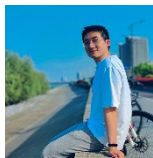


Classic Local Planning Frameworks for Mobile Robots

■ Lecture 8



主讲人 Qianhao Wang

Ph.D. Candidate in Robotics
Zhejiang University

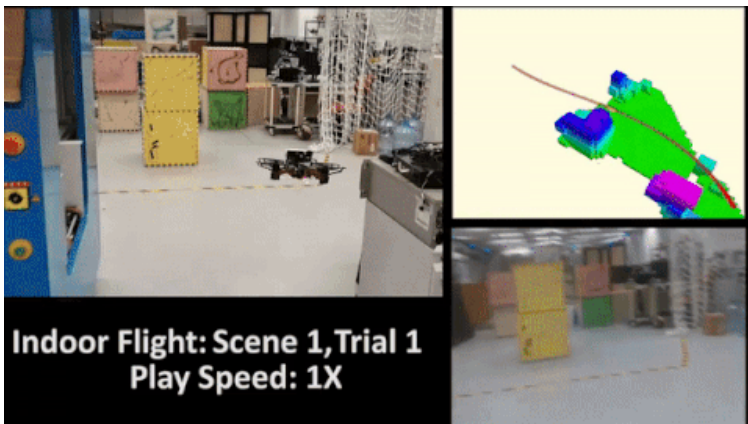


Fast-Planner^[1]

[1] Zhou B, Gao F, Wang L, et al. Robust and efficient quadrotor trajectory generation for fast autonomous flight[J]. IEEE RA-L, 2019.

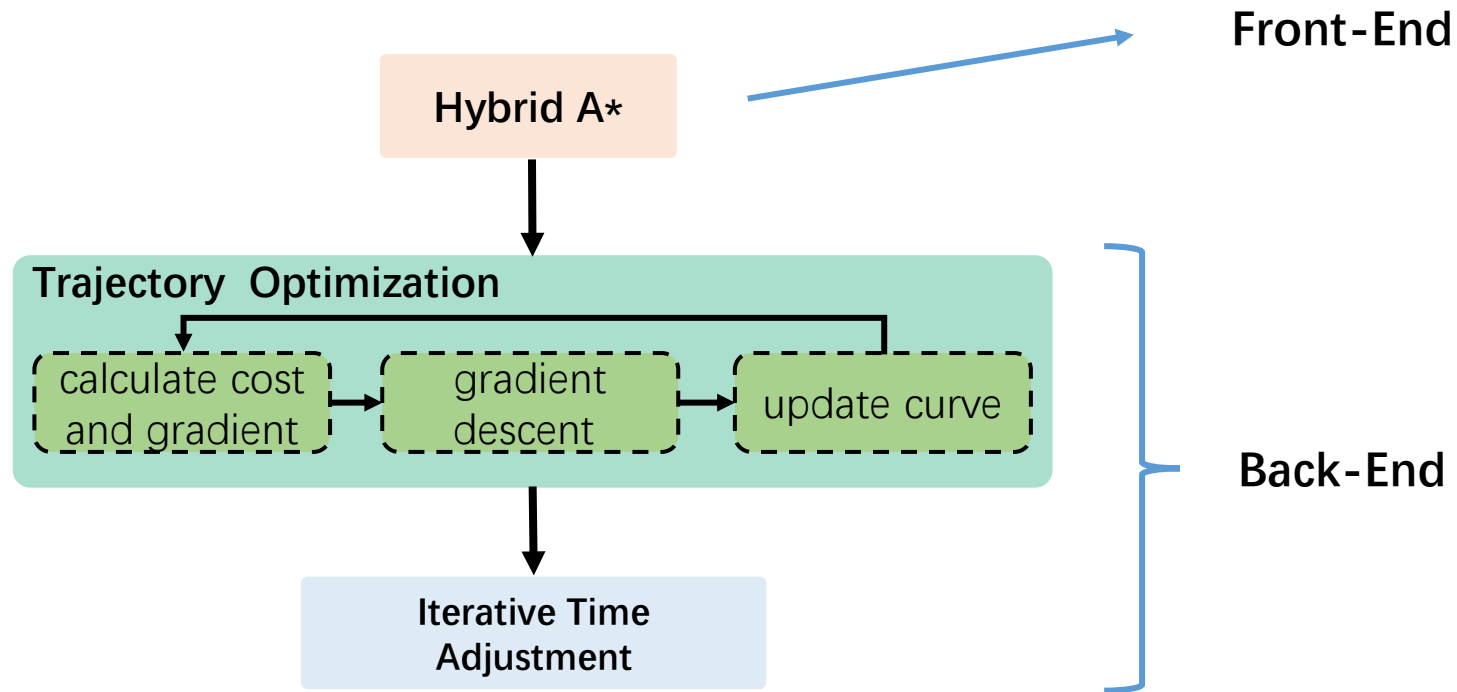


Fast-Planner





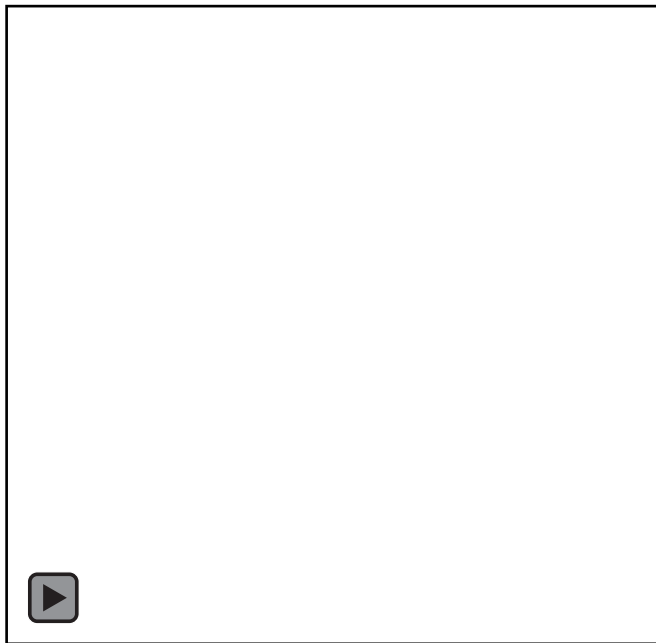
Fast-Planner



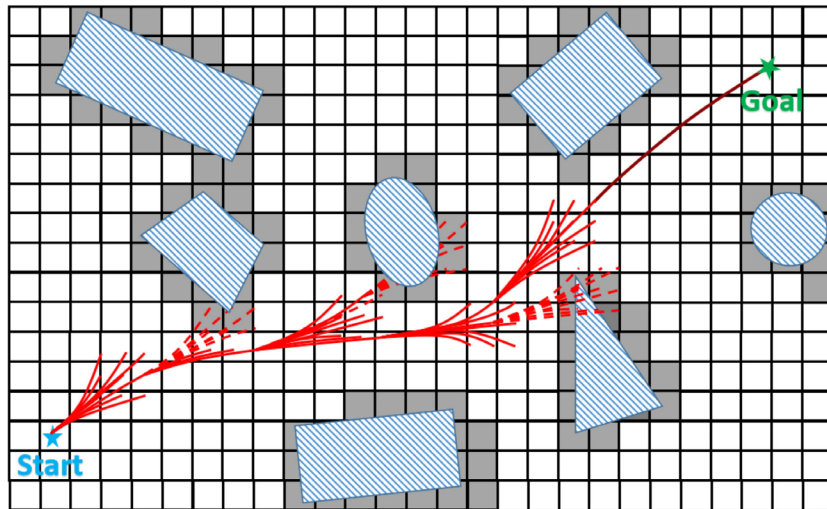


Front-End

A* vs Hybrid A*



A*



Hybrid A*



Front-End

A* vs Hybrid A*

Loop

- If the queue is empty, return FALSE; break;
- Remove the node "n" with the lowest $f(n)=g(n)+h(n)$ from the priority queue
- Mark node "n" as **expanded**
- If the node "n" is the goal state, return TRUE; break;
- For all **unexpanded** neighbors "m" of node "n"
 - If $g(m) = \text{infinite}$
 - $g(m) = g(n) + C_{nm}$
 - Push node "m" into the queue
 - If $g(m) > g(n) + C_{nm}$
 - $g(m) = g(n) + C_{nm}$
- end

End Loop

A*

Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}(), \mathcal{C}.\text{insert}(n_c)$ ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return RetrievePath();
6    $\text{primitives} \leftarrow \text{Expand}(n_c)$ ;
7    $\text{nodes} \leftarrow \text{Prune}(\text{primitives})$ ;
8   for  $n_i$  in  $\text{nodes}$  do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{\text{temp}} \leftarrow n_c.g_c + \text{EdgeCost}(n_i)$ ;
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i)$ ;
13      else if  $g_{\text{temp}} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c, n_i.g_c \leftarrow g_{\text{temp}}$ ;
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i)$ ;

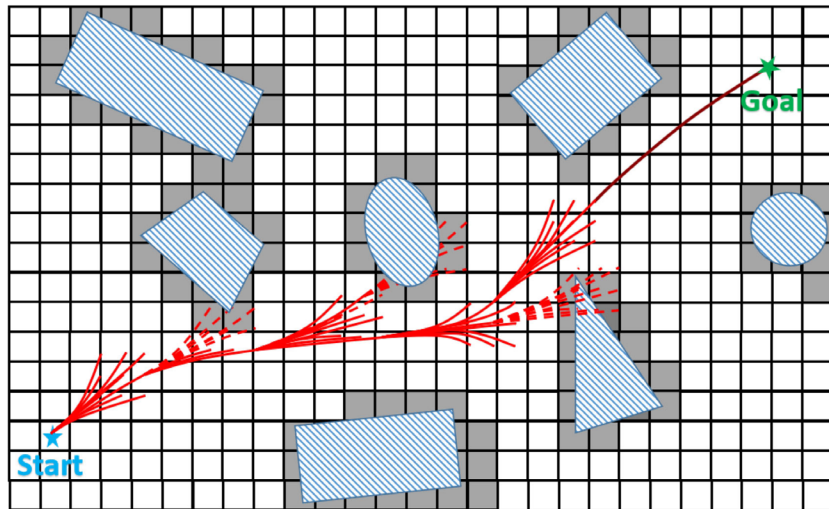
```

Hybrid A*



Front-End

Kinodynamic Path Searching: Hybrid A*



Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}()$ ,  $\mathcal{C}.\text{insert}(n_c)$  ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return RetrievePath();
6   primitives  $\leftarrow$  Expand( $n_c$ );
7   nodes  $\leftarrow$  Prune(primitives);
8   for  $n_i$  in nodes do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{\text{temp}} \leftarrow n_c.g_c + \text{EdgeCost}(n_i)$  ;
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i)$ ;
13      else if  $g_{\text{temp}} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c$ ,  $n_i.g_c \leftarrow g_{\text{temp}}$ ;
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i)$ ;

```



Front-End

Hybrid A*: Primitives Generation

- represent the trajectory by three independent 1-D time-parameterized polynomial functions

$$\mathbf{p}(t) := [p_x(t), p_y(t), p_z(t)]^T$$

For example, $K=2$, $\mu = x$

$$p_\mu(t) = \sum_{k=0}^K a_k t^k, \mu \in \{x, y, z\}$$

$$p_x(t) = a_0 + a_1 t + a_2 t^2$$

- From the view of quadrotor systems, it corresponds to a linear time-invariant (LTI) system.

state vector: $\mathbf{x}(t) := [\mathbf{p}(t)^T, \dot{\mathbf{p}}(t)^T, \dots, \mathbf{p}^{(n-1)}(t)^T]^T \in \mathbb{R}^{3n}$

control input: $\mathbf{u}(t) := \mathbf{p}^{(n)}(t) \in \mathcal{U} := [-u_{max}, u_{max}]^3 \subset \mathbb{R}^3$



Front-End

Hybrid A*: Primitives Generation

state vector: $\mathbf{x}(t) := [\mathbf{p}(t)^T, \mathbf{p}'(t)^T, \dots, \mathbf{p}^{(n-1)}(t)^T]^T \in \mathbb{R}^{3n}$

control input: $\mathbf{u}(t) := \mathbf{p}^{(n)}(t) \in \mathcal{U} := [-u_{max}, u_{max}]^3 \subset \mathbb{R}^3$

- The state space model can be defined as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_3 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_3 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{I}_3 \\ \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{0} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{I}_3 \end{bmatrix}$$



- The state equation : $\mathbf{x}(t) = e^{\mathbf{A}t} \boxed{\mathbf{x}(0)} + \int_0^t e^{\mathbf{A}(t-\tau)} \boxed{\mathbf{B}\mathbf{u}(\tau)} d\tau$

initial state

control input



Front-End

Hybrid A*: Primitives Generation

- The state equation : $\mathbf{x}(t) = e^{At} \boxed{\mathbf{x}(0)} + \int_0^t e^{A(t-\tau)} \boxed{\mathbf{B}\mathbf{u}(\tau)} d\tau$

initial state

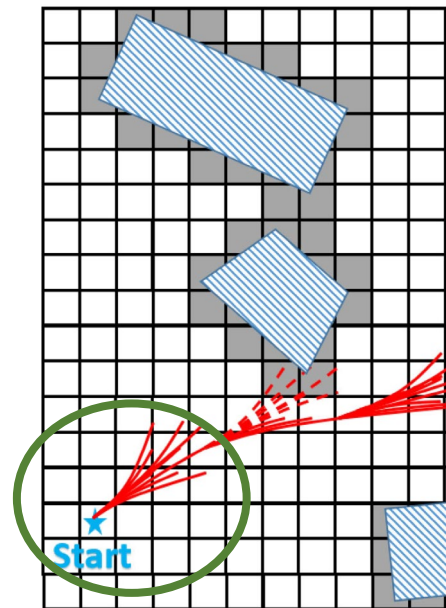
control input



- Each axis $[-u_{max}, u_{max}]$ is discretized uniformly as

$$\left\{ -u_{max}, -\frac{r-1}{r}u_{max}, -\frac{r-2}{r}u_{max}, \dots, \frac{r-2}{r}u_{max}, \frac{r-1}{r}u_{max}, u_{max} \right\}$$

which results in $(2r + 1)^3$ primitives.





Front-End

Hybrid A*: Primitives Generation

- The state equation : $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau) d\tau$

state vector: $\mathbf{x}(t) := [\mathbf{p}(t)^T, \dot{\mathbf{p}}(t)^T, \dots, \mathbf{p}^{(n-1)}(t)^T]^T \in \mathbb{R}^{3n}$

control input: $\mathbf{u}(t) := \mathbf{p}^{(n)}(t) \in \mathcal{U} := [-u_{max}, u_{max}]^3 \subset \mathbb{R}^3$

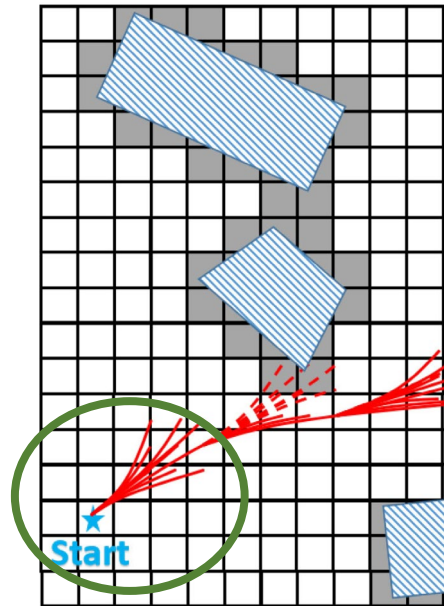
- In Fast-Planner

state vector:

$$\mathbf{x}(t) := [\mathbf{p}(t)^T, \dot{\mathbf{p}}(t)^T]^T = [p_x(t), p_y(t), p_z(t), v_x(t), v_y(t), v_z(t)]^T \in \mathbb{R}^6$$

control input:

$$\mathbf{u}(t) := \mathbf{p}^{(2)}(t) = [a_x(t), a_y(t), a_z(t)]^T \in \mathbb{R}^3$$

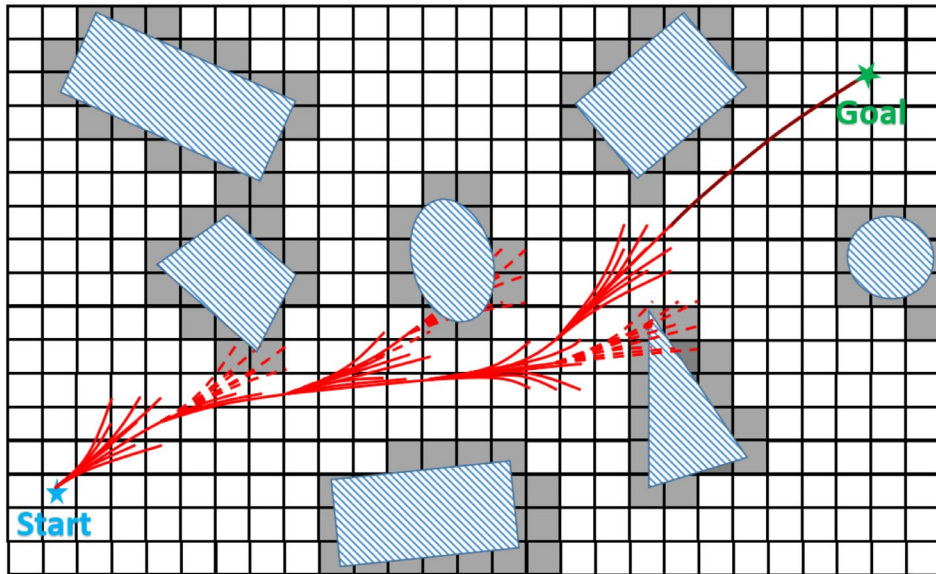




Front-End

The state equation : $\mathbf{x}(t) = e^{At}\mathbf{x}(0) + \int_0^t e^{A(t-\tau)}\mathbf{B}\mathbf{u}(\tau) d\tau$

Hybrid A*: Actual Cost



actual cost of one primitive : $e_c = (\|\mathbf{u}_d\|^2 + \rho)\tau$

actual cost of one trajectory consists of J primitives:

$$g_c = \sum_{j=1}^J \left(\|(\mathbf{u}_d)_j\|^2 + \rho \right) \tau$$

Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}()$ ,  $\mathcal{C}.\text{insert}(n_c)$ ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return RetrievePath();
6   primitives  $\leftarrow \text{Expand}(n_c)$ ;
7   nodes  $\leftarrow \text{Prune}(primitives)$ ;
8   for  $n_i$  in nodes do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{temp} \leftarrow n_c.g_c + \text{EdgeCost}(n_i)$ ;
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i)$ ;
13      else if  $g_{temp} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c$ ;
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i)$ ;

```

actual cost of an optimal trajectory from the start state to the current state



Front-End

Hybrid A*: Heuristic Cost

- We compute a closed form trajectory that minimizes $\mathcal{T}(T)$ from $\mathbf{x}_{\mu c}$ to the goal state $\mathbf{x}_{\mu g}$ by applying the Pontryagin's minimum principle:

$$p_{\mu}^*(t) = \frac{1}{6}\alpha_{\mu}t^3 + \frac{1}{2}\beta_{\mu}t^2 + v_{\mu c}t + p_{\mu c}$$

$$\begin{bmatrix} \alpha_{\mu} \\ \beta_{\mu} \end{bmatrix} = \frac{1}{T^3} \begin{bmatrix} -12 & 6T \\ 6T & -2T^2 \end{bmatrix} \begin{bmatrix} p_{\mu g} - p_{\mu c} - v_{\mu c}T \\ v_{\mu g} - v_{\mu c} \end{bmatrix}$$

- $p_{\mu c}, v_{\mu c}, p_{\mu g}, v_{\mu g}$ are the current and goal position and velocity.

$$a_{\mu}^*(t) = \alpha_{\mu}t + \beta_{\mu}$$

$$\mathbf{u}(t) := [a_x(t), a_y(t), a_z(t)]^T$$

$$\begin{aligned} \mathcal{T}^*(T) &= \int_0^T \|\mathbf{u}(t)\|^2 dt + \rho T \\ &= \sum_{\mu \in \{x, y, z\}} \left(\frac{1}{3}\alpha_{\mu}^2 T^3 + \frac{1}{2}\alpha_{\mu}\beta_{\mu}T^2 + \beta_{\mu}^2 T \right) + \rho T \end{aligned}$$

$$p(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

$$p(0) = p_{\mu c}, \quad \dot{p}(0) = v_{\mu c}$$

$$p(T) = p_{\mu g}, \quad \dot{p}(T) = v_{\mu g}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ T^3 & T^2 & T & 1 \\ 3T^2 & 2T & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} p_{\mu c} \\ v_{\mu c} \\ p_{\mu g} \\ v_{\mu g} \end{bmatrix}$$

Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}()$ ,  $\mathcal{C}.\text{insert}(n_c)$ ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return RetrievePath();
6    $\text{primitives} \leftarrow \text{Expand}(n_c)$ ;
7    $\text{nodes} \leftarrow \text{Prune}(\text{primitives})$ ;
8   for  $n_i$  in nodes do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{\text{temp}} \leftarrow n_c.g_c + \text{EdgeCost}(n_i)$ ;
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i)$ ;
13      else if  $g_{\text{temp}} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c$ ,  $n_i.g_c \leftarrow g_{\text{temp}}$ ;
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i)$ ;

```



Front-End

Hybrid A*: Heuristic Cost

$$\begin{bmatrix} \alpha_\mu \\ \beta_\mu \end{bmatrix} = \frac{1}{T^3} \begin{bmatrix} -12 & 6T \\ 6T & -2T^2 \end{bmatrix} \begin{bmatrix} p_{\mu g} - p_{\mu c} - v_{\mu c} T \\ v_{\mu g} - v_{\mu c} \end{bmatrix}$$

$$\mathcal{J}^*(T) = \sum_{\mu \in \{x, y, z\}} \left(\frac{1}{3} \alpha_\mu^2 T^3 + \frac{1}{2} \alpha_\mu \beta_\mu T^2 + \beta_\mu^2 T \right) + \rho T$$

- To find the optimal time T that minimize the cost, we substitute α_μ, β_μ into $\mathcal{J}^*(T)$ and find the roots of

$$\frac{\partial \mathcal{J}^*(T)}{\partial T} = 0$$

- The root making a minimum cost $\min \mathcal{J}^*$ and feasible trajectory is selected and denoted as T_h .



Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}(), \mathcal{C}.\text{insert}(n_c)$  ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return  $\text{RetrievePath}()$ ;
6    $\text{primitives} \leftarrow \text{Expand}(n_c)$ ;
7    $\text{nodes} \leftarrow \text{Prune}(\text{primitives})$ ;
8   for  $n_i$  in  $\text{nodes}$  do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{\text{temp}} \leftarrow n_c.g_c + \text{EdgeCost}(n_i)$  ;
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i)$ ;
13      else if  $g_{\text{temp}} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c, n_i.g_c \leftarrow g_{\text{temp}}$ ;
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i)$ ;

```

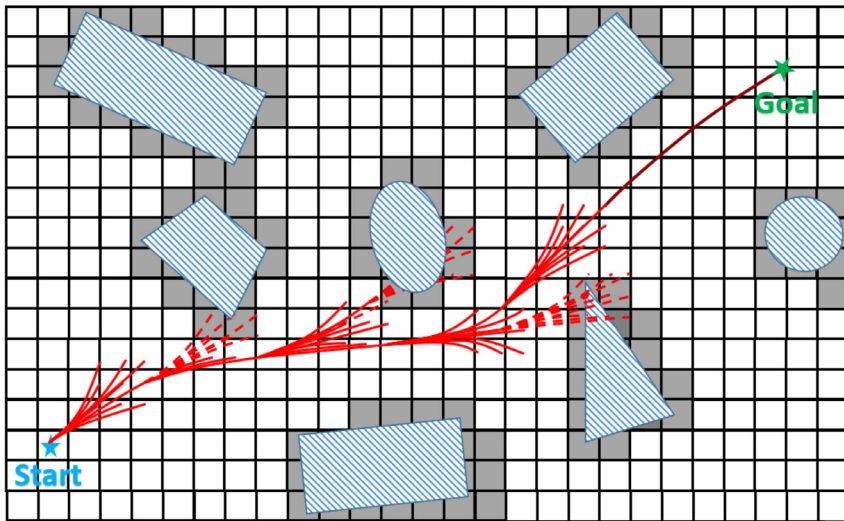


We use $\mathcal{J}^*(T_h)$ as the heuristic.



Front-End

Kinodynamic Path Searching: Hybrid A*



Algorithm 1: Kinodynamic Path Searching.

```

1 Initialize();
2 while  $\neg \mathcal{P}.\text{empty}()$  do
3    $n_c \leftarrow \mathcal{P}.\text{pop}()$ ,  $\mathcal{C}.\text{insert}(n_c)$  ;
4   if  $\text{ReachGoal}(n_c) \vee \text{AnalyticExpand}(n_c)$  then
5     return  $\text{RetrievePath}()$ ;
6    $\text{primitives} \leftarrow \text{Expand}(n_c);$ 
7    $\text{nodes} \leftarrow \text{Prune}(\text{primitives});$ 
8   for  $n_i$  in nodes do
9     if  $\neg \mathcal{C}.\text{contain}(n_i) \wedge \text{CheckFeasible}(n_i)$  then
10       $g_{\text{temp}} \leftarrow n_c.g_c + \text{EdgeCost}(n_i);$ 
11      if  $\neg \mathcal{P}.\text{contain}(n_i)$  then
12         $\mathcal{P}.\text{add}(n_i);$ 
13      else if  $g_{\text{temp}} \geq n_i.g_c$  then
14        continue;
15       $n_i.\text{parent} \leftarrow n_c$ ;  $n_i.g_c \leftarrow g_{\text{temp}};$ 
16       $n_i.f_c \leftarrow n_i.g_c + \text{Heuristic}(n_i);$ 

```
