凸优化与自主轨迹规划 大作业

姓名	学号
杨思远	3120210063

> 学习以下论文:
>
> Guoxu Zhang and Xinfu Liu, “UAV Collision Avoidance Using Mixed-Integer Second-Order Cone Programming,” Journal of Guidance, Control, and Dynamics, 2022.
>
> 1. 详细写出针对论文中原始的无人机避障轨迹优化问题的凸化处理过程(利用课堂所学解释各个约束为什么为凸约束)与离散化过程，将其转换为混合整数二阶锥优化问题。(10 分)
> 2. 列出论文中算法 1 和算法 2 的算法框架，并解释算法 1 和算法 2 的区别。(10分)
> 3. 自行设置无禁飞区的多个飞行任务(如给定始末位置与始末航向角)，利用算法 1 和算法 2 进行求解(求解器可用 MOSEK/ECOS)，分析数值结果。(25分)
> 4. 自行设置含禁飞区的多个飞行任务(如 1-3 个禁飞区)，利用算法 1 和算法 2 进行求解，分析数值结果。(30 分)
> 5. 自行设置飞行任务，利用论文中的方法与其它方法(如 GPOPS 软件，SNOPT求解器)进行求解，分析比较不同方法得到的解，对比不同方法的计算效率与可靠性等。(15 分)
> 6. 论文中的假设 1 要求航向角在-90°与 90°之间，若去除该假设(即航向角可以为 0 到 360°之间任意角度)，且不考虑禁飞区约束，分析如何对原始 的无人机轨迹优化问题进行凸化处理，写出凸化处理的过程(对是否给出数 值计算结果不做要求)。(10 分)

任务1

原始的无人机避障轨迹优化中，约束分别为

1. 无人机动力学约束

$$\begin{cases} \dot{x} = V\cos\theta \\ \dot{y} = V\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{1}$$

2. 控制限幅约束

$$|\omega| \leq \omega_{\max} \tag{2}$$

3. 避障约束

$$(x, y) \in X \backslash \cup_{j=1}^{M} X_{o,j} \tag{3}$$

4. 终端约束

$$x(t_f) = x_f, y(t_f) = y_f, \theta(t_f) = \theta_f \tag{4}$$

优化问题为

$$\begin{aligned}\text{Problem } \mathcal{O}: \min J &= \int_0^{t_f} 1\,\mathrm{d}t \\ \text{s. t. Eqs.}&(1),(2),(3),(4)\end{aligned}$$

假设无人机一直保持从左到右运动，即航向角 $\theta \in (-\pi/2, \pi/2)$。这可以保证无人机的横坐标 x 单调递增，从而无人机的微分方程状态变量可以更换为 x，(1) 式即变形为

$$\begin{cases} y' = \tan\theta \\ \theta' = \dfrac{\omega}{V\cos\theta} \end{cases} \tag{5}$$

其中，$(\cdot)'$ 表示对变量 x 求导数。这样，优化问题中就消去了变量 x。

在 (5) 式中，y 状态量的方程依然非线性，采用变量代换 $\vartheta := \tan\theta$。则 (5) 式可以写为

$$\begin{cases} y' = \vartheta \\ \vartheta' = \sqrt{1+\vartheta^2}^3\omega/V \end{cases} \tag{6}$$

其中用到了 $1/\cos\theta = \sqrt{1+\tan^2\theta} = \sqrt{1+\vartheta^2}$ 及 $\vartheta' = (\tan\theta)' = \theta'/\cos^2\theta$。

同时目标函数也变形为

$$J = \frac{1}{V}\int_{x_0}^{x_f} \sqrt{1+\vartheta^2}\,\mathrm{d}x \tag{7}$$

在 (6) 式中，ϑ 状态量的方程依然非线性，新引入约束

$$\delta = \sqrt{1+\vartheta^2} \tag{8}$$

于是式 (6) 即为线性微分方程

$$\begin{cases} y' = \vartheta \\ \vartheta' = \delta^3\omega/V \end{cases} \tag{9}$$

目标函数即为

$$J = \frac{1}{V}\int_{x_0}^{x_f} \delta\,\mathrm{d}x \tag{10}$$

再做变量代换 $u := \delta^3\omega/V$，消掉变量 ω，则式 (9) 即为

$$\begin{cases} y' = \vartheta \\ \vartheta' = u \end{cases} \tag{11}$$

控制限幅约束式 (2) 中也出现了 ω，则可以写成

$$|u| \le \omega_{\max}\delta^3/V \tag{12}$$

另外，终端约束式 (4) 也可变形为

$$y(x_f) = y_f,\ \vartheta(x_f) = \vartheta_f \tag{13}$$

这里 $\vartheta_f = \tan\theta_f$。

由于 x 成为了新的状态量，则避障约束式 (3) 可以写成

$$y \geq \bar{l}_j(x) \text{ 或者 } y \leq \underline{l}_j(x),\ j = 1, \ldots, M \tag{14}$$

这里 $\bar{l}_j(x) = \max\{y|(x,y) \in \partial X_{o,j}\}$，$\underline{l}_j(x) = \min\{y|(x,y) \in \partial X_{o,j}\}$，$\partial X_{o,j}$ 代表集合 $X_{o,j}$ 的边界。由于在优化问题中难以表示"或者"关系，考虑引入布尔变量，将式 (14) 写为

$$y \geq \bar{l}_j(x) + D(\eta_j - 1), y \leq \underline{l}_j(x) + D\eta_j,\ j = 1, \ldots, M \tag{15}$$

其中 $\eta_j \in \{0, 1\}$ 是与第 j 个障碍相关的布尔类型变量，D 是个足够大的正常数。

至此，原优化问题 Problem $\mathcal{O}$ 等价地变形为

$$\begin{aligned} \text{Problem } \mathcal{E}: \min\ & J = \frac{1}{V}\int_{x_0}^{x_f} \delta \, \mathrm{d}x \\ \text{s.t. Eqs. } & (8),\ (11),\ (12),\ (13) \text{ and } (15) \end{aligned}$$

通过松弛，再将其中的一些约束凸化。将式 (8) 放缩为

$$\delta \geq \sqrt{1 + \vartheta^2} \tag{16}$$

则松弛后新的优化问题为

$$\begin{aligned} \text{Problem } \mathcal{R}: \min\ & J = \frac{1}{V}\int_{x_0}^{x_f} \delta \, \mathrm{d}x \\ \text{s.t. Eqs. } & (16),\ (11),\ (12),\ (13) \text{ and } (15) \end{aligned}$$

最后，将式 (12) 线性化为

$$|u| \leq \frac{\omega_{\max}}{V}[3(\delta^{(k)})^2\delta - 2(\delta^{(k)})^3] \tag{17}$$

其中 $\delta^{(k)}$ 是 k 次迭代的 δ 值。若离散化的时间点个数为 $(N+1)$，则优化变量包含 $\boldsymbol{y} = [y_0, y_1, \ldots, y_N]^T$，$\boldsymbol{\vartheta} = [\vartheta_0, \vartheta_1, \ldots, \vartheta_N]^T$，$\boldsymbol{u} = [u_0, u_1, \ldots, u_N]^T$，$\boldsymbol{\delta} = [\delta_0, \delta_1, \ldots, \delta_N]^T$，及 $\boldsymbol{\eta} = [\eta_0, \eta_1, \ldots, \eta_M]^T$。定义 $\boldsymbol{z} = [\boldsymbol{y}^T, \boldsymbol{\vartheta}^T, \boldsymbol{u}^T, \boldsymbol{\delta}^T]^T$。则最终优化问题可表述如下

$$\begin{aligned} \text{Problem } \mathcal{D}(\boldsymbol{\delta}^{(k)}): \min_{\boldsymbol{z},\boldsymbol{\eta}}\ & \boldsymbol{c}^T\boldsymbol{z} \\ \text{s.t. } & H\boldsymbol{z} \leq \boldsymbol{p} \\ & \Theta\boldsymbol{z} - \boldsymbol{b} \succeq_K 0 \\ & g_m(\boldsymbol{z}, \boldsymbol{\eta}) \leq 0, m = 1, \ldots, 2M \end{aligned}$$

其中线性约束来自式 (11), (13), (17)，二阶锥约束来自式 (16)，第三个约束来自式 (15)。它是一个混合整数二阶锥优化问题。

除了正式表述之外，面向优化程序设计，也可以将离散后的问题表述成如下形式：

$$\text{Problem } \mathcal{C}(\delta^{(k)}, N, x_0^*, y_0^*, \vartheta_0^*, x_N^*, y_N^*, \vartheta_N^*, V, \omega_{\max}, D, M, X_{o,1}, \dots, X_{o,M}):$$

$$\min_{\substack{y_0,y_1,\dots,y_N,\\ \vartheta_0,\vartheta_1,\dots,\vartheta_N,\\ u_0,u_1,\dots,u_N,\\ \delta_0,\delta_1,\dots,\delta_N,\\ x_0,x_1,\dots,x_N,\\ t_0,t_1,\dots,t_N,\\ \eta_0,\eta_1,\dots,\eta_N}} t_N \quad \text{s.t.} \begin{cases} y_0 = y_0^*, x_0 = x_0^*, \vartheta_0 = \vartheta_0^* \\ y_N = y_N^*, x_N = x_N^*, \vartheta_N = \vartheta_N^* \\ t_0 = 0 \\ \begin{cases} y_{i+1} = y_i + \vartheta_i \frac{x_N^* - x_0^*}{N} \\ \vartheta_{i+1} = \vartheta_i + u_i \frac{x_N^* - x_0^*}{N} \\ x_{i+1} = x_i + \frac{x_N^* - x_0^*}{N} \\ t_{i+1} = t_i + \frac{\delta_i}{V} \frac{x_N^* - x_0^*}{N} \end{cases}, i = 0, 1, \dots, N-1 \\ \begin{cases} u_i \le \frac{\omega_{\max}}{V} (3\delta_i^{(k)^2} \delta_i - 2\delta_i^{(k)^3}) \\ -u_i \ge \frac{\omega_{\max}}{V} (3\delta_i^{(k)^2} \delta_i - 2\delta_i^{(k)^3}) \end{cases}, i = 0, 1, \dots, N \\ \delta_i \ge \sqrt{1 + \vartheta_i^2}, \ i = 0, 1, \dots, N \\ \begin{cases} y_i \ge \bar{l}_j(x_i) + D(\eta_j - 1) \\ y_i \le \underline{l}_j(x_i) + D\eta_j \end{cases}, \\ \quad i = 0, 1, \dots, N, \ j = 1, \dots, M, \{y | (x_i, y) \in X_{o,j}\} \ne \emptyset \end{cases}$$

任务2

算法1的框架如下：

1. 设置 $k = 0$，并选择一个 $\boldsymbol{\delta}$ 的初始猜测值 $\boldsymbol{\delta}^{(0)}$；
2. 在第 $k+1$ 次迭代中，求解 $\text{Problem } \mathcal{D}(\boldsymbol{\delta}^{(k)})$ 得到 $\{\boldsymbol{z}^{(k+1)}; \boldsymbol{\eta}^{(k+1)}\}$；
3. 判断是否满足迭代收敛条件 $\max\{|\boldsymbol{\delta}^{(k+1)} - \boldsymbol{\delta}^{(k)}|\} \le \epsilon_\delta$，这里 ϵ_δ 是提前设置的误差收敛阈值；
    1. 如果满足，转到步骤 4；
    2. 否则，令 $k = k + 1$ 并且回到步骤 2；
4. 停止，最终解即为 $\{\boldsymbol{z}^{(k+1)}; \boldsymbol{\eta}^{(k+1)}\}$。

算法2的框架如下：

1. 选择一个坐标系，使得原点在无人机目前位置，并且 x 轴径直指向无人机目标位置。然后另 $\boldsymbol{\delta}^{(0)} = [1, 1, \dots, 1]^T$；
2. 求解 $\text{Problem } \mathcal{D}(\boldsymbol{\delta}^{(0)})$ 得到 $\{\boldsymbol{z}^{(1)}; \boldsymbol{\eta}^{(1)}\}$；
3. 停止，最终解即为 $\{\boldsymbol{z}^{(1)}; \boldsymbol{\eta}^{(1)}\}$。

算法1和算法2的区别在于：

- 算法1需要迭代，在其中有循环过程；而算法2无需迭代。
- 算法1可以任意指定 $\boldsymbol{\delta}$ 的初始值；而算法2需要指定特殊的初始值。

任务3

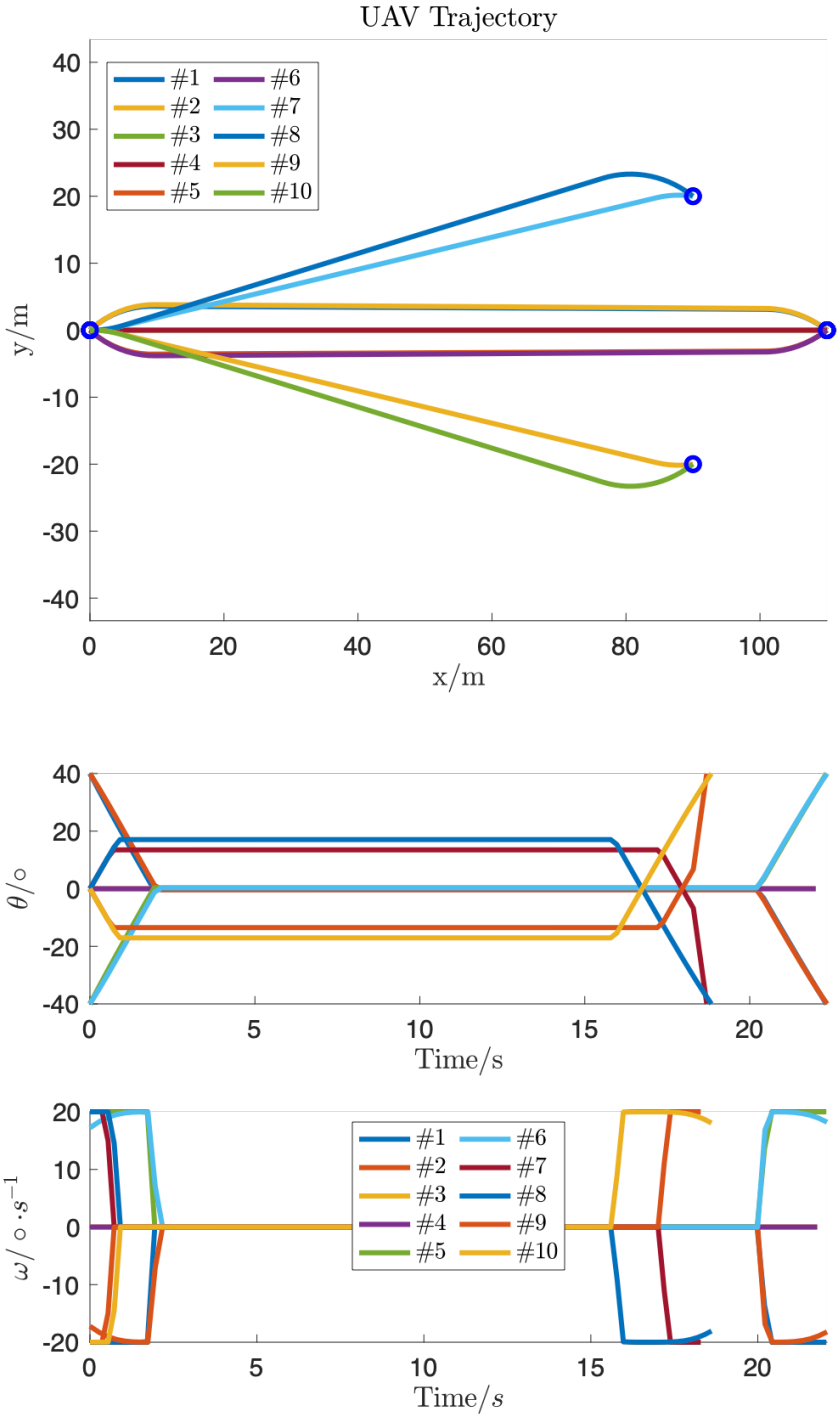
设置10个飞行任务为：

	初始位置 (x_0^*, y_0^*)	初始航向角 θ_0^*	终端位置 (x_N^*, y_N^*)	终端航向角 θ_N^*	采用何种算法
任务1	$(0,0)$	40°	$(110,0)$	-40°	算法1
任务2	$(0,0)$	40°	$(110,0)$	-40°	算法2
任务3	$(0,0)$	0°	$(110,0)$	0°	算法1
任务4	$(0,0)$	0°	$(110,0)$	0°	算法2
任务5	$(0,0)$	-40°	$(110,0)$	40°	算法1
任务6	$(0,0)$	-40°	$(110,0)$	40°	算法2
任务7	$(0,0)$	0°	$(90,20)$	-40°	算法1
任务8	$(0,0)$	0°	$(90,20)$	-40°	算法2
任务9	$(0,0)$	0°	$(90,-20)$	40°	算法1
任务10	$(0,0)$	0°	$(90,-20)$	40°	算法2

其余参数的取值为：

名称	离散区间个数 N	速度 $V/m\cdot s^{-1}$	航向角速度最大值 $\omega_{\max}/\mathrm{rad}\cdot s^{-1}$
值	100	5	20

运行程序，求解器选用 MOSEK，得到结果如下图所示。

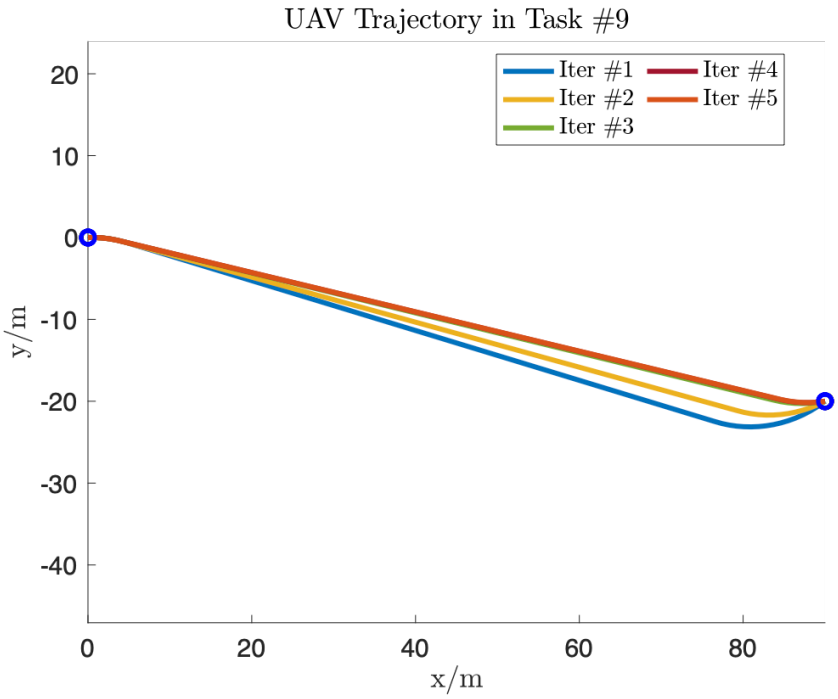


从结果中不难看出，对于10个飞行任务，都成功得到了符合要求的解。

其中，对于前6个飞行任务，即起点和终点的纵坐标相同的6个任务，算法1和算法2的执行结果几乎一致，线路在图中几乎重叠；而对于另外4个飞行任务，算法1和算法2的执行结果有较大不同，主要在于终点的航向角变化情况。算法2的执行结果相较于算法1，无人机在接近终点处更早转向，以满足终端航向角约束。

在飞行任务的求解结果中，无人机都只在起点与终点附近调整航向，在飞行的大部分过程中，都保持相同航向的飞行，无偏航角速度。不同飞行任务的结果中，算法1所给出的结果，无人机在初始以及末段的角速度均为限幅之内的最大值；然而算法2所给出的结果中，初始及末段的角速度略小于最大值，由此可见，即使算法2调整了变量的初始值以减小线性化的误差，迭代次数对于解的最优性的影响依然存在。

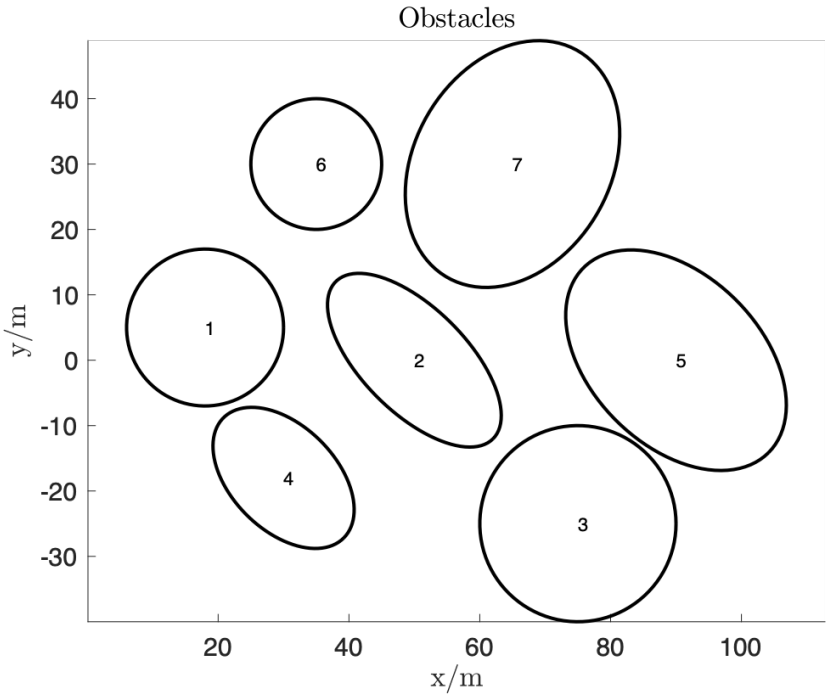
对于其中的任务9，每次迭代的轨迹结果如图所示。



从图中可以看出，迭代3次之后的轨迹几乎重合，而前2次的结果与其余结果均有较大差距，此结果印证了迭代次数对于解的最优性的影响的猜想。

任务4

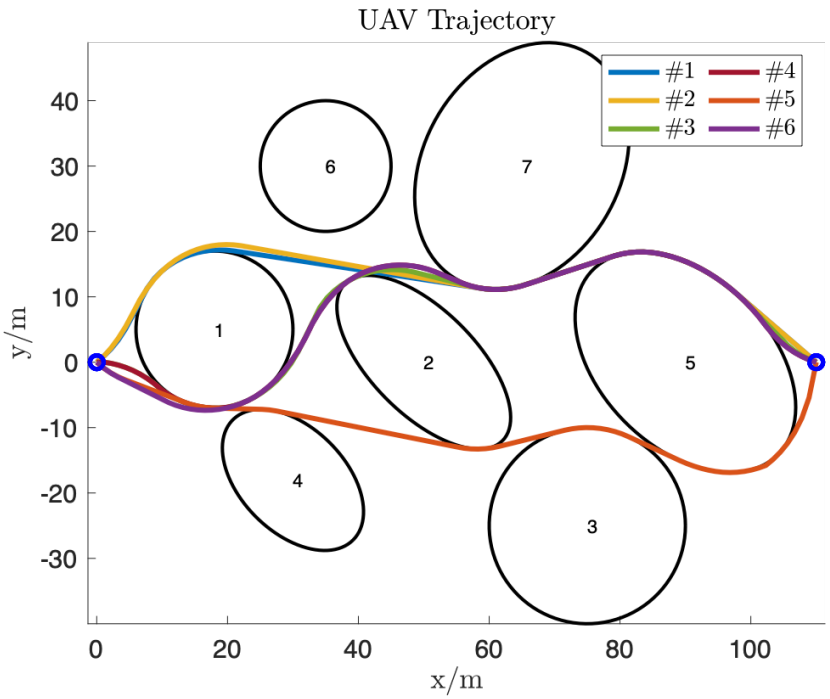
在任务区域内设置7个禁飞区，如下图所示。

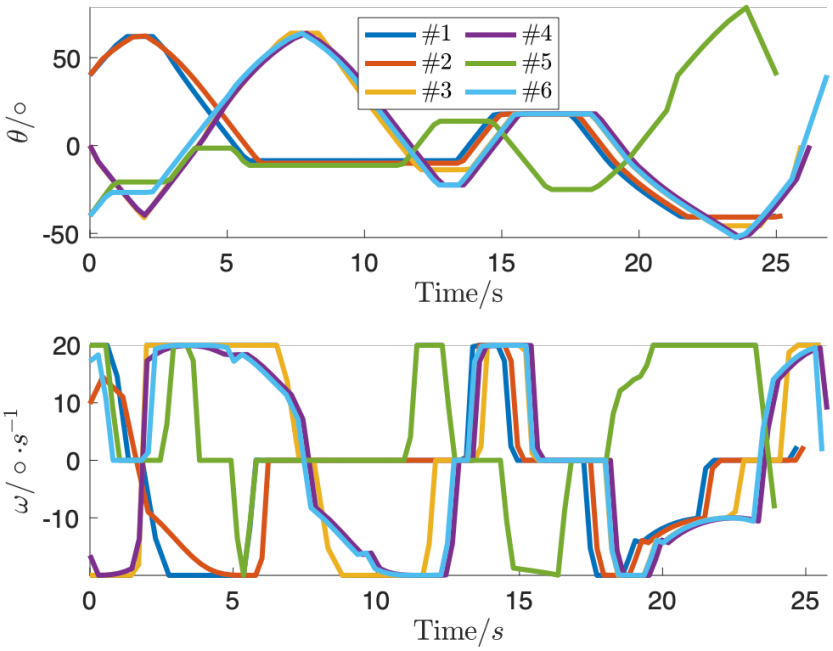


设置6个飞行任务为：

	初始位置 (x_0^*, y_0^*)	初始航向角 θ_0^*	终端位置 (x_N^*, y_N^*)	终端航向角 θ_N^*	采用何种算法
任务1	$(0,0)$	-40°	$(110,0)$	40°	算法1
任务2	$(0,0)$	-40°	$(110,0)$	40°	算法2
任务3	$(0,0)$	0°	$(110,0)$	0°	算法1
任务4	$(0,0)$	0°	$(110,0)$	0°	算法2
任务5	$(0,0)$	40°	$(110,0)$	-40°	算法1
任务6	$(0,0)$	40°	$(110,0)$	-40°	算法2

运行仿真，求解器选用 MOSEK，得到结果如下图所示。



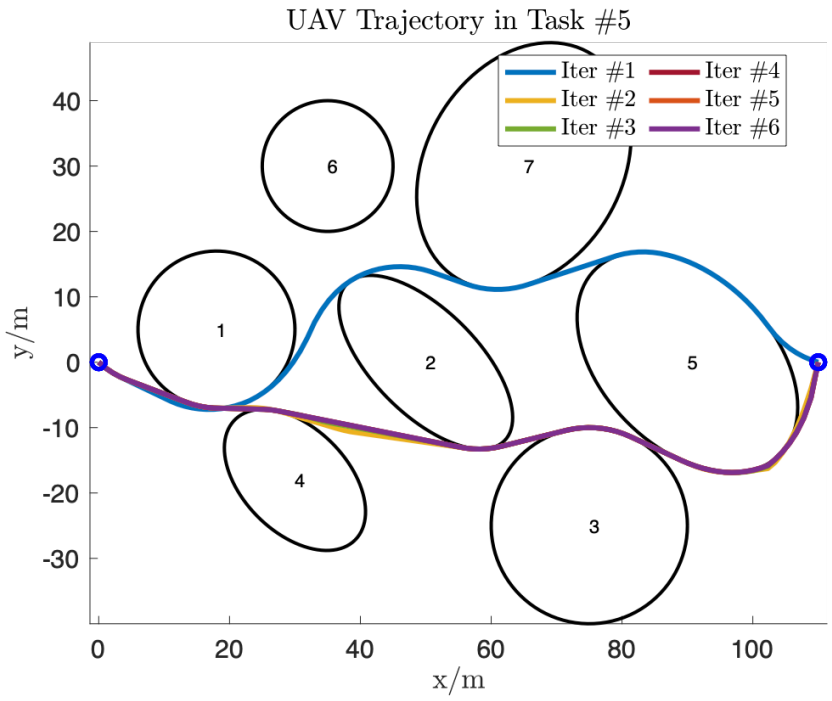


对于有禁飞区的任务，求解时间有明显增长，原因在于无禁飞区的轨迹规划问题为没有整数变量的二阶锥优化问题，而有禁飞区的限制之后，优化模型中加入了整数变量，问题成为混合整数二阶锥优化问题，相对降低了求解速度。

对于有禁飞区的任务，在相同的初始条件下，算法1和算法2所给出的轨迹仍会有较大区别（例如任务5和任务6），推测是由于加入禁飞区之后，优化问题变得更加复杂，在迭代过程中最终的轨迹会有较大的变化而导致的结果。

由于需要避开禁飞区，所以最终轨迹中，不可避免地会有较多的转向，在每条轨迹中，也会有多次航向角速度达到最大值的情况。

对于其中的任务5，每次迭代的轨迹结果如图所示。



可以看出，第一次迭代时的轨迹与其余迭代轮次的轨迹结果有较大差距，绕行障碍物的方法不同，与任务6的结果，即相同条件下设置不同初值并只迭代一次的结果类似。这也印证了，迭代次数会对解的最优性产生影响，并且在存在禁飞区的情况下，影响还会相较于无禁飞区的情况更大。

任务5

归功于 YALMIP 可以选择求解器的设置，可以不更改建模及求解代码，而仅仅通过修改求解函数中的求解器参数，从而实现切换求解器求解。

对于任务3和任务4的代码，我们分别采用Mosek求解器以及Gurobi求解器求解，并分别计时，结果如下。

求解器名称	任务3求解用时 / 秒	任务4求解用时 / 秒
Mosek	45	564
Gurobi	60	586

从结果中可以看出，对于两个任务的求解过程，Mosek的求解速度都优于Gurobi，其中任务3的用时相较于Gurobi减少25%，优势较为明显；任务4用时减少3.7%，几乎和Gurobi时间相当。

这也就说明了，对于属于二次锥优化问题无禁飞区优化问题的无禁飞区轨迹优化问题来说，采用Mosek更为节省时间；对于属于混合整数二次锥优化问题的有禁飞区轨迹优化问题来说，求解速度差别不大，两者均可以采用。

附录 MATLAB源代码

```
clc
close all
clear all

o(1) = Obstacle(18, 5, 12);
o(2) = Obstacle(50, 0, 17, 8, -45);
o(3) = Obstacle(75, -25, 15);
o(4) = Obstacle(30, -18, 13, 8, -45);
o(5) = Obstacle(90, 0, 20, 13, -45);
o(6) = Obstacle(35, 30, 10, 10, -30);
o(7) = Obstacle(65, 30, 15, 20, -30);

cnt = 0;


% Task 3
tic
for i = 1 : 3
    for j = 1 : 2
        cnt = cnt + 1;
        fprintf('Situation #%d...\n', cnt);
        res{cnt} = misocp(0, 0, 80 - 40 * i, 110, 0, -80 + 40 * i, [], ...
```

```
            Method=['algo' num2str(j)], IterRes='off', Solver='Mosek');
    end
end
for i = 1 : 2
    for j = 1 : 2
        cnt = cnt + 1;
        fprintf('Situation #%d...\n', cnt);
        res{cnt} = misocp(0, 0, 0, 90, 60 - 40 * i, -120 + 80 * i, [], ...
            Method=['algo' num2str(j)], IterRes='off', Solver='Mosek');
    end
end
toc
plotit(res, OmegaLegend='on');


% Task 4
plotobstacles(o);
tic
cnt = 0;
for i = 1 : 3
    for j = 1 : 2
        cnt = cnt + 1;
        fprintf('Situation #%d...\n', cnt);
        res4{cnt} = misocp(0, 0, 80 - 40 * i, 110, 0, -80 + 40 * i, o, ...
            Method=['algo' num2str(j)], IterRes='off', Solver='Mosek');
    end
end
toc

plotit(res4, ThetaLegend='on');

function ret = misocp(x_start, y_start, theta_start, x_end, y_end, theta_end, obstacles,
options)
    arguments
        x_start
        y_start
        theta_start
        x_end
        y_end
        theta_end
        obstacles
        options.Method = 'algo1'
        options.IterRes = 'off'
        options.Solver = 'Mosek'
    end
    x_0 = x_start;
    y_0 = y_start;
    theta_0 = deg2rad(theta_start);
    vartheta_0 = tan(theta_0);
```

```
x_f = x_end;
y_f = y_end;
theta_f = deg2rad(theta_end);
vartheta_f = tan(theta_f);

m = size(obstacles, 2);
for i = 1 : m
    o(i) = obstacles(i);
end

omega_max = deg2rad(20);
V = 5;
N = 101;
D = 1000;
dx = (x_f - x_0) / (N - 1);


y = sdpvar(N, 1);
vartheta = sdpvar(N, 1);
u = sdpvar(N, 1);
delta = sdpvar(N, 1);
x = sdpvar(N, 1);
t = sdpvar(N, 1);
eta = binvar(m, 1);

delta_s = ones(N, 1);

if strcmp(options.Method, 'algo1')
    iter_num = 10;
    delta_s = delta_s * 1.1;
else
    iter_num = 1;
end

x_r = x_0 : dx : x_f;


for count = 1 : iter_num
    fprintf('Iteration #%d...\n', count);
    F = (y(1) == y_0);
    F = F + (x(1) == x_0);
    F = F + (vartheta(1) == vartheta_0);
    F = F + (y(N) == y_f);
    F = F + (x(N) == x_f);
    F = F + (vartheta(N) == vartheta_f);
    F = F + (t(1) == 0);

    for i = 1 : N - 1
```

```
            F = F + (y(i + 1) == y(i) + vartheta(i) * dx);
            F = F + (vartheta(i + 1) == vartheta(i) + u(i) * dx);
            F = F + (x(i + 1) == x(i) + dx);
            F = F + (t(i + 1) == t(i) + delta(i) * dx / V);
        end

        for i = 1 : N
            F = F + (u(i) <= omega_max / V * ...
                (3 * delta_s(i) ^ 2 * delta(i) - 2 * delta_s(i) ^ 3));
            F = F + ((-u(i)) <= omega_max / V * ...
                (3 * delta_s(i) ^ 2 * delta(i) - 2 * delta_s(i) ^ 3));
        end

        for i = 1 : N
            F = F + (delta(i) >= sqrt(1 + vartheta(i) ^ 2));
        end

        for i = 1 : N
            for j = 1 : m
                [flag, y_max, y_min] = o(j).intersect(x_r(i));
                if flag
                    F = F + (y(i) >= y_max + D * (eta(j) - 1));
                    F = F + (y(i) <= y_min + D * eta(j));
                end
            end
        end
        opt = sdpsettings('verbose', 0, 'solver', options.Solver);
        solve = solvesdp(F, t(N), opt);

        if strcmp(options.IterRes, 'on')
            ret.y(:, count) = double(y);
            ret.x(:, count) = double(x);
            ret.delta(:, count) = double(delta);
            ret.vartheta(:, count) = double(vartheta);
            ret.theta(:, count) = rad2deg(atan(ret.vartheta(:, count)));
            ret.t(:, count) = double(t);
            ret.omega(:, count) = rad2deg(double(u) * V ./ double(delta) .^ 3);
        end

        if strcmp(options.Method, 'algo1')
            if max(abs(double(delta) - delta_s)) <= 1e-2
                fprintf('Converging! Exit!\n');
                break;
            end
        end


        delta_s = double(delta);
    end
```

```
    ret.algo = strcmp(options.Method, 'algo2') + 1;
    ret.N = N;
    ret.m = m;
    ret.obstacles = obstacles;
    if strcmp(options.IterRes, 'off')
        ret.y = double(y);
        ret.x = double(x);
        ret.delta = double(delta);
        ret.vartheta = double(vartheta);
        ret.theta = rad2deg(atan(ret.vartheta));
        ret.t = double(t);
        ret.omega = rad2deg(double(u) * V ./ double(delta) .^ 3);
    end

end

function plotit(res, op)
    arguments
        res
        op.ThetaLegend = 'off'
        op.OmegaLegend = 'off'
    end
    figure
    hold on;

    for i = 1 : res{1, 1}.m
        o = res{1, 1}.obstacles(i);
        b = o.boundary;
        plot(b(:, 1), b(:, 2), 'k', 'LineWidth', 2);
        text(o.ox, o.oy, num2str(i));
    end
    for i = 1 : size(res, 2)
        p(i) = plot(res{1, i}.x(:, end), res{1, i}.y(:, end), 'LineWidth', 3);
        plot([res{1, i}.x(1, end) res{1, i}.x(end, end)], ...
             [res{1, i}.y(1, end) res{1, i}.y(end, end)], ...
            'bo','MarkerSize', 8, 'LineWidth', 2.5);
        lgd{i} = ['$ \rm \#' num2str(i) '$'];
    end
    legend(p, lgd ,'Location','best','Interpreter','latex', 'NumColumns', 2);
    xlabel('x/m', 'Interpreter','latex');
    ylabel('y/m','Interpreter','latex');
    axis equal
    title('UAV Trajectory','Interpreter','latex');
    set(gca, 'fontsize', 15);
    hold off;

    figure
    subplot(2, 1, 1);
    hold on;
```

```
    for i = 1 : size(res, 2)
        p(i) = plot(res{1, i}.t(:, end), res{1, i}.theta(:, end), 'LineWidth', 3);
    end
    xlabel('{\rm Time}/s', 'Interpreter','latex');
    ylabel('$\theta/\circ$','Interpreter','latex');
    if strcmp(op.ThetaLegend, 'on')
        legend(p, lgd ,'Location','north','Interpreter','latex', 'NumColumns', 2);
    end
    axis tight
    set(gca, 'fontsize', 15);

    subplot(2, 1, 2);
    hold on;
    for i = 1 : size(res, 2)
        p(i) = plot(res{1, i}.t(1 : end - 1, end), res{1, i}.omega(1 : end - 1, end), '-',
'LineWidth', 3);
    end
    if strcmp(op.OmegaLegend, 'on')
        legend(p, lgd ,'Location','north','Interpreter','latex', 'NumColumns', 2);
    end
    xlabel('${\rm Time}/s$', 'Interpreter','latex');
    ylabel('$\omega/\circ\cdot s^{-1}$','Interpreter','latex');
    axis tight
    set(gca, 'fontsize', 15);


    figure
    hold on
    id = 1;
    for i = 1 : size(res, 2)
        if size(res{1, i}.y, 2) >= size(res{1, id}.y, 2)
            id = i;
        end
    end
    for i = 1 : res{1, 1}.m
        o = res{1, 1}.obstacles(i);
        b = o.boundary;
        plot(b(:, 1), b(:, 2), 'k', 'LineWidth', 2);
        text(o.ox, o.oy, num2str(i));
    end
    for i = 1 : size(res{1, id}.y, 2)
        p_iter(i) = plot(res{1, id}.x(:, i), res{1, id}.y(:, i), 'LineWidth', 3);
        plot([res{1, id}.x(1, i) res{1, id}.x(end, i)], ...
             [res{1, id}.y(1, i) res{1, id}.y(end, i)], ...
            'bo','MarkerSize', 8, 'LineWidth', 2.5);
        lgd_iter{i} = ['$\rm Iter\ \#' num2str(i) '$'];
    end
    legend(p_iter, lgd_iter ,'Location','best','Interpreter','latex', 'NumColumns', 2);
    xlabel('x/m', 'Interpreter','latex');
```

```
    ylabel('y/m','Interpreter','latex');
    axis equal
    title(['$\rm UAV\ Trajectory\ in\ Task\ \# ' num2str(id) '$'],'Interpreter','latex');
    set(gca, 'fontsize', 15);
    hold off;

end

function plotobstacles(ob)
    figure
    hold on;
    for i = 1 : size(ob, 2)
        o = ob(i);
        b = o.boundary;
        plot(b(:, 1), b(:, 2), 'k', 'LineWidth', 2);
        text(o.ox, o.oy, num2str(i));
    end
    xlabel('x/m', 'Interpreter','latex');
    ylabel('y/m','Interpreter','latex');
    axis equal
    title('Obstacles','Interpreter','latex');
    set(gca, 'fontsize', 15);
    hold off
end

classdef Obstacle
    properties
        ox
        oy
        ra
        rb
        gamma
    end
    methods
        function obj = Obstacle(ox_, oy_, ra_, rb_, g_)
            if nargin == 3
                rb_ = ra_;
                g_ = 0;
            elseif nargin == 4
                g_ = 0;
            end
            obj.ox = ox_;
            obj.oy = oy_;
            obj.ra = ra_;
            obj.rb = rb_;
            obj.gamma = deg2rad(g_);
        end
        function [flag, y_max, y_min] = intersect(obj, x)
            R = [cos(obj.gamma) -sin(obj.gamma); sin(obj.gamma) cos(obj.gamma)];
```

```
            r = [1 / obj.ra ^ 2, 0; 0, 1 / obj.rb ^ 2];
            syms inter_y real;
            p = [x - obj.ox; inter_y - obj.oy];
            sol = solve((R' * p)' * r * (R' * p) == 1, inter_y);
            flag = size(sol, 1) > 0;
            y_max = 0;
            y_min = 0;
            if size(sol, 1) > 0
                y_max = double(max(sol));
                y_min = double(min(sol));
            end
        end
        function b = boundary(obj)
            t = linspace(0, 2 * pi);
            xx = obj.ra * cos(t);
            yy = obj.rb * sin(t);
            R = [cos(obj.gamma) -sin(obj.gamma); sin(obj.gamma) cos(obj.gamma)];
            b = [xx' yy'] * R';
            b(:, 1) = b(:, 1) + obj.ox;
            b(:, 2) = b(:, 2) + obj.oy;
        end
    end
end
```