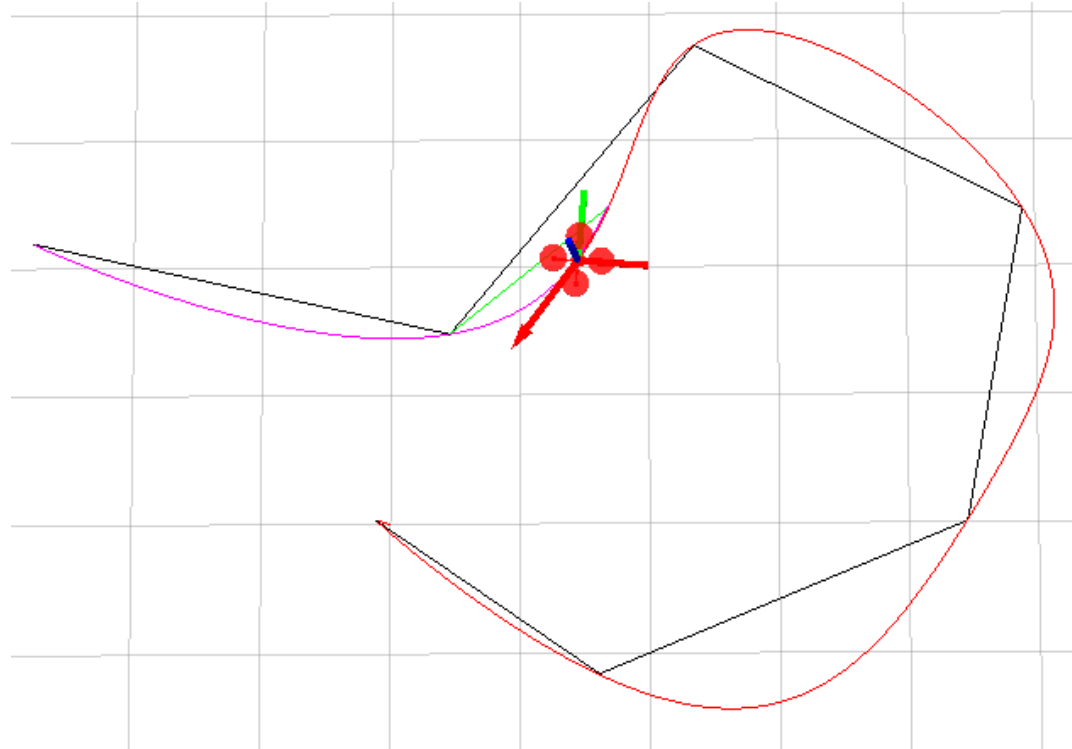


Continuity constraints are implicitly enforced via the selection matrix

$d_{0,1}^{(0)}$   
 $d_{0,1}^{(1)}$   
 $d_{0,1}^{(2)}$   
 $d_{0,1}^{(3)}$   
 $d_{T,1}^{(0)}$   
 $d_{T,1}^{(1)}$   
 $d_{T,1}^{(2)}$   
 $d_{T,1}^{(3)}$   
 $d_{0,2}^{(0)}$   
 $d_{0,2}^{(1)}$   
 $d_{0,2}^{(2)}$   
 $d_{T,2}^{(0)}$   
 $d_{T,2}^{(1)}$   
 $d_{T,2}^{(2)}$   
 $d_{T,2}^{(3)}$

## Same results as convex optimization

- Final trajectory



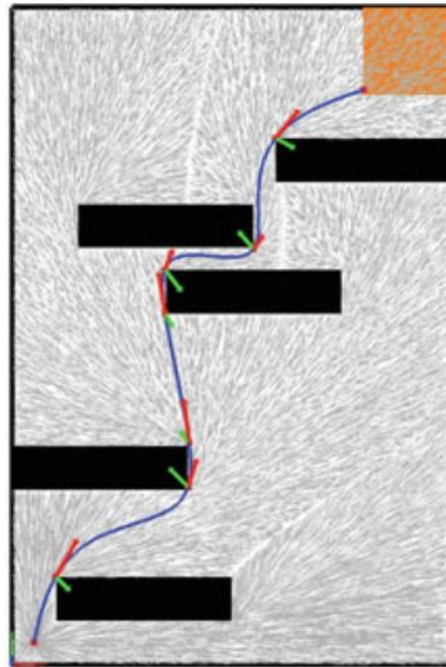
**Ask:** how to get these waypoints?



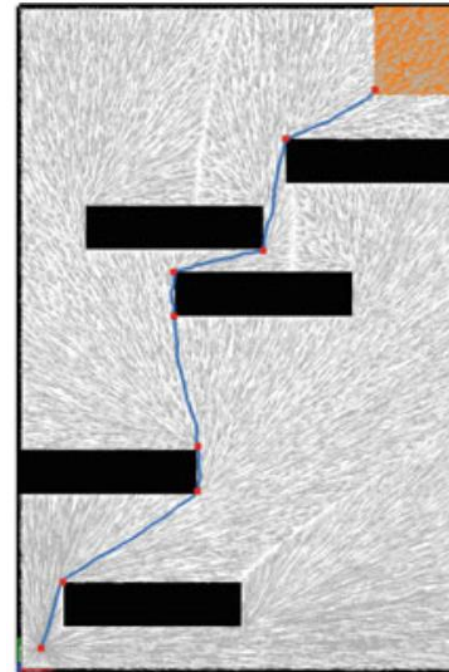


# Hierarchical approach

- path planning + trajectory generation
  - Low complexity solution
    - Path planning can be more efficient since it's in a much lower dimension state space.



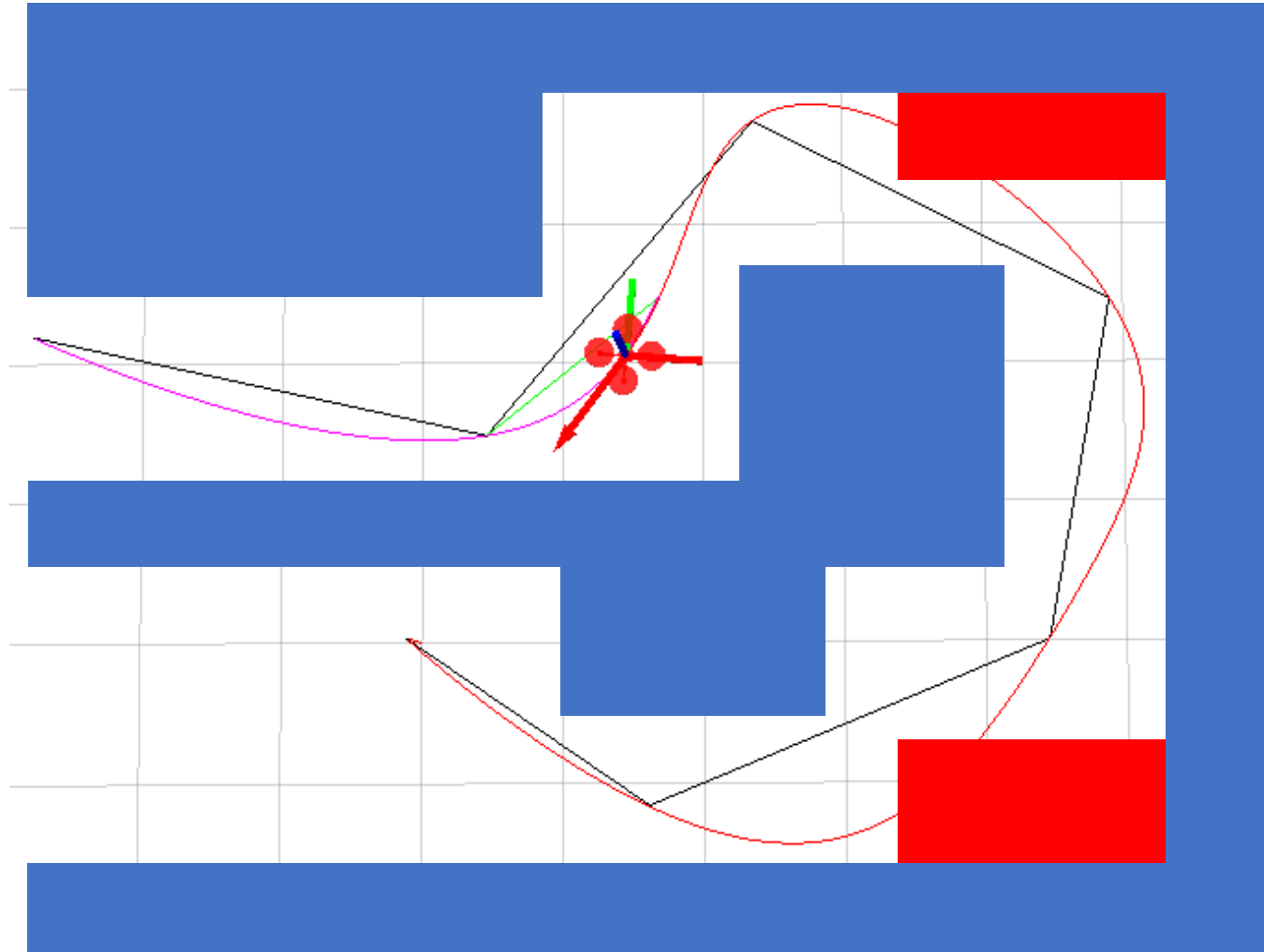
We already know how to fit the polynomial for given waypoints



Then how to get these collision-free waypoints? → the role of path planning



## Problem: safety issue

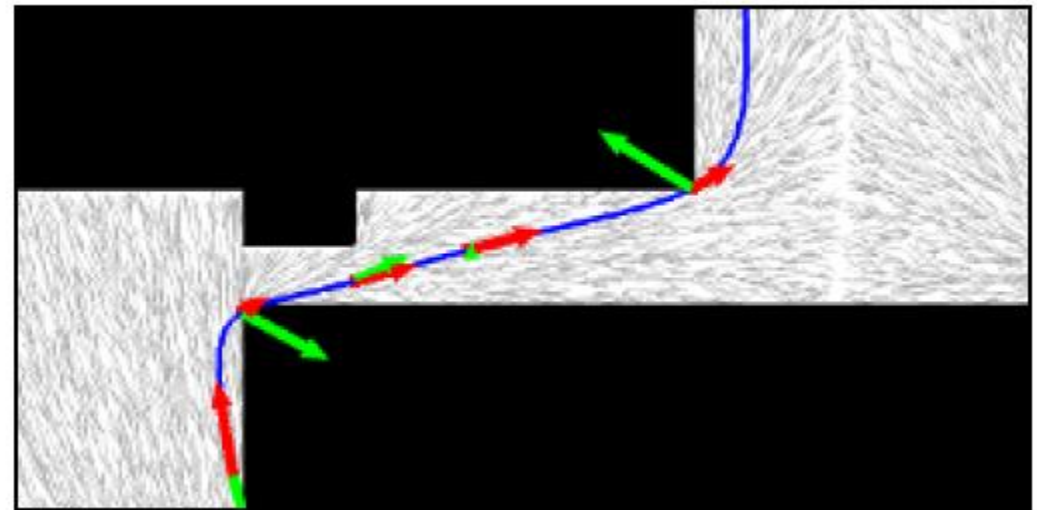
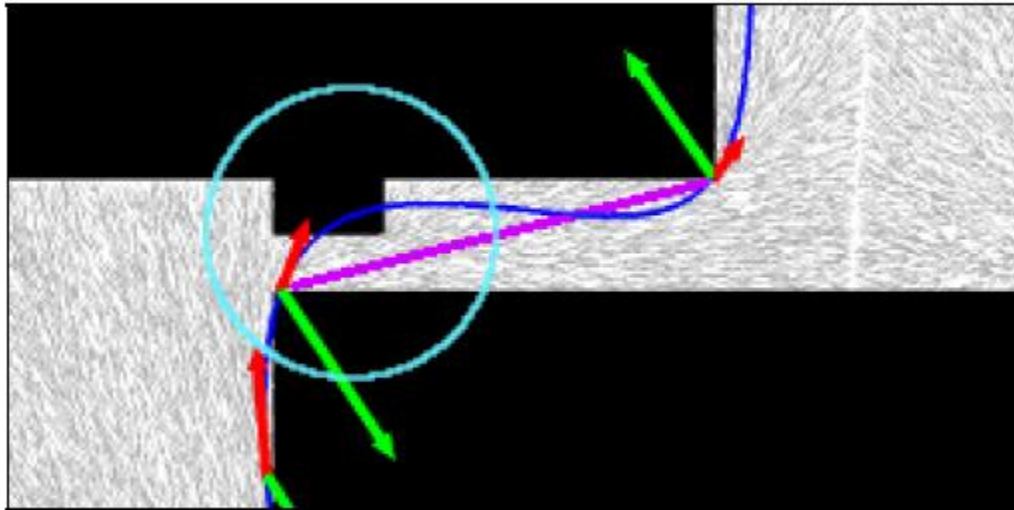


**Ask:** How to Ensure Collision-Free Trajectories?



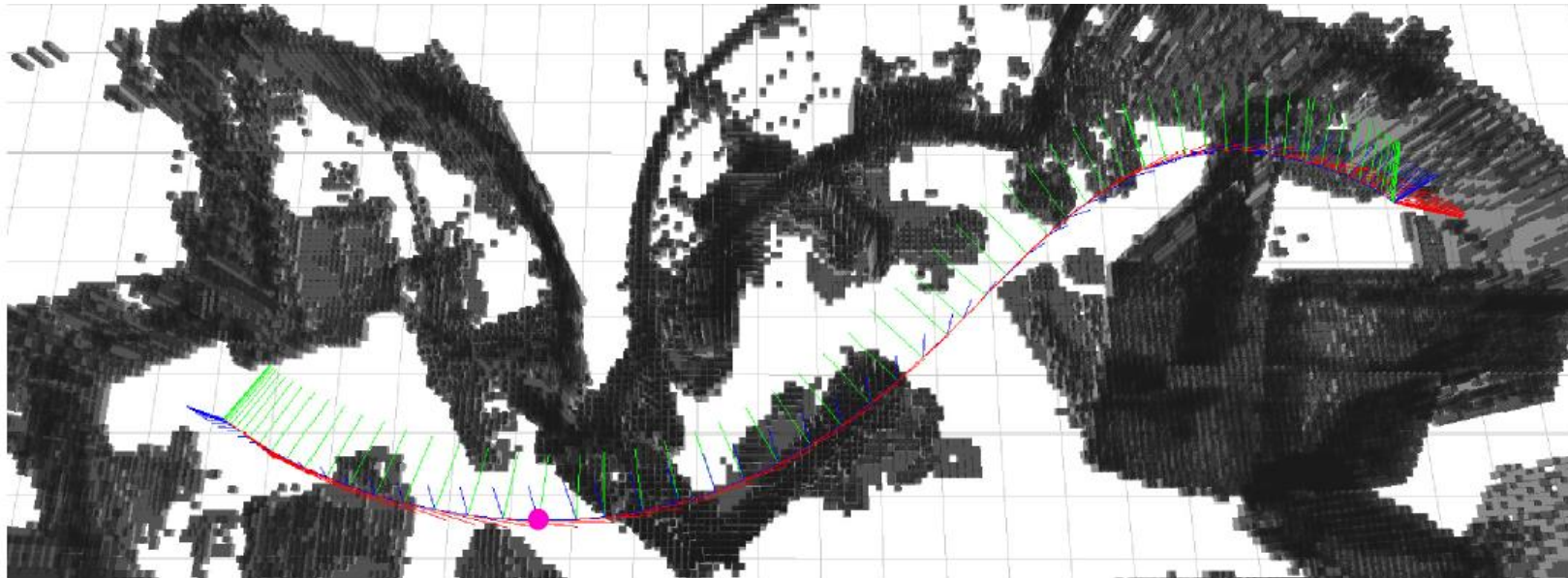
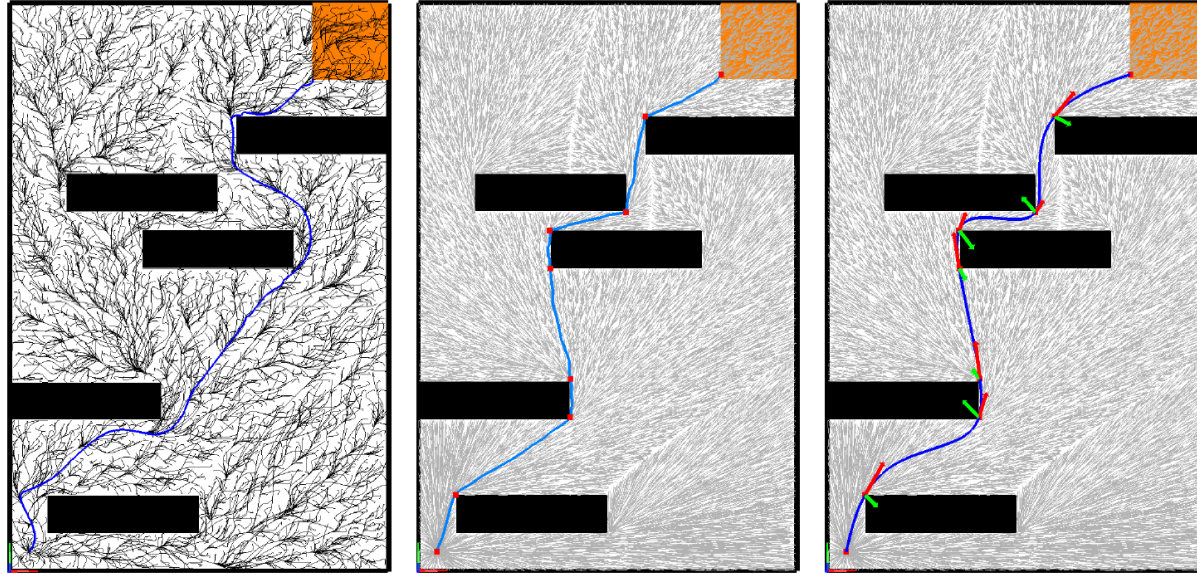
## Iterative approach

- The initial path is collision-free.
- We can approach the trajectory to the path iteratively.
- This is done by adding intermediate waypoints





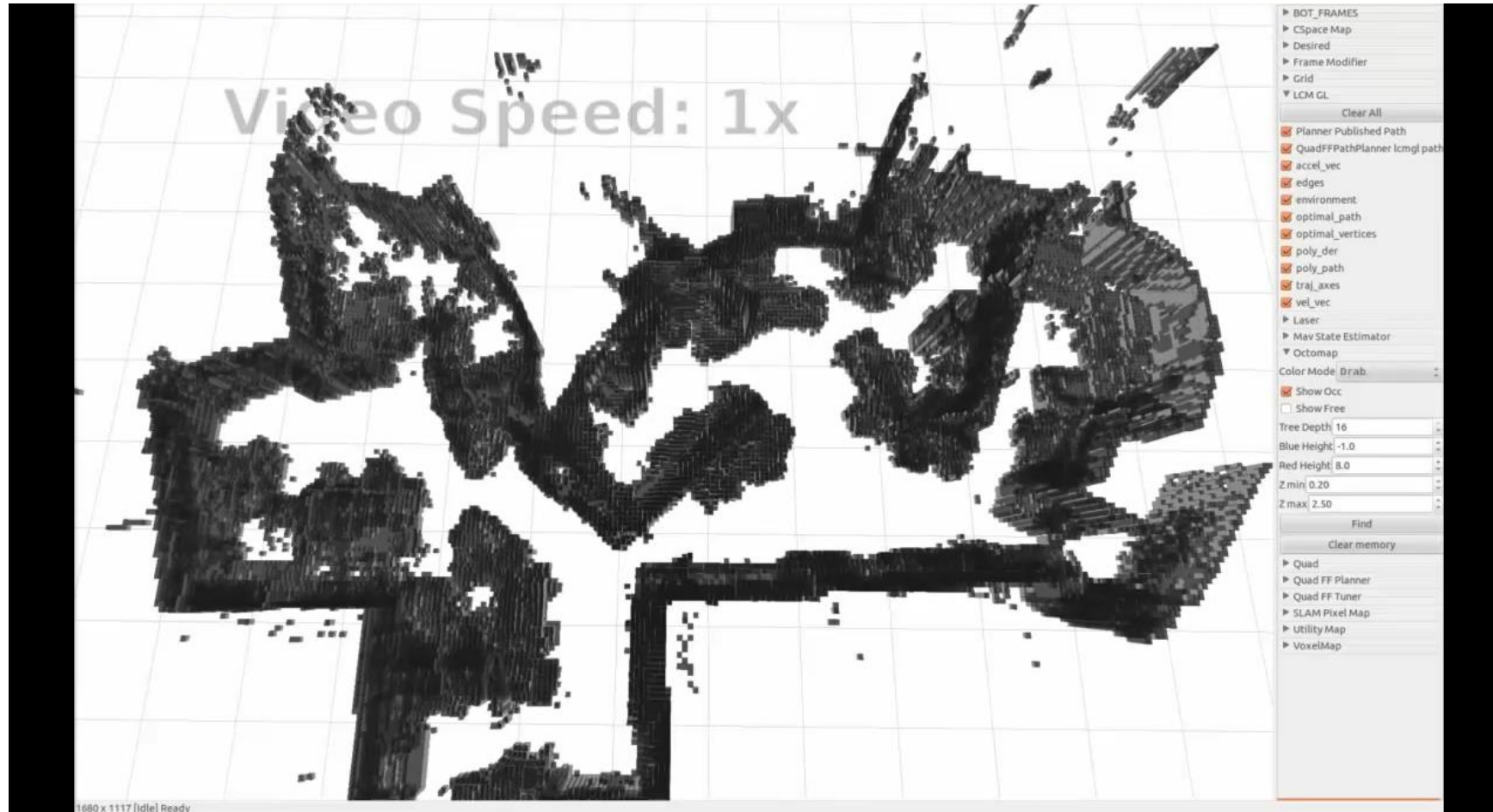
# RRT\* + minimum snap





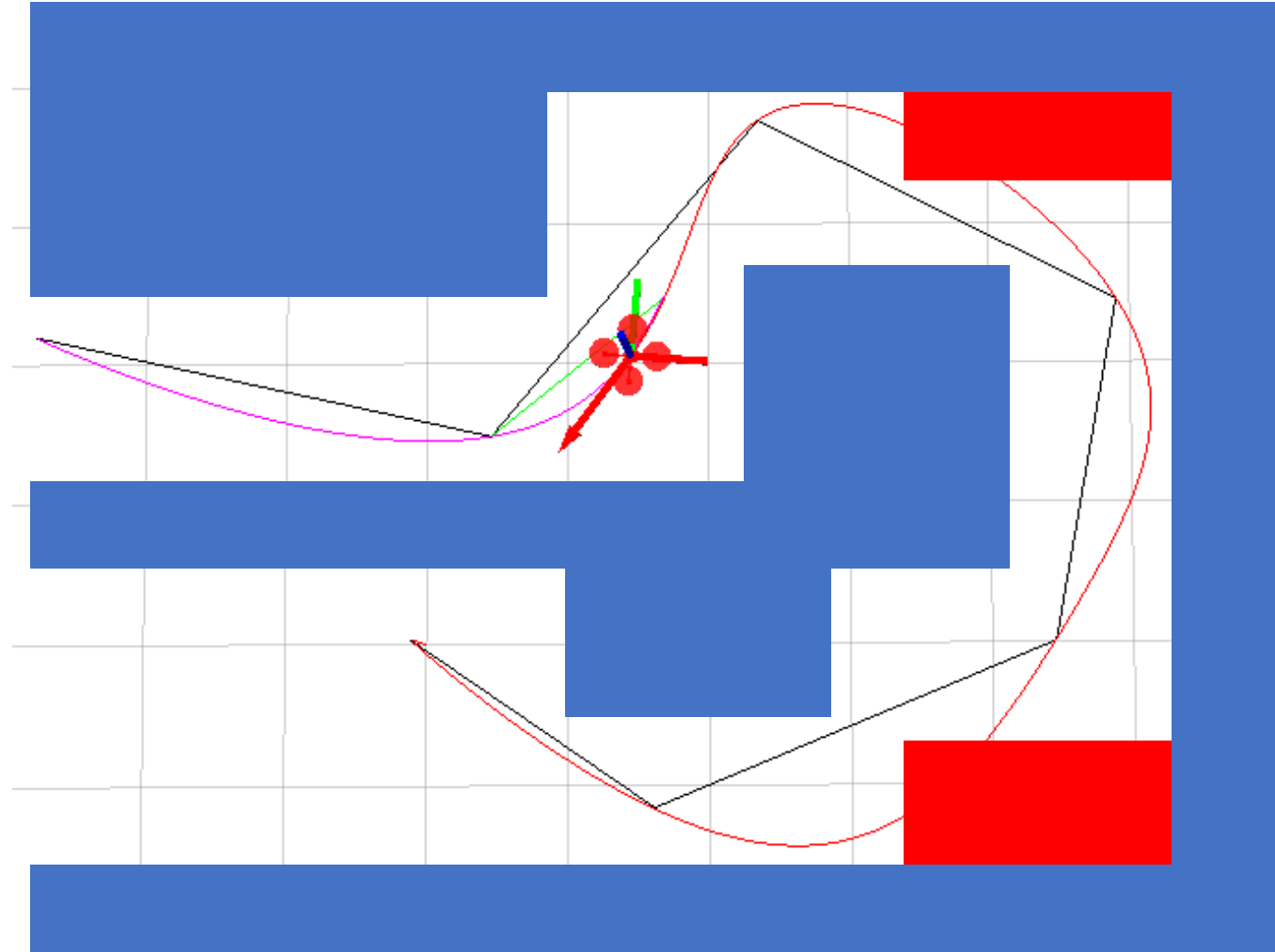


# Experimental validation





## Better solutions?

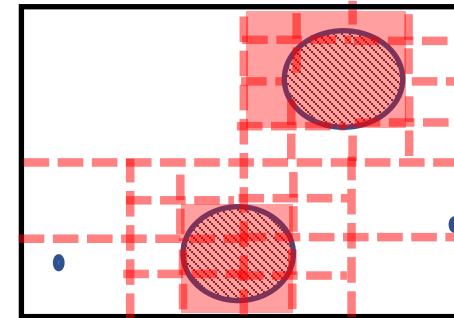




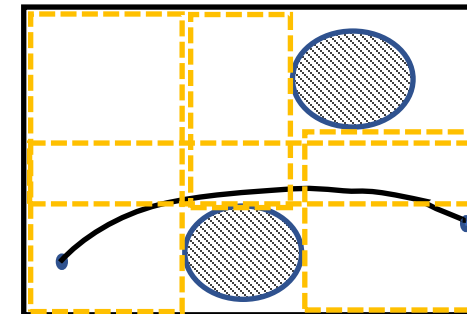
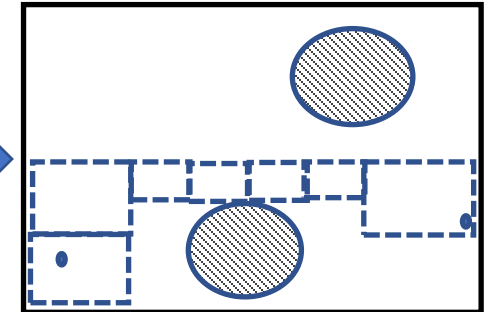
# Smooth Trajectory Generation with Guaranteed Obstacle Avoidance



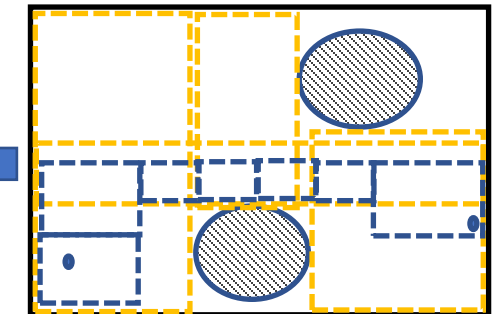
Step 1. detect obstacles



Step 2. search a flight corridor



Step 4. generate **dynamically-feasible** trajectories that fits **entirely** within the flight corridor

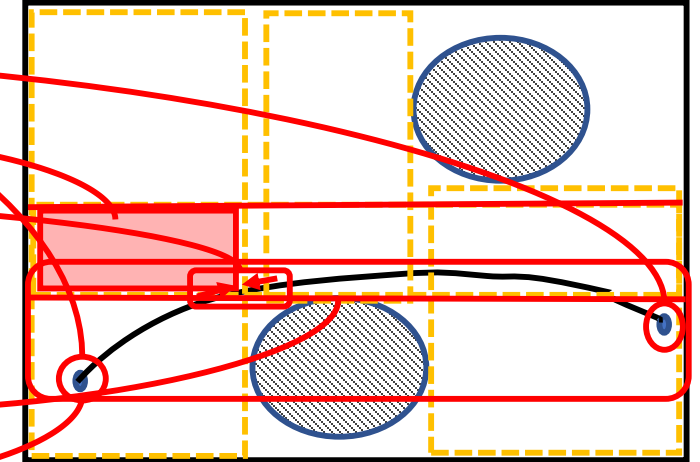


Step 3. inflate flight corridor



# Smooth Trajectory Generation with Guaranteed Obstacle Avoidance

- **Instant** linear constraints:
  - Start, goal state constraint ( $\mathbf{A}\mathbf{p} = \mathbf{b}$ )
  - Transition point constraint ( $\mathbf{A}\mathbf{p} = \mathbf{b}, \mathbf{A}\dot{\mathbf{p}} \leq \mathbf{b}$ )
  - Continuity constraint ( $\mathbf{A}\mathbf{p}_i = \mathbf{A}\mathbf{p}_{i+1}$ )
- **Interval** linear constraints:
  - Boundary constraint ( $\mathbf{A}(t)\mathbf{p} \leq \mathbf{b}, \forall t \in [t_l, t_r]$ )
  - Dynamic constraint ( $\mathbf{A}(t)\dot{\mathbf{p}} \leq \mathbf{b}, \forall t \in [t_l, t_r]$ )
    - Velocity constraints
    - Acceleration constraints

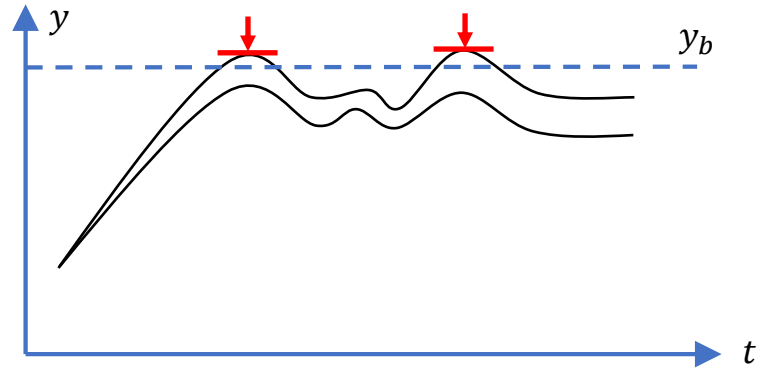






# How to make constraints globally activated

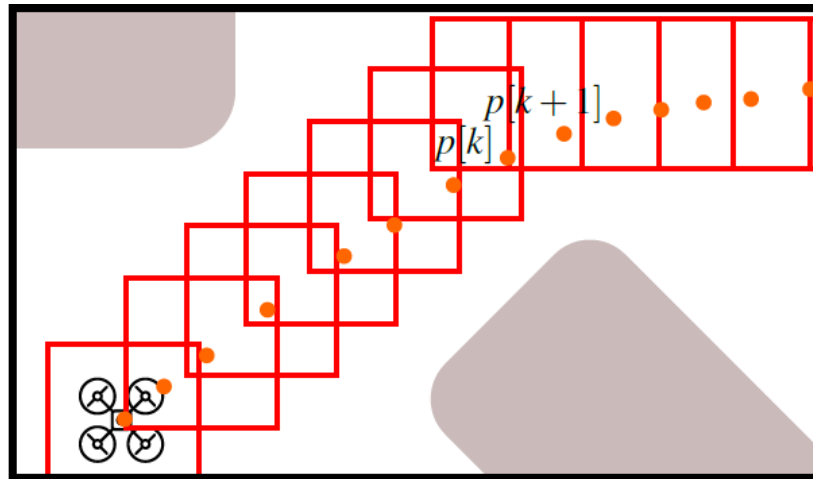
- Iteratively check extremum and add extra constraints.



*Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, J. Chen, ICRA 2016*

- 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 ?

- Adding numerous constraints at discrete time ticks.



*A hybrid method for online trajectory planning of mobile robots in cluttered environments, L Campos-Macías, RAL 2017*

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



# II. Autonomous Flight in Cluttered Indoor Environments



## Solution 2



# Implementation Details

# Convex Solvers



# Solve a convex trajectory generation program

- Your target is to formulate a trajectory generation problem into **Disciplined convex optimization programs** in P.41.
- Many off-the-shelf can help you solve them.
- Choose a proper solver according to your requirement.

## CVX

<http://cvxr.com/cvx/>

Matlab wrapper. Let you write down the convex program like mathematical equations, then call other solvers to solve the problem.

## Mosek

<https://www.mosek.com/>

Very robust convex solvers, can solve almost all kinds of convex programs. Can apply free academic license. Only library available (x86).

## OOQP

<http://pages.cs.wisc.edu/~swright/ooqp/>

Very fast, robust QP solver. Open sourced.

## GLPK

<https://www.gnu.org/software/glpk/>

Very fast, robust LP solver. Open sourced.

# Numerical Stability



# Normalization

- Time normalization
  - Some very small time durations may break the generation entirely.
  - Scale short time durations to a normal number (1.0).
  - Or adding scale factor to all piece of the curve.

$$f(t) = \begin{cases} \sum_{i=0}^N p_{1,i} \left( \frac{t - T_0}{T_1 - T_0} \right)^i & T_0 \leq t \leq T_1 \\ \sum_{i=0}^N p_{2,i} \left( \frac{t - T_1}{T_2 - T_1} \right)^i & T_1 \leq t \leq T_2 \\ \vdots & \vdots \\ \sum_{i=0}^N p_{M,i} \left( \frac{t - T_{M-1}}{T_M - T_{M-1}} \right)^i & T_{M-1} \leq t \leq T_M \end{cases}$$

Use relative timeline!

- Problem scale (spatial) normalization
  - If the problem is underlying for large-scale scene.
  - Such as waypoints with  $x = 100.0 \text{ m}$
  - Consider solve a tiny problem (a sandbox), and re-scale the solution back.

These two operations highly increase the numerical stability in practice.





# Other engineering stuff

## 1. Solve 3 axis independently, or together?

$$\min \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}$$
$$\text{s. t. } \mathbf{A}_{eq} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \mathbf{d}_{eq}$$

## 2. Is closed-form solution always better?

- When matrix operation is expansive, the numerical convex solver is much more robust.
- Modern solver (Mosek) has pretty good stability.

## 3. Is polynomial can do anything?

- Almost, but not all.
- One can prove that the polynomial function (with several order) is the best solution for minimize single squared control input. But when things come to it's not (out of this course's scope).



$$J = \int_0^T \rho_1 \cdot \text{jerk}^2(t) + \rho_1 \cdot \text{snap}^2(t) dt.$$

- Typically, solving 3 small-scaled problems is better (stable, faster) than 1 large-scaled problem.
- Coupled generation may add different weighting into 3 axis.

### Efficiency comparison

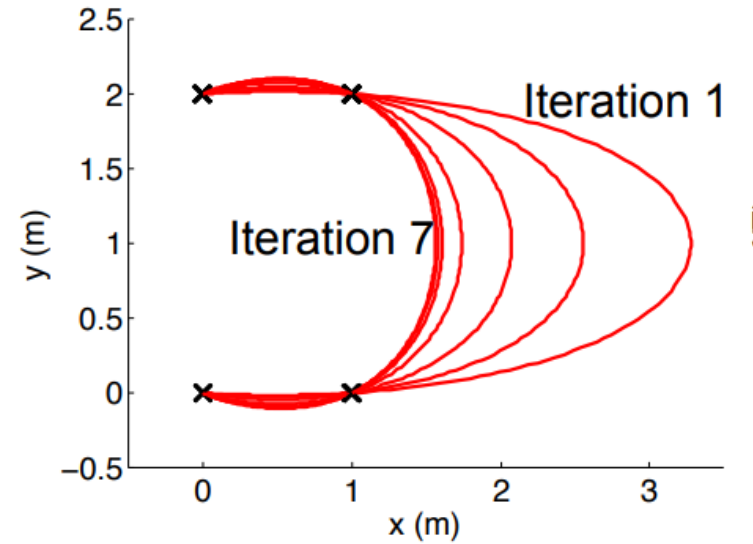
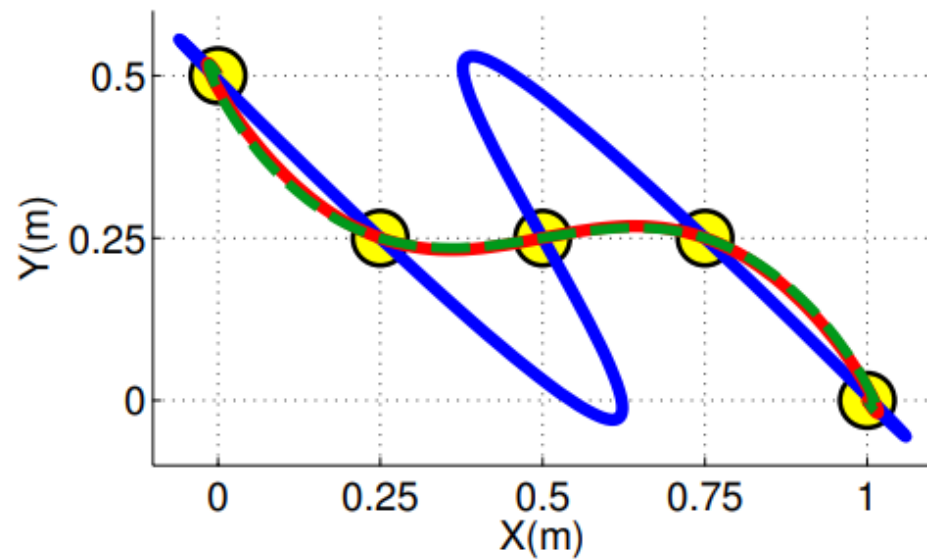
Benchmark Problem: 3-Segment Joint Optimization	
Method	Solution Time (ms)
MATLAB <code>quadprog.m</code>	9.5
MATLAB Constrained	1.7
MATLAB Unconstrained (Dense)	2.7
C++/Eigen Constrained	0.18
<b>C++/Eigen Unconstrained (Dense)</b>	<b>0.34</b>

# Time Allocation



## Problem Definition

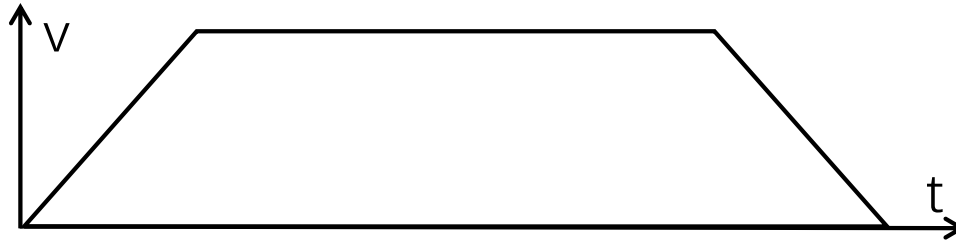
- Piecewise trajectory depends on a piecewise time allocation.
- Time allocation significantly affect the final trajectory.
- How to get a proper time allocation?





# Naïve solution

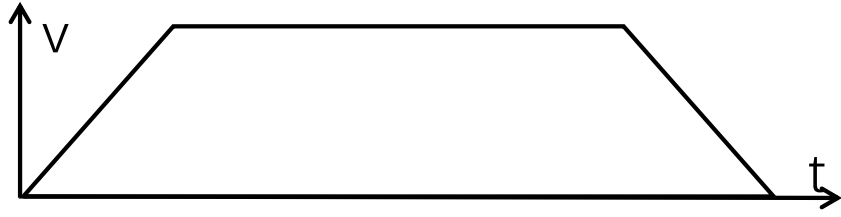
- Use “trapezoidal velocity” time profile to get time durations for each pieces.
  - Assume in every piece, accelerate to max. velocity -> de-accelerate to 0.
  - Accelerate + max. velocity + de-accelerate.
- Use expected average velocity to get time durations for each pieces.



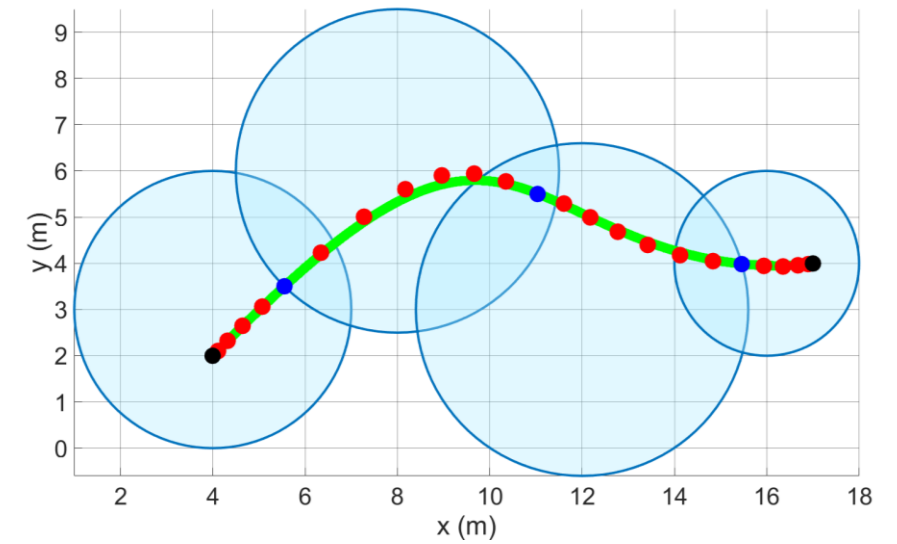
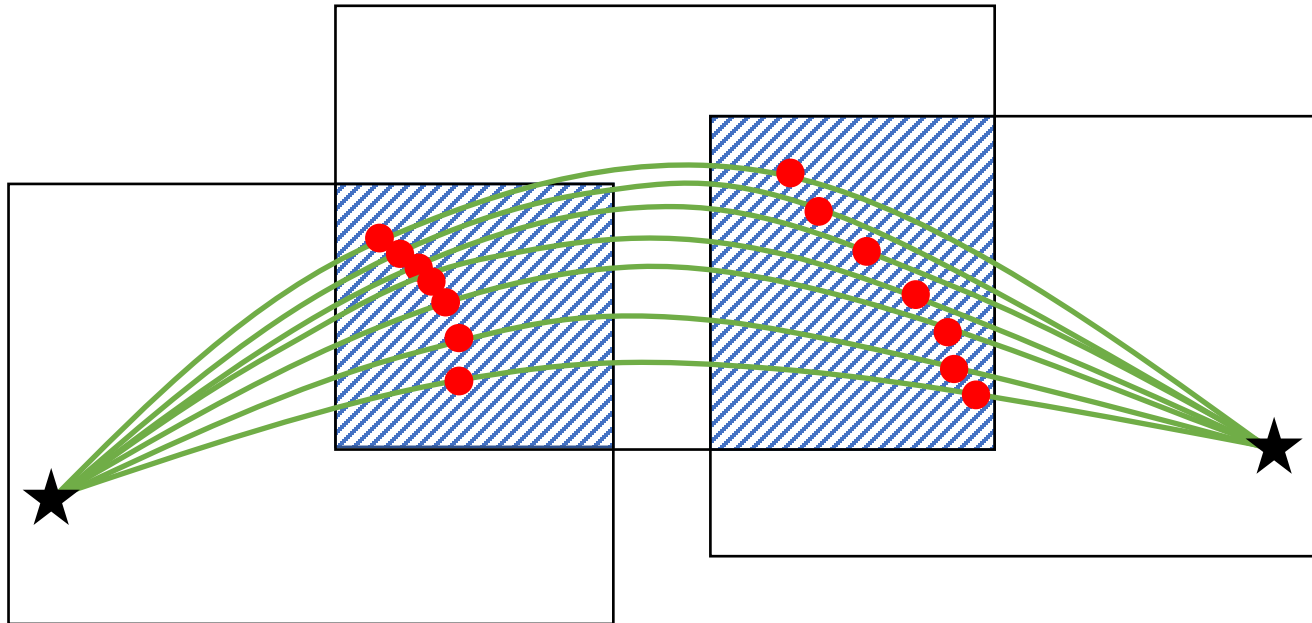
- Far away from optimal.
- Only get conservative time profile.
- Has no reaction to environmental styles



# Naïve solution



- Looks stupid for one piece.
- Works fine in corridor based trajectory generation.
- Overlapping regions in corridor provides large auto-adjustment space.



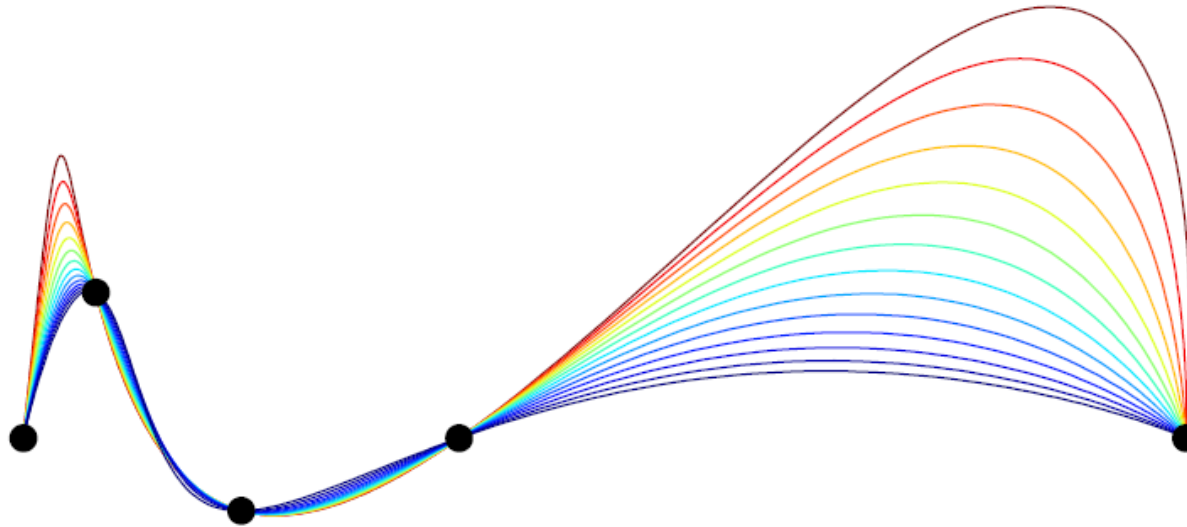


## Iterative numerical solution

$$J_T = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & \\ & \ddots & \\ & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T + k_T \sum_{i=1}^M T_i$$

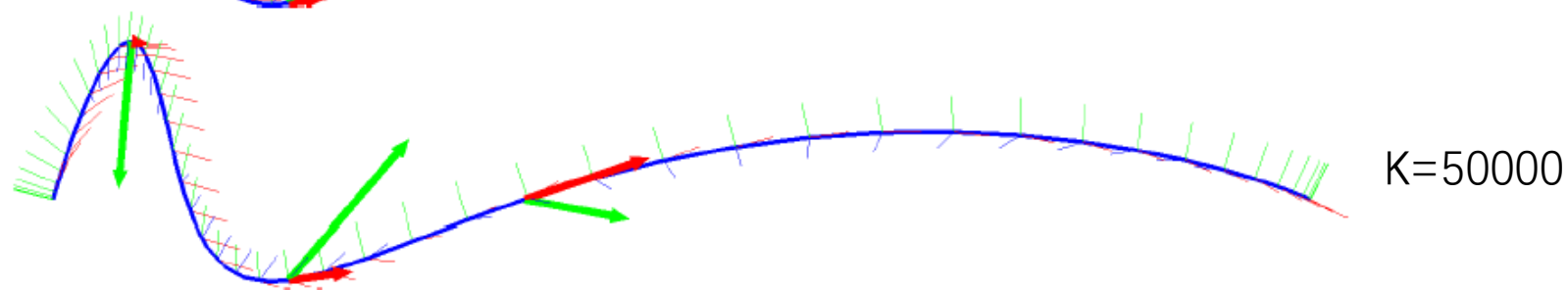
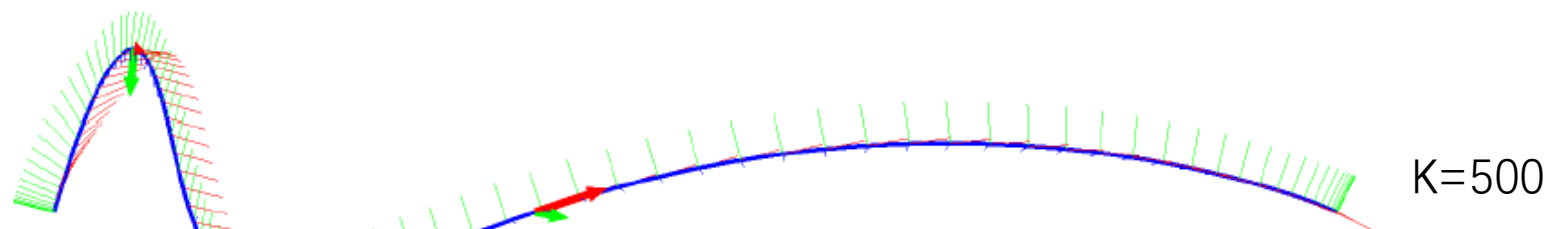
Penalize time duration in the overall cost

- Minimize this objective function **J**
- Get the gradient to  $T$  numerically





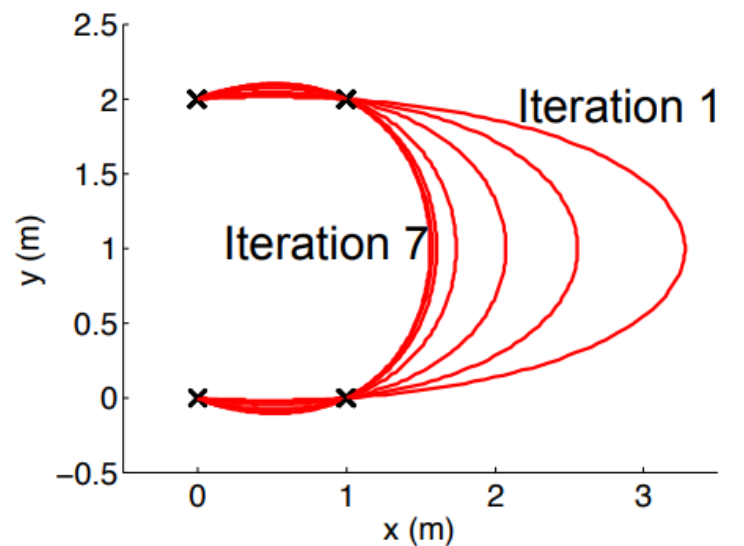
# Iterative numerical solution



- Varying segment time ratio
- Varying total time

## Dynamic limit?

- Unconstrained, find the best time allocation, fix the ratio.
- scale the total trajectory time preserving the optimal ratio.
- until a constraint becomes active.



- Varying segment time ration
- Fixed total time



# Homework





## Implement minimum snap in Matlab or C++/ROS

Homework 1.1: In matlab, use the 'quadprog' QP solver, write down a minimum snap trajectory generator

Homework 1.2: In matlab, generate minimum snap trajectory based on the closed form solution

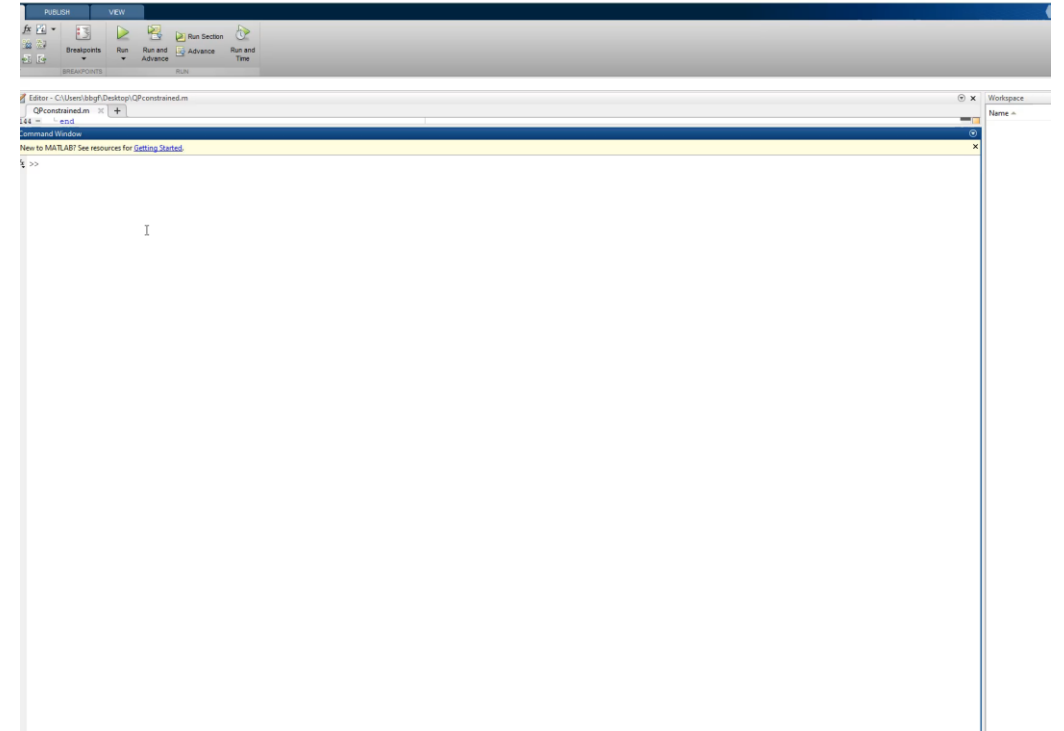
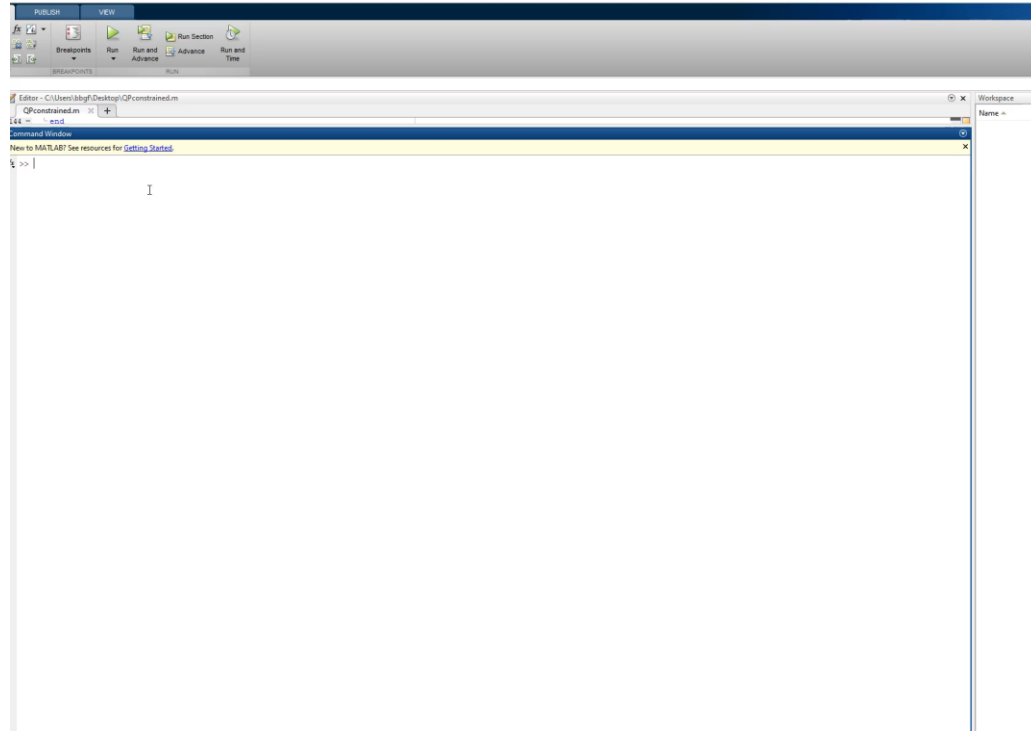
Homework 2.1: In C++/ROS, use the OOQP solver, write down a minimum snap trajectory generator

Homework 2.2: In C++/ROS, use Eigen, generate minimum snap trajectory based on the closed form solution

- Choose one homework. I suggest the first one.
- I highly suggest you finish all these options, since they are more than fundamental in planning.

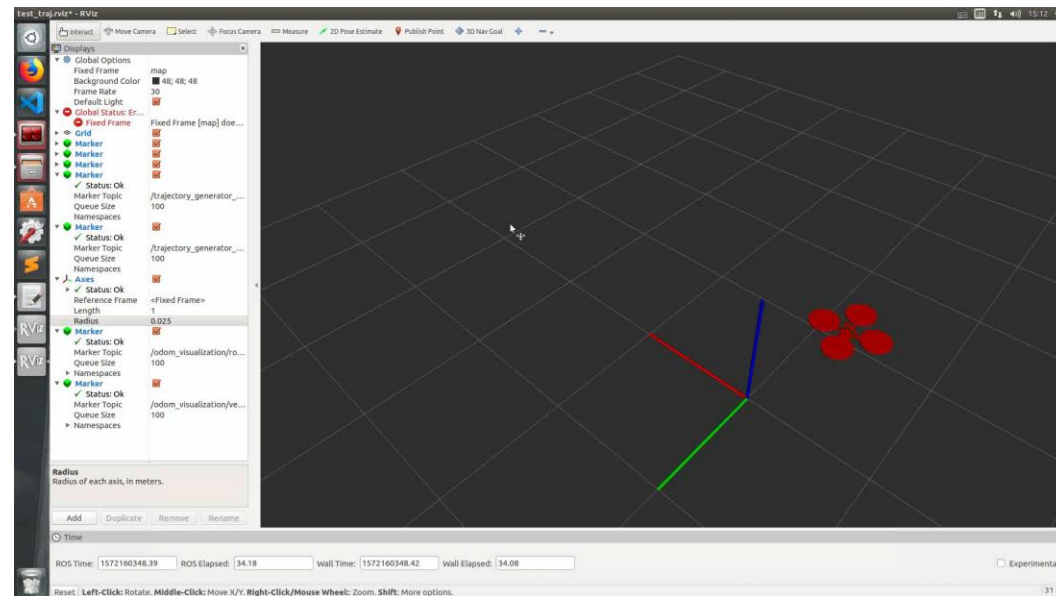
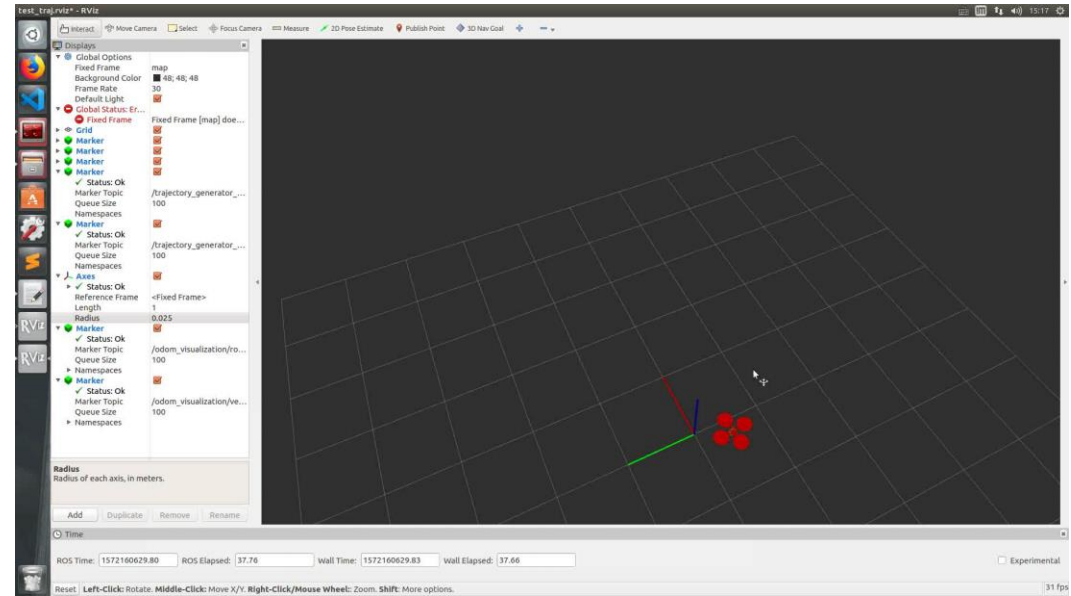
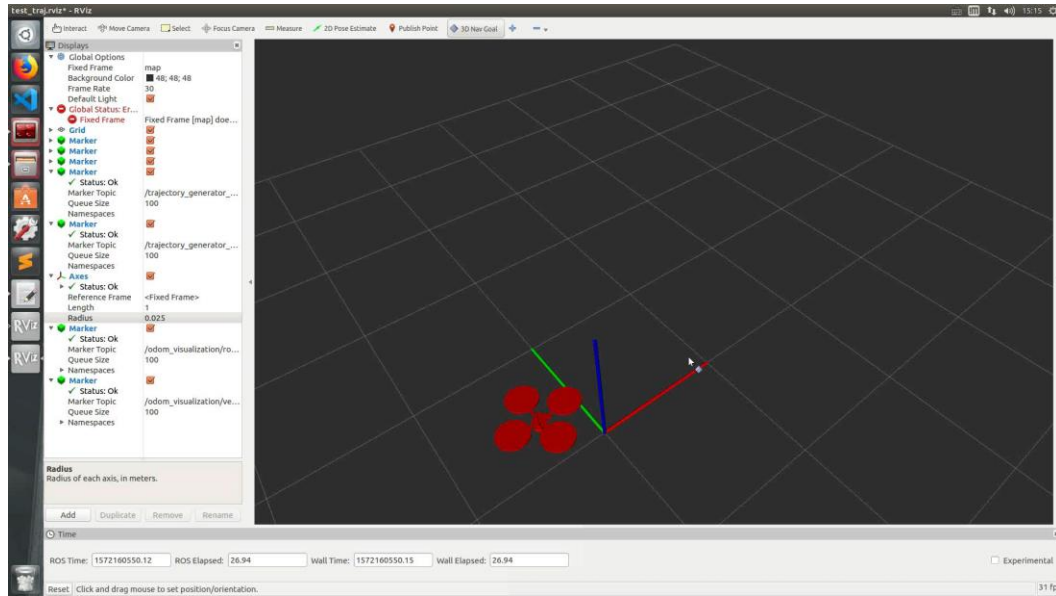


# Matlab homework expectation





# C++/ROS homework expectation





**Thanks for Listening!**