基于变步长搜索算法的板凳龙表演问题研究

摘要

本文主要研究了板凳龙表演的问题。通过螺旋线的极坐标方程，结合几何分析与变步长搜索算法，求解出了舞龙队各把手在各时刻的位置和速度、碰撞时刻、最小螺距、调头曲线及最大龙头速度等关键参数。

针对问题一，我们求解了舞龙队各把手在任意时刻的位置与速度。首先，借助曲线弧长公式推导出**螺旋线长度与极角的关系**，进而利用龙头的匀速运动确定任意时刻龙头前把手的位置；随后，通过几何分析推导出相邻把手间极角的**递推关系式**，并结合螺旋线方程得出任意时刻各把手的极坐标位置；最后，利用瞬时速度的定义，设置极小时间间隔，近似计算出了各把手的速度。详见支撑材料。

针对问题二，我们确定了舞龙队发生碰撞的时刻和位置。首先，将碰撞近似刻画为最内侧板凳顶角与外侧板凳边缘的碰撞；接着，利用板凳间的几何关系，确定了**碰撞发生所需的距离与位置条件**；最终，通过**变步长搜索算法，查找到碰撞时刻为 408.4 秒，且碰撞发生在第一块板与第十一块板之间**。

针对问题三，我们确定了龙头能够恰好不发生碰撞进入调头区域的最小螺距。通过将问题转化为寻找龙头刚进入调头区域时可能发生碰撞的最小螺距，结合**变步长搜索算法**，分析龙头距原点的距离与螺距之间的关系，最终确定**最小螺距为 0.4487 米**。

针对问题四，我们求得了龙头调头时的最短调头曲线及各把手的位置和速度。首先，通过几何分析，得出调整调头时大小圆半径比**无法改变调头曲线长度恒为** 4.5π **的结论，**发现无法同时满足盘入盘出点中心对称与调头曲线优化的条件。因此，我们放宽了盘入盘出点对称性与大圆小圆半径比 2:1 的约束，经由**仿真建模**，最终得出在盘出点可变条件下的**最小调头曲线长度为** 10.36 **米，盘入盘出点与中心螺线的夹角为** $95.3°$。然而，考虑到实际表演中的**观赏性和失误容错率问题，最终选取了夹角为** $120°$**，总弧长为** 11.78 **米的调头路径，**并据此计算出龙头及各把手的位置和速度。

针对问题五，我们计算了在把手速度上限条件下的龙头最大速度。首先，利用舞龙队螺旋盘入的几何关系，**确定各把手的速度与龙头速度成比例**；然后，通过遍历各把手在不同时间点的速度，找出最大速度比，从而确定在保证各把手速度不超过 2 m/s 的情况下龙头的最大速度。最终，**计算得出最大速度比为** 1.0256**，最大龙头速度为** 1.95 **m/s**。

本文的特色在于将几何分析与变步长搜索算法相结合，在考虑复杂模型的基础上，保证了求解精度与计算效率，为板凳龙表演的分析与优化提供了参考依据。

关键字：**板凳龙　阿基米德螺线　极坐标　变步长搜索算法　几何分析　仿真建模**

一、 问题重述

1.1 问题背景

“板凳龙”，又称“盘龙”，是浙江和福建地区具有代表性的传统民俗文化活动之一。在活动中，人们将几十条甚至上百条板凳首尾相连，形成一条长长的“龙”形队列。舞龙时，龙头领队前行，龙身和龙尾紧随其后，盘旋成圆盘状。为了增强表演的观赏效果，舞龙队不仅需要灵活地完成盘龙的进退动作，还要尽可能减少占地面积，同时提高移动的速度。如何优化这些因素并使得盘龙表演更加吸引观众的注意力成为了我们亟待解决的问题。

1.2 问题要求

板凳龙由 223 节板凳组成，包括 1 节龙头板凳、221 节龙身板凳和 1 节龙尾板凳。龙头的长度为 341cm，龙身和龙尾的板凳长度则为 220cm，所有板凳的宽度均为 30cm。每节板凳上都有两个直径为 5.5cm 的孔，孔中心距离板凳端部 27.5cm。相邻板凳通过把手连接固定。

问题一：舞龙队伍沿着螺距为 55cm 的等距螺线顺时针盘入，所有连接点的中心都位于螺线上，且龙头的行进速度恒定为 1m/s。龙头最初位于螺线第 16 圈与 x 轴交点。任务要求计算并记录从初始时刻至 300s 内，每秒钟龙头、龙身、龙尾各个关键节点的具体位置和速度。结果保存至 `result1.xlsx` 文件中，并在特定时刻通过表格形式展示龙头前把手、龙身部分节点和龙尾后把手的具体位置与速度。

问题二：要求舞龙队沿问题一设定的螺线进行盘入，计算队伍无法继续盘入的停止时刻（即队伍内部板凳即将碰撞的时刻），并记录此时队伍的整体位置和速度。将结果保存到 `result2.xlsx` 文件中，并在论文中提供特定节点（龙头前把手、部分龙身节点及龙尾后把手）的具体位置和速度。

问题三：从顺时针盘入到逆时针盘出，舞龙队需要在调头时预留足够的空间。假设调头空间为以螺线中心为圆心、直径为 9m 的圆形区域，计算龙头前把手在不发生碰撞的情况下盘入至调头区域边界时的最小螺距。

问题四：考虑调头问题。已知螺距为 1.7m，盘出螺线与盘入螺线中心对称，调头路径由两段相切的圆弧组成 S 形轨迹，且该路径分别与螺线相切。在此前提下，调整圆弧的形状，缩短整个调头路径。

问题五：根据问题四中给定的路线，计算出在保证舞龙队各把手的速度均不超过 2 米/秒的情况下，龙头能够保持的最大恒定速度。

二、 问题分析

2.1 问题一分析

问题一要求我们求解 0 到 300 秒内舞龙队各把手的位置和速度。首先，我们进行极坐标转换，并用极坐标表示螺旋线的方程。其次，通过曲线弧长公式，得到螺旋线总长度与极角的函数关系。考虑到龙头前把手匀速运动，利用该关系可以计算任意时刻龙头的极角。接着，通过相邻节点极角的递推关系式和螺旋线方程，能够得到该时刻所有节点的极坐标。最后，使用瞬时速度的定义式，结合极小时间段内的位移变化，计算出各时刻各节点的速度。

2.2 问题二分析

问题二要求我们找到舞龙队发生碰撞的时刻，并给出此时各把手的位置与速度。首先，我们对碰撞进行了刻画。由于只需寻找首次碰撞的时刻，碰撞必然发生在最内圈，且是内侧板凳顶角与外圈板凳边缘的碰撞。接着，通过分析碰撞发生时的距离条件和位置条件，我们确定了判断碰撞发生的条件。最后，采用变步长搜索算法查找碰撞时

2.3 问题三分析

问题三要求我们找到使龙头不发生碰撞地进入调头空间的最小螺距。该问题可转化为求解使龙头恰好在进入调头空间时发生碰撞的螺距。我们使用变步长搜索法进行求解。首先，通过大步长调整螺距，确定不同螺距下龙头距离原点的位置，锁定接近调头空间半径 4.5 m 附近的螺距范围。接着，通过小步长搜索，找到使龙头成功进入调头空间的最小螺距。

2.4 问题四分析

问题四要求我们在调头区域内调整圆弧，缩短调头曲线，并给出调头前后 100 秒内各把手的位置和速度。在满足各部分相切、两条螺线对称的几何条件下，通过几何分析和绘图，尝试改变两圆弧的半径比，结果发现调头弧长无法进一步优化。因此，我们舍弃了螺线对称的条件，改变盘出螺线的盘出点位置以及两圆弧的半径比，建立几何模型并计算调头曲线长度。依照结果设计调头路径后，使用 MATLAB 进行仿真建模，得到了各时刻各把手的位置与速度。

2.5 问题五分析

问题五要求我们求出龙头的最大速度，使得舞龙队各把手在各时刻的速度不超过 2 m/s。因此，我们只需找到最大速度比，即可通过调整龙头速度，求得在最大把手速度不超过 2 m/s 的情况下，龙头的最大速度。

三、 模型假设

1. **假设一：**所有把手位于同一高度，不考虑高度对于问题的影响。
2. **假设二：**忽略把手孔径的大小，以孔的中心作为把手的位置。
3. **假设三：**舞龙队操纵把手在固定的螺线上单向运动，不存在任何失误。
4. **假设四:** 问题四种盘出点与盘入点关于原点的夹角默认为钝角，否则会因为转弯半径过小，不符合现实情形无法转弯。

四、 符号说明

表 1　符号说明表

符号	说明	单位
i	把手序号	-
L	螺距	m
$v_{i,t}$	序号为 i 的把手在 t 时刻的速度	m/s
$\theta_{i,t}$	序号为 i 的把手在 t 时刻的极角	rad
$r_{i,t}$	序号为 i 的把手在 t 时刻的极径	m
a	螺线方程参数	-
$l(\theta)$	从螺线中心起算，旋转角度为 θ 时的螺线长度	m
(x_{in}, y_{in})	盘入点在直角坐标系下的坐标	-
(x_{out}, y_{out})	盘出点在直角坐标系下的坐标	-
h_i	第 i 号把手与第 $i+1$ 号把手的距离（$i=0$ 时取 2.86，其余取 1.65）	m
d	内圈板凳的顶角与外圈板凳前后把手连线的距离	m
b	舞龙队发生碰撞时龙头与原点的距离	m
L_m	不碰撞进入调头空间的最小螺距	m
D	调头区域直径	m
$\vec{\alpha}$	盘入盘出螺线切线的单位向量	-

符号	说明	单位
k	调头时经过大圆与小圆半径比	-
ω	盘入点盘出点与圆心的连线夹角	rad
S	调头曲线弧长	m
$\eta_{i,t}$	t 时刻 i 号把手与 0 号把手的速度比	-

五、 模型的建立与求解

5.1 问题一的建立与求解

5.1.1 直角坐标系的建立与极坐标转换

以螺线中心为原点，水平方向为 x 轴，竖直方向为 y 轴，如图 1a 所示，建立平面直角坐标系。

(a) 平面直角坐标系与螺线方程　　**(b) 极坐标系与螺线方程**

图 1　螺线方程在不同坐标系下的表示

我们同样将 x 轴作为极轴，螺线中心作为极点，建立极坐标系。通过极坐标转换公式，可以将螺线方程表示为极坐标形式。

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases} \quad \Rightarrow \quad r = a\theta \tag{1}$$

其中，a 是螺线方程中的参数，下面将求解 a。如图 1b所示，由于 $L = 0.55$，螺线向外旋转一周会使极径增加 L，而极角增加 2π，因此我们有：

$$r_1 = r_2 + L = a\left(\theta + 2\pi\right) = a\theta + L \tag{2}$$

$$\Rightarrow a = \frac{L}{2\pi} \tag{3}$$

5.1.2 螺线长度 $l(\theta)$ 与极角 θ 的关系

利用曲线弧长的公式，我们可知：

$$\begin{aligned} l(\theta) &= \int_0^\theta \sqrt{\left(\frac{\mathrm{d}r}{\mathrm{d}\theta}\right)^2 + r^2}\,\mathrm{d}\theta \\ &= \int_0^\theta \sqrt{a^2 + a^2\theta^2}\,\mathrm{d}\theta \\ &= a\int_0^\theta \sqrt{1+\theta^2}\,\mathrm{d}\theta \\ &= \frac{a}{2}\left[\theta\sqrt{1+\theta^2} + \ln\left(\theta + \sqrt{1+\theta^2}\right)\right] \end{aligned} \tag{4}$$

其中，$l(\theta)$ 为从螺线中心 (0,0) 起算，旋转角度为 θ 时的螺线长度。

5.1.3 把手的位置与速度

1. **龙头前把手的极角：** 我们现在考虑板凳龙的队列。将所有 224 个把手编号，记为 i。龙头前把手编号为 0 号，龙身前把手从 1 号依次编号至 221 号，龙尾前把手编号为 222 号，龙尾后把手为 223 号。根据 5.1.2，只要知道龙头前把手在时刻 t 的螺线长度 $l(\theta_{0,t})$，即可求得该时刻龙头前把手的极角 $\theta_{0,t}$：

$$\int_0^t v_{0,t}\,\mathrm{d}t + l(\theta_{0,t}) = l(\theta_{0,0}) \tag{5}$$

   其中，$\int_0^t v_{0,t}\,\mathrm{d}t$ 表示在时间 t 内，龙头前把手沿螺线行进的距离。由于其为匀速运动 $v_{0,t} \equiv 1$，该项也可以化简为 $v_{0,t}\cdot t$。

2. **龙身与龙尾把手的极角：** 如图 2 所示，利用余弦定理可以推导出相邻两龙身把手之间的递推关系式，即：

$$\begin{cases} r_{i,t} = a\theta_{i,t} \\ r_{i+1,t} = a\theta_{i+1,t} = a\left(\theta_{i,t} + \gamma\right) = r_{i,t} + a\gamma \end{cases} \tag{6}$$

$$\begin{aligned} \cos\gamma &= \frac{r_{i,t}^2 + r_{i+1,t}^2 - h_i^2}{2r_{i,t}\cdot r_{i+1,t}} \\ &= \frac{r_{i,t}^2 + \left(r_{i,t} + a\gamma\right)^2 - h_i^2}{2r_{i,t}\left(r_{i,t} + a\gamma\right)} \end{aligned} \tag{7}$$

   其中，$r_{i,t}$ 为第 i 号把手在时刻 t 的极径；h_i 为第 i 号把手与 $i+1$ 号把手的距离；γ 为两把手与螺线中心连线形成的角度。方程 (7) 仅有未知量 γ，通过求解该方程可以得到 γ，从而相邻两龙身把手之间的递推关系可表示为：

$$\theta_{i+1,t} = \theta_{i,t} + \gamma \tag{8}$$

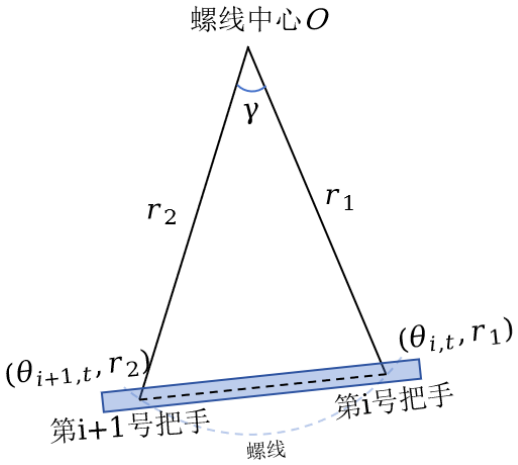


图 2　相邻把手间的几何关系

3. **把手的位置:** 根据螺旋线方程 $r = \frac{L}{2\pi}\theta$，我们可以得出第 i 号把手在极坐标系下的位置为 $(\theta_{i,t}, \frac{L}{2\pi}\theta_{i,t})$。
4. **把手的速度**：基于瞬时速度的定义，我们采用微分的思想。通过取一个极小的时间间隔 Δt，在此时间段内，可以假设各个把手的速度保持不变。此时，把手的位移与时间的比值即为该时刻的瞬时速度:

$$v_{i,t} \approx \frac{|\vec{c}_{i,t+\Delta t} - \vec{c}_{i,t}|}{\Delta t} \tag{9}$$

其中，$\vec{c}_{i,t}$ 表示第 i 号把手在时刻 t 的直角坐标，$x_{i,t}$ 为其横坐标，$y_{i,t}$ 为其纵坐标，表达式为:

$$\vec{c}_{i,t} = (x_{i,t}, y_{i,t}) = (r_{i,t}\cos\theta_{i,t}, r_{i,t}\sin\theta_{i,t}) \tag{10}$$

5.1.4 问题求解

通过给定龙头前把手在平面直角坐标系中的初始位置 $(8.8, 0)$，可以确定其在极坐标系中的极角 $\theta_{0,0} = 32\pi$ 以及螺线长度 $l(\theta_{0,0}) = 442$。为了计算任意时刻各个把手的位置与速度，**我们选取时间步长为 1 秒，遍历从 0 秒到 300 秒的时间段**。在每个时刻，采用以下步骤来确定该时刻各个把手的位置与速度:

- **确定龙头前把手的极角**：基于龙头前把手的初始螺线长度 $l(\theta_{0,0}) = 442$、匀速运动状态 $v_{0,t} = 1$ 以及公式 (5)，可以求得任意时刻龙头前把手的极角位置。
- **确定龙身与龙尾把手的极角**：利用递推关系式 (8)，从龙头前把手开始，依次向后递推计算各个把手在该时刻的极角。

- **确定极坐标：**根据螺线方程 $r = \frac{0.55}{2\pi}\theta$ （螺距 $L = 0.55\,m$），可以计算出此时各个把手的极坐标。
- **确定速度：**考虑一个极小的时间间隔 $\Delta t = 0.05$ 秒，利用瞬时速度的定义式 (9)，可以求得该时刻各个把手的瞬时速度。

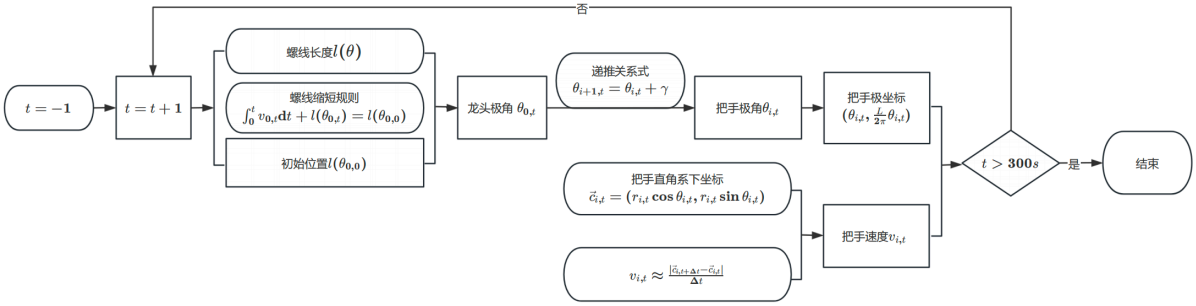


图 3　问题一流程图

5.1.5 结果分析

通过 MATLAB 求解，我们计算出了从 0 秒到 300 秒期间，每秒整个舞龙队各把手的位置和速度，并将结果保存在`result1.xlsx` 文件中。表 2和表 3分别给出了部分时刻部分把手的位置和速度。

表 2　部分把手的位置

	0s	60s	120s	180s	240s	300s
龙头 x (m)	8.800000	5.799209	-4.084887	-2.963609	2.594494	4.420274
龙头 y (m)	0.000000	-5.771092	-6.304479	6.094780	-5.356743	2.320429
第 1 节龙身 x (m)	8.363824	7.456758	-1.445473	-5.237118	4.821221	2.459489
第 1 节龙身 y (m)	2.826544	-3.440399	-7.405883	4.359627	-3.561949	4.402476
第 51 节龙身 x (m)	-9.518732	-8.686317	-5.543150	2.890455	5.980011	-6.301346
第 51 节龙身 y (m)	1.341137	2.540108	6.377946	7.249289	-3.827758	0.465829
第 101 节龙身 x (m)	2.913983	5.687116	5.361939	1.898794	-4.917371	-6.237722
第 101 节龙身 y (m)	-9.918311	-8.001384	-7.557638	-8.471614	-6.379874	3.936008

	0s	60s	120s	180s	240s	300s
第 151 节龙身 x (m)	10.861726	6.682311	2.388757	1.005154	2.965378	7.040740
第 151 节龙身 y (m)	1.828753	8.134544	9.727411	9.424751	8.399721	4.393013
第 201 节龙身 x (m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第 201 节龙身 y (m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾（后）x (m)	-5.305444	7.364557	10.974348	7.383896	3.241051	1.785033
龙尾（后）y (m)	-10.676584	-8.797992	0.843473	7.492370	9.469336	9.301164

如图 4 所示，300s 时各把手的位置如图所示。红点表示龙头，蓝点表示各个把手的位置。红色曲线表示龙身在 16 圈以内的部分，绿色曲线表示 16 圈以外的部分。

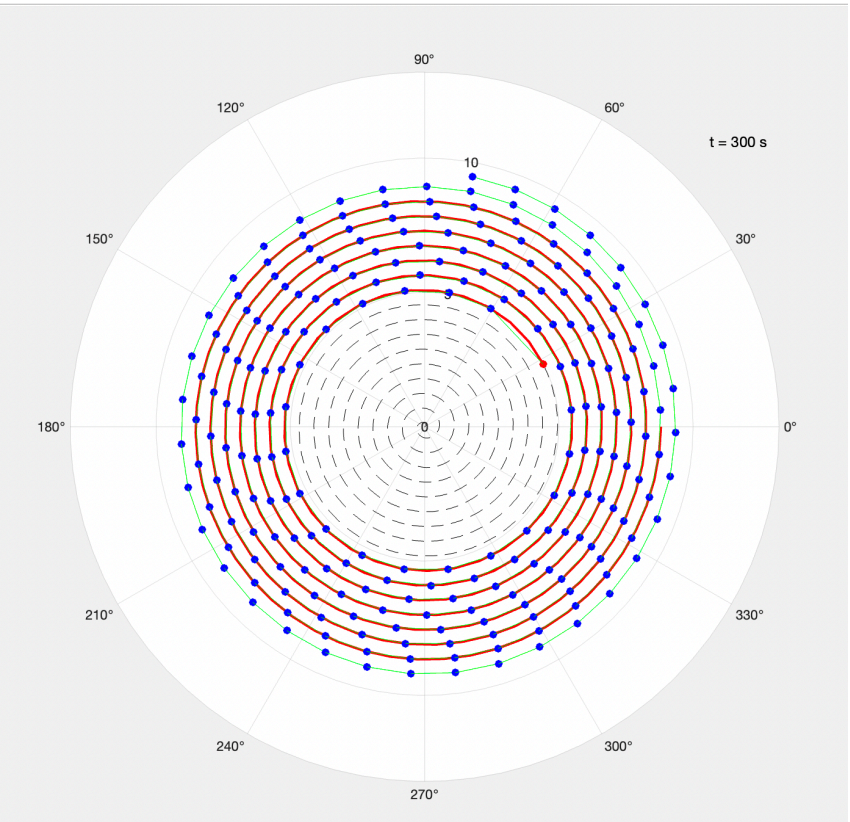


图 4　300s 时舞龙队俯视图

表 3　部分把手速度

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 (m/s)	0.999999	0.999998	0.999998	0.999998	0.999997	0.999996
第 1 节龙身 (m/s)	0.999970	0.999960	0.999943	0.999914	0.999856	0.999705
第 51 节龙身 (m/s)	0.999741	0.999661	0.999537	0.999329	0.998939	0.998062
第 101 节龙身 (m/s)	0.999574	0.999452	0.999268	0.998969	0.998433	0.997299
第 151 节龙身 (m/s)	0.999447	0.999298	0.999077	0.998726	0.998113	0.996859
第 201 节龙身 (m/s)	0.999347	0.999179	0.998934	0.998550	0.997892	0.996572
龙尾（后） (m/s)	0.999310	0.999136	0.998882	0.998488	0.997815	0.996476

根据上述表格我们可以近一步地得到以下结论：

- **龙头速度**：在我们的结果中，龙头的速度未能精确达到 1 m/s。这是由于我们采用瞬时速度的定义进行计算，而 Δt 无法取到无限小（此处取 0.05 秒），因此龙头速度不可能恰好为 1 m/s。随着舞龙队逐渐向内盘旋，这种误差会逐渐增大，考虑到 $0.999996 \approx 1$ m/s，该结果仍在合理且可接受的误差范围内。
- **节数增加速度递减**：龙身从第 1 节到第 201 节的速度呈现逐渐减小的趋势。这表明，随着队伍沿螺旋盘入，后方板凳的运动在每秒内受到一定的阻滞或延迟，导致速度逐步减缓。因此，越靠后的龙节，速度下降的幅度越大。这种现象在较长时间的行进中会更加显著，与我们在“板凳龙”相关资料 [1] 和视频 [2] 中观察到的现象一致。
- **龙身速度递减规律**：随着时间的推移，除速度恒定外的把手速度均逐渐减小。
- **速度的渐进性**：从第 1 节到第 201 节龙身的速度减缓幅度较小，说明舞龙队的整体速度变化较为平稳，没有出现明显的速度骤降或波动。这表明模型运行较为顺畅，没有发生板凳之间的碰撞或速度严重失衡的情况。

5.2 问题二的建立与求解

5.2.1 简化与准备

1. **只考虑最内一圈板凳的碰撞情况：**

$$\begin{cases} \rho = \frac{(r^2+a^2)^{\frac{3}{2}}}{r^2+2a^2} == \frac{a(\theta^2+1)^{3/2}}{\theta^2+2} \\ \frac{d\rho}{d\theta} = \frac{(\theta^2+2)\cdot 3a\theta(\theta^2+1)^{1/2} - a(\theta^2+1)^{3/2}\cdot 2\theta}{(\theta^2+2)^2} \end{cases} \tag{11}$$

其中，ρ 为螺线的曲率半径。将 $a = \frac{L}{2\pi}$ 代入 $\frac{d\rho}{d\theta}$ 中可以得出，当 $\theta > 0$ 时，总有 $\frac{d\rho}{d\theta} > 0$，因此随着极角的增大，曲率半径单调递增。外侧螺线的曲率半径大于内侧螺线的曲率半径。在较外侧，矩形板凳与螺线的贴合度较高；而在内侧，矩形板凳与螺线之间的间隙较大，更容易发生板凳相撞。如果外侧板凳发生碰撞，那么内侧板凳必定先发生碰撞，因此我们只需要考虑最内侧板凳的碰撞情况。

2. **只考虑矩形板凳顶角发生的碰撞：**我们将两块矩形板凳在平面上的重叠定义为碰撞。只需考虑两块矩形板凳的顶角与边缘的碰撞，而无需考虑边与边的碰撞情况。因为如果两块板凳的长边相交，那么在此之前一块板凳的顶角一定已经与另一块板凳的边缘发生了重叠（即发生碰撞）。

3. **仅考虑内侧板凳顶角撞击外侧板凳边缘：**当内侧板凳的顶角碰到外侧板凳边缘时，由于外侧板凳的前后把手连线位于螺旋线的弧内（相对于平直板凳外凸），且相邻螺旋线的间距（螺距）始终固定为 1.7 m，因此更容易发生碰撞。相反，当外侧板凳的顶角接近内侧板凳的边缘时，由于内侧板凳的前后把手连线也位于螺旋线的弧内，距离较远，碰撞不易发生。由此可见，内侧板凳的顶角必然先于外侧板凳顶角碰到板凳边缘，因此我们只需考虑内侧板凳顶角与外侧板凳边缘的碰撞情况。

5.2.2 板凳两角坐标的求解

为了准确描述板凳之间的碰撞，我们需要求出矩形板凳两角的坐标。如 图 5a 所示，设内侧板凳的前把手为 $A_1(x_1, y_1)$，后把手为 $A_2(x_2, y_2)$，外侧板凳的前把手为 $B_1(x_3, y_3)$，后把手为 $B_2(x_4, y_4)$。则：

$$\overrightarrow{A_2A_1} = (x_1 - x_2, y_1 - y_2) \tag{12}$$

D_1 的坐标为：

$$(x_1, y_1) + \overrightarrow{A_2A_1} \cdot \frac{0.275}{|\overrightarrow{A_2A_1}|}$$

其中，$|\overrightarrow{A_2A_1}|$ 在龙头处取值为 $3.41 - 0.275 \times 2 = 2.86$，在龙身或龙尾处取值为 $2.20 - 0.275 \times 2 = 1.65$。

根据 C_1D_1 的距离为 0.15 且 $C_1D_1 \perp A_2A_1$，我们可以求出板凳顶边 C_1D_1 所在的

单位向量 $\vec{\mu}$：

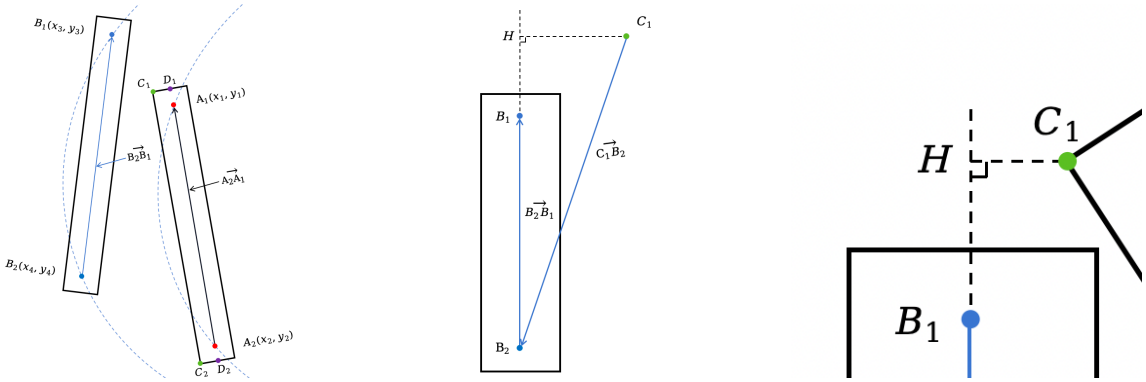
$$\vec{\mu} = (y_2 - y_1, x_1 - x_2) \cdot \frac{0.15}{|\overrightarrow{A_2A_1}|} \tag{13}$$

由于无法预先判断 $\vec{\mu}$ 的方向，即 $\vec{\mu}$ 可能是 $C_1\vec{D}_1$ 或 $D_1\vec{C}_1$ 的单位向量，因此我们通过将 D_1 的坐标与 $\vec{\mu}$ 相加或相减，得到板凳两顶角的坐标。选择两坐标中模长较大的，即可得到 C_1 的坐标：

$$(x_1, y_1) + \overrightarrow{A_2A_1} \cdot \frac{0.275}{|\overrightarrow{A_2A_1}|} \pm \vec{\mu}$$

同理，我们也可以求出 C_2 的坐标：

$$(x_2, y_2) + \overrightarrow{A_1A_2} \cdot \frac{0.275}{|\overrightarrow{A_2A_1}|} \pm \vec{\mu}$$



(a) 碰撞的距离判定示意图 **(b) 碰撞的位置判定示意图** **(c) 不发生碰撞的示意图**

图 5　碰撞判定示意图

5.2.3 碰撞的判定

1. **距离判定**：判断两块板凳是否相撞，只需判定一块板凳的顶角是否与另一块板凳的边缘相撞即可。在数学上，相撞的一个必要条件是顶角到另一块板凳前后把手连线的距离是否小于板凳宽度的一半。

$$d = \frac{\left|\overrightarrow{CB_1} \times \overrightarrow{CB_2}\right|}{\left|\overrightarrow{B_1B_2}\right|} \tag{14}$$

   其中，d 为 C 点到 B_1B_2 线段的距离。

2. **位置判定**：为了确保内侧板凳顶角与外侧板凳的边缘确实相碰，而不是如 **??** 中那样错开，内侧板凳顶角关于外侧板凳前后把手连线的投影 H 需要落在外侧板凳上。通过此条件来保证碰撞的位置判定条件成立。利用 C_1H 垂直于 B_1B_2，且 B_1B_2 与

B_2H 共线的几何关系，我们可以得到：

$$\overrightarrow{C_1H} = \overrightarrow{C_1B_1} + \overrightarrow{B_1H} \tag{15}$$

$$= \overrightarrow{C_1B_1} + \lambda\overrightarrow{B_1B_2} \tag{16}$$

$$\overrightarrow{C_1H} \cdot \overrightarrow{B_1B_2} = \lambda\overrightarrow{B_1B_2} \cdot \overrightarrow{B_1B_2} + \overrightarrow{C_1B_1} \cdot \overrightarrow{B_1B_2} = 0 \tag{17}$$

$$\Rightarrow \lambda = \frac{\overrightarrow{B_1C_1} \cdot \overrightarrow{B_1B_2}}{\overrightarrow{B_1B_2} \cdot \overrightarrow{B_1B_2}} \tag{18}$$

因此，我们可以求得 $\overrightarrow{B_1H}$ 和 $\overrightarrow{B_2H}$：

$$\begin{cases} \overrightarrow{B_1H} = \lambda\overrightarrow{B_1B_2} \\ \overrightarrow{B_2H} = \lambda\overrightarrow{B_1B_2} - \overrightarrow{B_1B_2} \end{cases} \tag{19}$$

为了判断 H 点是否在外侧板凳上，我们利用 $\overrightarrow{B_1H} \cdot \overrightarrow{B_2H}$ 的范围进行判定。当 H 在线段 B_1B_2 外时，点积为正；当 H 位于 D_1 或 D_2 时，取值为最大 $0.275 \times (0.275 + |\overrightarrow{A_2A_1}|)$。当 H 在线段 B_1B_2 上时，点积为负，其最小值在线段 B_1B_2 的中点处取到，值为 $-\frac{|\overrightarrow{A_1A_2}|^2}{4}$。只有当 $\overrightarrow{B_1H} \cdot \overrightarrow{B_2H} \in \left\{-\frac{|\overrightarrow{A_2A_1}|^2}{4}, 0.275 \times (0.275 + |\overrightarrow{A_2A_1}|)\right\}$ 时，内侧板凳才会与外侧板凳发生碰撞。

综上，我们得出板凳碰撞的条件为：

$$\begin{cases} d = \frac{\left|\overrightarrow{CB_1} \times \overrightarrow{CB_2}\right|}{\left|\overrightarrow{B_1B_2}\right|} < 0.15 \\ \overrightarrow{B_1H} \cdot \overrightarrow{B_2H} \in \left\{-\frac{|\overrightarrow{A_2A_1}|^2}{4}, 0.275 \times (0.275 + |\overrightarrow{A_2A_1}|)\right\} \end{cases} \tag{20}$$

5.2.4 模型的求解——变步长搜索算法

通过上文的分析，我们已经得出判断碰撞的条件。为了在每个时间点进行碰撞判断并减少计算量，我们采用了变步长搜索算法，具体过程如下：

Step 1: 初始化设定

- 螺距 $L = 0.55\,\text{m}$
- 龙头板凳前后把手间距为 $2.86\,\text{m}$，龙身板凳前后把手间距为 $|\overrightarrow{A_2A_1}| = 1.65\,\text{m}$
- 龙头速度 $v_{0,t} = 1\,\text{m/s}$
- 龙头前把手的初始极角 $\theta_{0,0} = 32\pi$，初始螺线长度 $l(\theta_{0,0}) = 442\,\text{m}$
- 搜索的起始时间为 300 秒，终止时间为 442 秒。第一次大步长搜索的步长为 10 秒，第二次中步长搜索的步长为 1 秒，第三次小步长搜索的步长为 0.1 秒。

Step 2: 碰撞判定

1. 根据问题一的求解，碰撞必然发生在 16 圈以内。通过计算 $[\frac{16\cdot 0.55\cdot 2\pi}{1.65}]=33$，可得出在第 16 圈螺线中最多容纳 33 个把手。因此，我们只需对 1 至 32 号把手进行遍历（遍历到 33 号和 32 号的效果相同）。
2. 在判断第 i 号板凳是否发生碰撞时，只考虑内侧顶角碰撞外侧板凳边缘的情况，因此只需遍历第 $i+2$ 至第 223 块板凳（对应于第 $i+1$ 至第 223 号把手），因为 i 号之前的板凳已作为外侧板凳进行过碰撞判断（不考虑外侧顶角碰撞内侧板凳的情况）。此外，$i+1$ 号板凳与 i 号板凳相连，不判断碰撞。
3. 在具体判断碰撞之前，首先通过检查内外板凳节点之间的距离是否大于 1 米来排除不可能发生碰撞的情况（1 米是一个足够宽的上界），以减少计算量。然后，对剩余的板凳及把手，按照公式 (20) 进行碰撞判定。

Step 3: 粗步长搜索

根据问题一的结果，300 秒内不会发生碰撞。而由于螺旋线的总长度为 442 米，且龙头的速度为 1 米/秒，可以推断在 442 秒时，龙头将达到螺旋线的尽头，此时舞龙队必定发生碰撞。因此，首次碰撞发生的时间一定在 300 秒到 442 秒之间。我们以 10 秒为步长进行大步长搜索，重复 Step 2，确定碰撞发生的时间区间。

Step 4: 细步长定位

在找到的 10 秒碰撞区间内，先以 1 秒为步长进行中步长搜索（重复 Step 2），然后再以 0.1 秒为步长进一步精确定位，重复 Step 2 确定碰撞时间。

5.2.5 结果分析

通过使用 MATLAB 求解，**我们得出舞龙队发生碰撞的时间为 408.4 秒，碰撞发生在第 1 节的后把手（1 号把手）所对的顶角与第 11 节龙身板凳之间**。此时舞龙队各把手的位置和速度信息已存入 `result2.xlsx` 文件中，部分结果如下：

根据以上结果进行分析，我们可以得出以下结论：

- **龙头位置**：发生碰撞时，龙头的坐标为 $(-2.241382, 0.962947)$，此时它距离原点较近，约为 2.43948 米。
- **龙身速度**：与问题一中的结果一致，龙身的速度随着节数的增加呈递减趋势。特别地，前十节龙身的速度均大于 1 m/s。
- **速度无递减性**：与问题一中的分析不同，在本问题多考虑的 300 秒至 408.4 秒的时间范围内，龙身速度并未像问题一的结果那样持续递减。这表明，速度递减规律可能仅在特定的时间范围内存在。随着舞龙队沿螺旋线逐渐盘旋深入，螺旋曲率半径逐渐减小，情况变得更加复杂多变，因此龙身速度不再呈现出规律性的下降。

表 4　各把手 408.4s 时的位置和速度信息

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头 (m/s)	-2.241382	0.962947	0.999982
第 1 节龙身 (m/s)	0.281645	2.309771	1.007872
第 51 节龙身 (m/s)	3.020415	3.298676	0.991599
第 101 节龙身 (m/s)	-2.481197	-5.324603	0.989191
第 151 节龙身 (m/s)	-1.054539	-6.919365	0.988219
第 201 节龙身 (m/s)	-7.926677	0.793242	0.987692
龙尾（后）(m/s)	2.919685	7.829588	0.987530

5.3 问题三的建立与求解

5.3.1 问题转化

实际上，确定龙头能够进入调头空间的最小螺距，等价于找到舞龙队刚好在进入调头空间时发生碰撞的螺距大小。用 b 表示舞龙队发生碰撞时龙头与原点的距离，我们希望找到 $b = \frac{D}{2} = 4.5$ 时对应的螺距，该螺距即位最小螺距 L_m。

5.3.2 变步长搜索

为了探明 b 与 L 的关系，并确定 $b = 4.5$ 时对应的最小螺距 L_m，我们采用变步长搜索算法：

- **粗步长定位**：在 $L = 0.28$ 至 $L = 0.70$ 的范围内，我们以 0.01m 为步长等间隔选取了 43 个不同的螺距 L。其中，$L = 0.28\,\mathrm{m}$ 是搜索下界，这时龙头出发点 16 圈处已经在调头区域内；$L = 0.70\,\mathrm{m}$ 是搜索上界，方便观察更宽区间上 b 和 L 的关系。通过调整螺距 L 并结合问题二中的方法，计算对应的 b 值，进而绘制出 b 随 L 变化的曲线，如图 6所示。

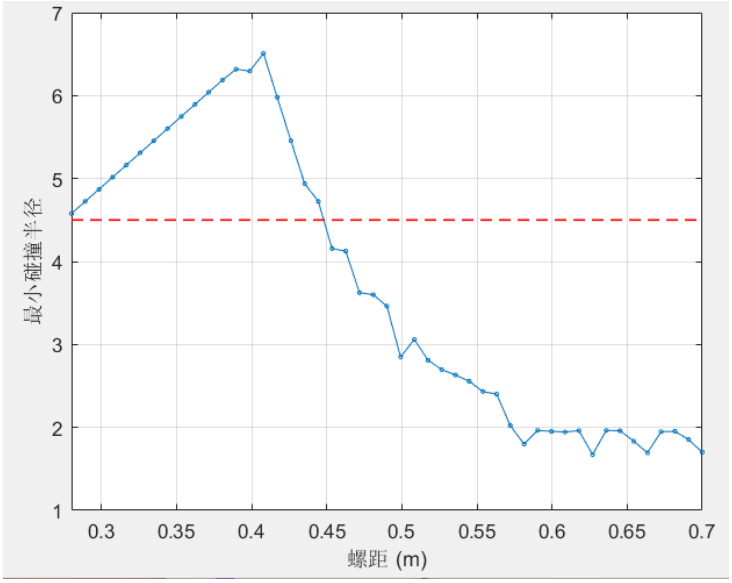


图 6　b 随 L 变化的曲线 (粗步长)

然而，与预期不同的是，由于板凳龙表演整体的复杂性以及碰撞的多样性，b 与 L 的关系并非简单的单调变化，而是存在波动。我们观察到 $b = 4.5$ 在 $L = 0.43$ 至 $L = 0.46$ 之间取到，因此接下来将在此区间进行细步长搜索。

- **细步长搜索**：以 0.001m 为步长，在 $L = 0.43$ 与 $L = 0.46$ 之间进行细步长搜索，以精确找到 $b = 4.5$ 时对应的最小螺距 L_m。

5.3.3 结果分析

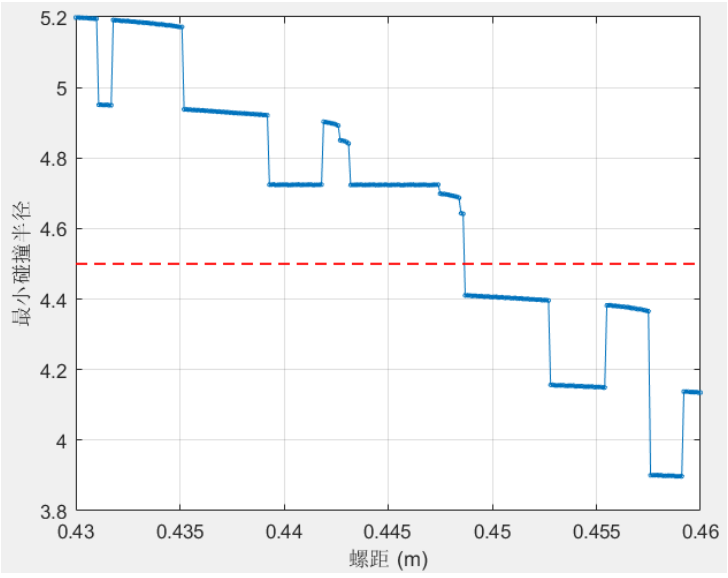


图 7　b 随 L 变化的曲线（细步长）

经过细步长搜索，我们发现 b 并不会恰好取到 4.5。当 $L = 0.4486$ 时，$b = 4.6420$，

但在 $L=0.4487$ 时 b 突变为 4.410，并且之后 b 呈递减趋势，未再回到 4.5 以上。

当 b 高于 4.5（如 $b=4.6420$）时，碰撞发生在龙头进入调头区域之前，说明此时螺距过小，无法使龙头进入调头空间。而当 b 低于 4.5（如 $b=4.410$）时，碰撞发生在调头区域内，说明龙头已经顺利进入调头区域。因此，$L=0.4487$ 是使得板凳间不发生碰撞且龙头能够顺利进入调头区域的最小螺距，也即 $L_m=0.4487$。

5.4 问题四的建立与求解

5.4.1 最佳调头曲线

盘入盘出点的螺线切线 $\vec{\alpha}$:

根据螺线方程 $r=\frac{L}{2\pi}\theta$，利用极坐标转换公式 (1)，可将其表示为直角坐标系下的螺线方程:

$$\begin{cases} x=\frac{L}{2\pi}\theta\cos\theta \\ y=\frac{L}{2\pi}\theta\sin\theta \end{cases} \tag{21}$$

对盘入盘出点的横纵坐标 x 和 y 同时关于 θ 求导，得到:

$$\begin{cases} \dot{x}=\frac{dx}{d\theta}=\frac{L}{2\pi}(\cos\theta-\theta\sin\theta) \\ \dot{y}=\frac{dy}{d\theta}=\frac{L}{2\pi}(\sin\theta+\theta\cos\theta) \end{cases} \tag{22}$$

由于盘入螺线与盘出螺线关于螺线中心呈中心对称，**因此盘入点与盘出点的切线方向是平行的，以下用 $\vec{\alpha}$ 表示其切线的单位向量**:

$$\vec{\alpha}=\frac{(\dot{x}_{in},\dot{y}_{in})}{\sqrt{\dot{x}_{in}^2+\dot{y}_{in}^2}}=\frac{(\dot{x}_{out},\dot{y}_{out})}{\sqrt{\dot{x}_{out}^2+\dot{y}_{out}^2}} \tag{23}$$

其中，(x_{in},y_{in}) 为盘入点坐标，(x_{out},y_{out}) 为盘出点坐标。

盘入盘出点的圆切线 $\vec{\alpha}_0$:

盘入盘出点除了由螺线方程描述，也可以通过调头区域的圆周方程来表示（以盘入点为例）:

$$\begin{cases} x_{in}=R\cos\theta \\ y_{in}=R\sin\theta \end{cases} \tag{24}$$

其中，$R=\frac{D}{2}=\frac{9}{2}$ m 为调头空间的半径。对 x 和 y 同时关于 θ 求导，得到盘入盘出点在圆上的切线单位向量 $\vec{\alpha}_0$（盘入盘出点的圆切线单位向量仍平行）:

$$\begin{cases} \dot{x}_{in}=\frac{dx}{d\theta}=-R\sin\theta \\ \dot{y}_{in}=\frac{dy}{d\theta}=R\cos\theta \end{cases} \tag{25}$$

$$\vec{\alpha_0} = (-\sin\theta, \cos\theta) \tag{26}$$

令 $\vec{\alpha}$ 与 $\vec{\alpha_0}$ 之间的夹角为 β，结合问题三的求解过程，我们可以得到：

$$\beta = arccos\,\vec{\alpha} \cdot \vec{\alpha_0} = 0.0601\ rad \tag{27}$$

因此，为了简化问题，我们可以认为在调头时，盘入盘出点在螺线上的切线与在圆上的切线相同，以下不作区分，均用 $\vec{\alpha}$ 表示。实际上盘入盘出点的切线也相互平行，并且在我们下面讨论的过程中圆切线没有特殊性，因此可以忽略。

圆弧调整方案——调整大圆与小圆的调整半径比 k:

依据题目要求，我们需要找到同时满足 2:1 半径比、盘入盘出点中心对称，并满足各部分相切条件的最小调头曲线。为了符合题目要求并实现最小调头路径，**舞龙队应在刚进入调头空间时立即进入圆弧路段进行调头，以避免浪费不必要的调头空间**。

然而，当上述条件同时满足时，调头曲线长度为定值，无法进一步优化。因此，我们尝试改变大圆与小圆的半径比，将其从 2:1 放宽为 k，以探究是否能够缩短调头曲线，并找到最小的调头曲线长度。

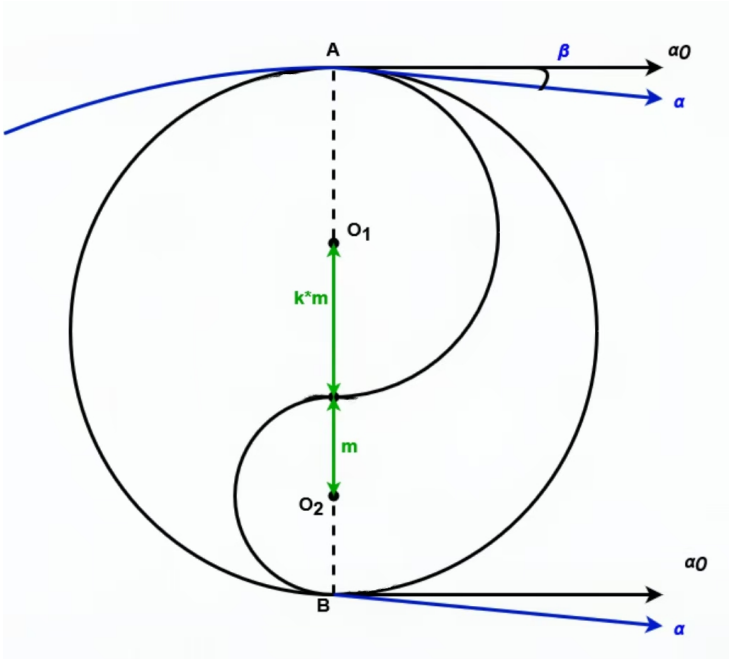


图 8　半径比调整方案

如图 8所示，假设从盘入点开始经过的第一个大圆弧的半径为 km，第二个经过的小圆弧的半径为 m，则大圆与小圆的半径比为 k。由于盘入点 A 和盘出点 B 是调整圆上的圆切点，且关于圆心对称，因此 O_1、O_2、A 和 B 四点共线，且线段 AB 为圆的直径 D。因此，满足以下关系式：

$$D = 2km + 2m = 9 \tag{28}$$

总调头弧长 S 可表示为:

$$S = \pi km + \pi m = (k+1)m\pi = \frac{D}{2}\pi = 4.5\pi \tag{29}$$

由此可见，在保持盘入点与盘出点关于螺线中心对称的条件下，总弧长 S 恒定为 4.5π，并且与大小圆的半径比 k 无关。因此，通过调整半径比并不能实现调头曲线变短的目标。

尽管将半径比例从固定的 2:1 调整为可变的 k，我们仍无法满足题目中“最小调头路径”的要求。**其根本原因在于，最小化调头路径（即调头路径可变）与盘入盘出点对称这两个条件相互不兼容**。只要这两个条件同时存在，无论如何调整半径比 k，都无法实现路径最小化。

通过查阅有关“板凳龙”的资料 [2]，我们发现，实际生活中的舞龙队伍盘入点与盘出点并非完全对称，而是存在一定的偏移。因此，为了尽可能减小调头曲线的弧长，可以放松盘入盘出点对称的约束，允许盘出点在调头区域的圆周上一定范围内自由调整，从而在保持相切条件的前提下，最小化调头曲线的弧长。

圆弧调整方案——自由调整盘出点位置:

实际上，我们不仅可以放松盘入盘出点的对称条件，还可以放宽大圆与小圆的半径比例，以便找到最小的调头路径。

如图 9所示，假设调头时首次经过的大圆与第二次经过的小圆的半径比为 k，盘入点 A 和盘出点 B 之间的夹角 $\angle AOB = \frac{\pi}{2} < \omega < \pi$（如果角度过小，则通过小圆弧时容易发生碰撞）。记大圆的半径为 r_1，即 $AO_1 = r_1$，小圆的半径为 r_2，即 $BO_2 = r_2$，两圆的相切点为 C，调头区域的半径为 $R = \frac{D}{2} = 4.5$。则有:

$$\frac{r_1}{r_2} = k \tag{30}$$

$$\begin{cases} OO_1 = R - r_1 \\ OO_2 = R - r_2 \\ O_1O_2 = r_1 + r_2 \end{cases} \tag{31}$$

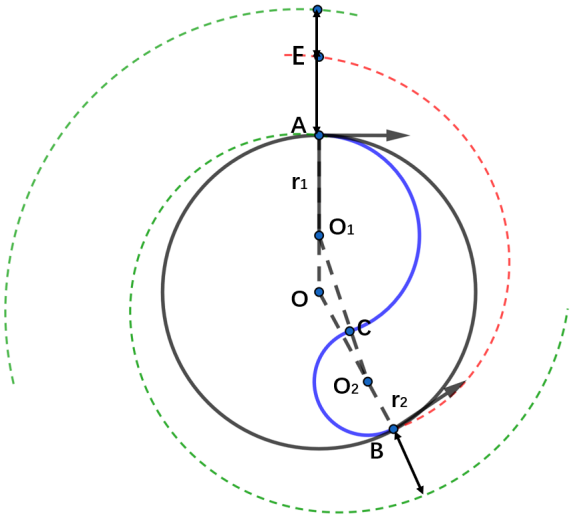


图 9　自由调整盘出点方案

1. **首先，我们确定 ω 的范围**：盘出点位置的选取受到板凳之间不能碰撞的约束。如图 9 所示，当盘出螺线达到 E 点并与盘入点 A 平行时，需要确保盘出螺线既不会与盘入螺线相撞，也不会与外侧的盘入螺线相撞。因此，盘出点与盘入点之间的距离 $EA = \frac{\omega}{2\pi} \cdot 1.7$ 必须小于第三问中计算出的最小螺距 $L_m = 0.45$。同时，盘出螺线还需保证 E 点与其外侧螺线的间距大于最小螺距，即 $\frac{\omega}{2\pi} \cdot 1.7 < 1.7 - 0.45$。由于 $\omega < \pi$ 自然满足该最大值约束。

   此外，盘出点 B 也不能与盘入螺线相撞，因此 $\frac{2\pi-\omega}{2\pi} \cdot 1.7 > 0.45$（同样自然满足条件）。因此我们得到 ω 的范围：

$$0.5294\pi < \omega < \pi \tag{32}$$

2. **其次，求解 r_1 和 r_2**：利用**余弦定理**（公式 (33)）和公式 (30)，可以解三角形 $\triangle O_1 O_2 O$，从而求得 r_1 和 r_2 的值。

$$\cos\omega = \frac{(R-r_1)^2 + (R-r_2)^2 - (r_1+r_2)^2}{2\,(R-r_1)\cdot(R-r_2)} \tag{33}$$

   当盘出点位于圆周左侧时过程也是同理，如图 10所示我们只需将图 9旋转即可。由于在判断 ω 的范围时，需要同时考虑盘出螺线在盘入盘出点处不与盘入螺线发生碰撞，因此 ω 所受到的约束条件相同。由此推导出的 ω 范围与最短调头半径对应的距离也是一致的。盘出点在左侧或右侧不影响我们的求解，故在文中我们仅以盘出点在右侧为例。

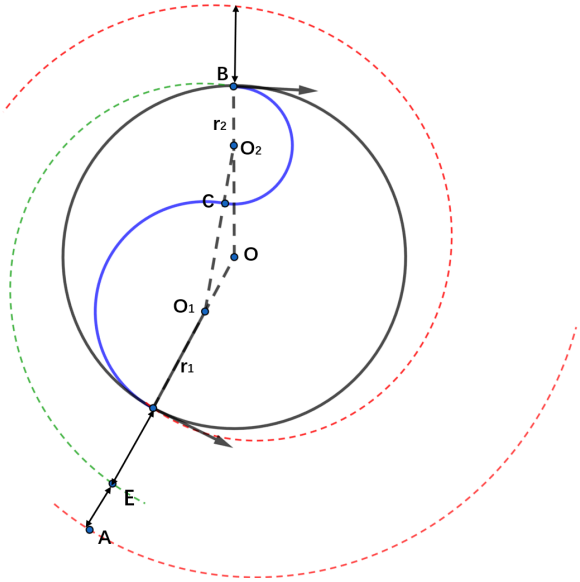


图 10　盘出点在左侧示意图

3. **然后，求解 ω_1 和 ω_2**：根据图中的几何关系，容易得到公式 (34)，并利用 $\triangle O_1O_2O$ 中的余弦定理得到公式 (35)。

$$\begin{cases} \omega_1 = \pi - \angle O_2O_1O \\ \omega_2 = \pi + \angle O_1O_2O \\ \angle O_1O_2O + \angle O_2O_1O + \omega = \pi \end{cases} \tag{34}$$

$$\cos\angle O_2O_1O = \frac{(R-r_1)^2 + (r_1+r_2)^2 - (R-r_2)^2}{2\,(R-r_1)\cdot(r_1+r_2)} \tag{35}$$

由于 $\angle O_2O_1O$ 为锐角，因此可以通过 $\angle O_2O_1O = \arccos\left(\cos\angle O_2O_1O\right)$ 反解出 $\angle O_2O_1O$，然后利用公式 (34) 求得 ω_1 和 ω_2。

4. **最后，求得调头曲线弧长 S**：

$$\begin{aligned} S &= r_1\cdot\omega_1 + r_2\cdot\omega_2 \\ &\triangleq S(k,\omega) \end{aligned} \tag{36}$$

取不同的 k,ω，对调头曲线作图。如图 11所示，我们发现，k 与 S 成负相关，ω 与 S 成正相关。随着 k 增大，ω 减小，S 减小。

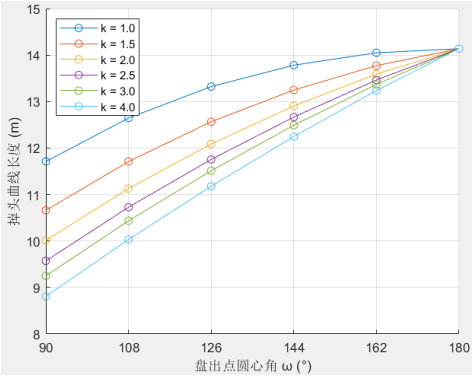


图 11　k、ω 与 S 的关系图

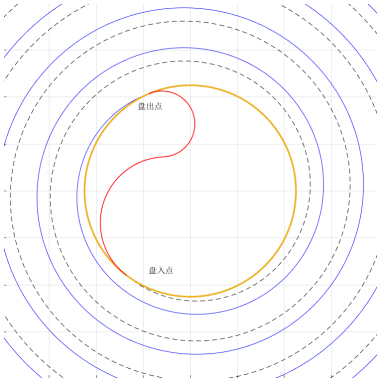


图 12　$k=2$，$\omega=120°$ 时的调头路径

因此，为了取得最小的 S，我们只需要对 k 取最大，然后对 ω 取最小即可。为了简便且与题目条件一致，我们选取 $k=2$，且 $\omega=0.5294\pi$，此时有**弧长最小值为** $S=10.36$。

然而，考虑到实际情况，$\omega=0.5294\pi=95.3°$ 意味着舞龙队盘入盘出点几乎是 $90°$ 的，这不但在观赏性上大打折扣并且由于小圆半径较小，**如果出现失误则很容易发生碰撞**。因此，我们对 ω 进行调整，选取 $\omega=\frac{2}{3}\pi$。此时 $S=11.78$，与最小弧长差距不大但保证了一定的表演容错率，并大大提高了观赏性。因此，在下面位置与速度的确定部分为，**我们也仍旧选择** $k=2$，$\omega=\frac{2}{3}\pi$，$S=11.78$ **进行计算**。

5.5 调头位置与轨迹

我们仍旧沿用第一问的思路，先判断出龙头在调头时的位置，然后依据板凳连接方式推算出其余把手的位置。

- **Step 1: 龙头位置的确定**

  当龙头行驶在上文算出的最佳调头曲线上时，通过计算龙头行驶的曲线长度，我们能够将其转化为龙头此时的坐标位置 $(x_{0,t},y_{0,t})$。

- **Step 2: 各节点位置的确定**

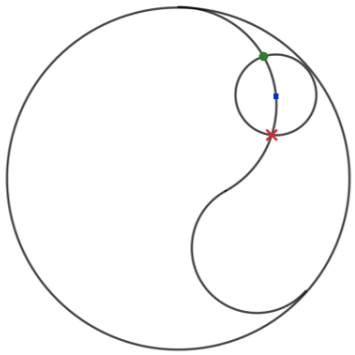
  我们对龙头与龙身节点的位置进行分类讨论，从而能够依据龙头位置递推出其余龙身位置。

  1. 龙头在调头曲线上，判断第一节龙身把手的位置。以龙头为圆心 $(x_{0,t},y_{0,t})$，以板凳把手前后间距 h_i 为半径作圆。如图 13a所示，此时圆与调头曲线的交点即为第一节龙身把手的坐标。然而，其交点不止一个，我们取四个交点中进过螺线长度小于龙头螺线长度的点为所要的第一节龙身前把手的位置。然而，如图 13b所示，由于第一节龙身把手可能在盘入曲线上，因此此时上述方法没有可取的解。因此，我们要找出龙头所成圆与盘入曲线的交点。该交点可能不止一个。与盘入点连线的弧长较短的交点为我们所要的第一节龙身前把手的位置。
  2. 龙头在盘出曲线上，判断第一节龙身把手的位置。如图 13c所示仍旧以龙头为圆

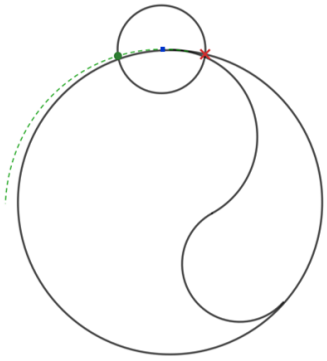
心把手间距为半径作圆。找到该圆与盘出曲线的交点位置，如果该交点距螺线中心的距离超过调头区域半径 4.5 m，则可认为龙身第一节把手也在盘出曲线之上，且为螺线长度小于龙头螺线长度的点。反之，如图 13d所示如果其小于 4.5 m，则说明交点仍在调头区域之内。此时我们取调头曲线与龙头为圆心的圆的交点，该点即为第一节龙身前把手的位置。

得到第一节龙身把手后，依据上述思路便可以得到任意一节把手的位置。

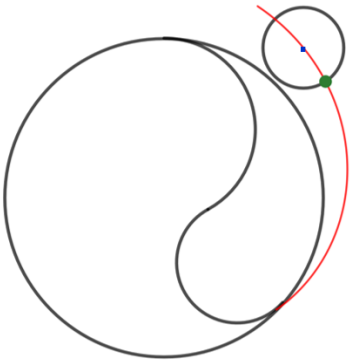
- **Step 3: 各把手速度的确定**依据第一问的思路，取一段极小的 $\Delta t = 0.05$，观察把手位移，运用瞬时速度的定义即可求出每个把手的瞬时速度。



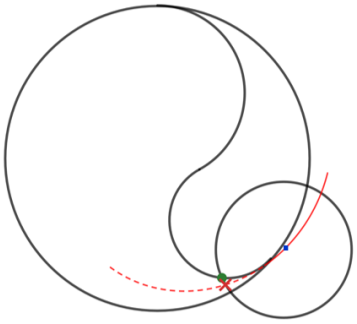
(a) 龙头与第一节把手在调头曲线上



(b) 龙头在调头曲线上与第一节把手在盘入螺线上



(c) 龙头与第一节把手在盘出螺线上



(d) 龙头在盘出螺线上与第一节把手在调头曲线上

图 13　龙头把手与龙身第一节把手的位置情况

通过 MATLAB 求解，我们计算出了以调头时刻为 0s，前后 100s 舞龙队各把手的的位置和速度。图 14给出了 30s 时刻舞龙队的示意图，其中蓝色节点表示把手位置，绿色曲线表示盘入曲线，黄色表示调头曲线，紫色表示盘出曲线。表 5和表 6给出了部分把手在部分时刻的位置和速度，更多详细结果见`result4.xlsx` 文件。

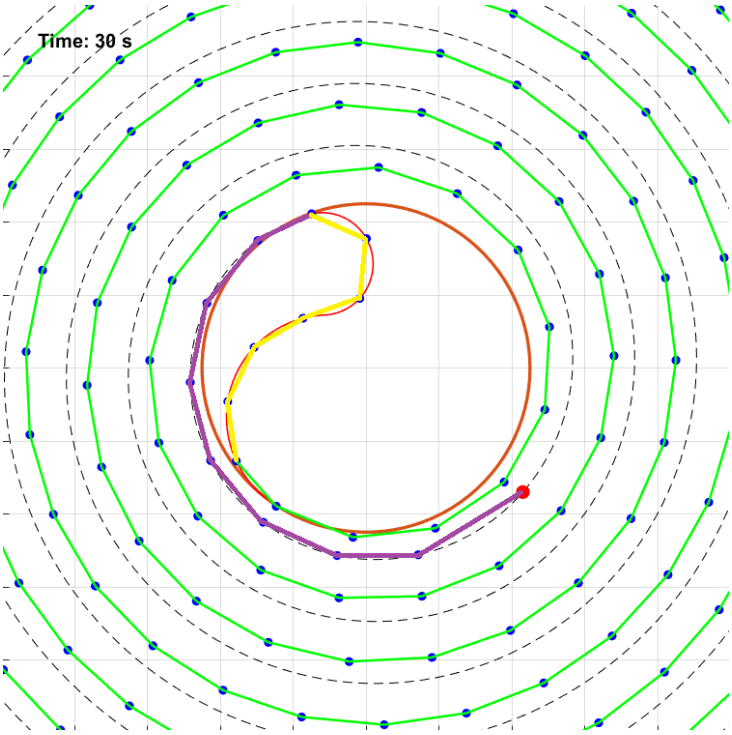


图 14　30s 时刻舞龙队示意图

表 5　调头时部分把手的位置

	-100 s	-50 s	0 s	50 s	100 s
龙头 x (m)	7.777963	6.608259	-2.711723	-5.767108	-8.204490
龙头 y (m)	3.717325	1.899035	-3.591191	2.765050	-0.793191
第 1 节龙身 x (m)	6.209154	5.366804	-0.063359	-0.871311	-7.710661
第 1 节龙身 y (m)	6.108650	4.475542	-4.670900	-0.080943	2.575570
第 51 节龙身 x (m)	-10.608087	-3.629785	2.460130	7.174883	5.037430
第 51 节龙身 y (m)	2.831323	-8.963870	-7.778098	-4.279989	1.112470
第 101 节龙身 x (m)	-11.922699	10.125879	3.008327	-0.306622	-9.226132
第 101 节龙身 y (m)	-4.802542	-5.972098	10.108593	-11.463665	-1.812518
第 151 节龙身 x (m)	-14.351012	12.974836	-7.002935	6.233627	3.278327
第 151 节龙身 y (m)	-1.981167	-3.810190	10.337388	12.422913	11.804954
第 201 节龙身 x (m)	-11.953061	10.522636	-6.872996	-12.337629	-13.520206

	-100 s	-50 s	0 s	50 s	100 s
第 201 节龙身 y (m)	10.566869	-10.807305	12.382527	10.130740	-5.383306
龙尾 (后) x (m)	-1.010884	0.189634	-1.933455	12.077595	12.938502
龙尾 (后) y (m)	-16.527586	15.720593	-14.713154	-11.666581	8.455900

表 6　调头时部分把手的速度

	-100 s	-50 s	0 s	50 s	100 s
龙头 (m/s)	0.999999	0.999998	0.999995	0.999997	0.999998
第 1 节龙身 (m/s)	0.999903	0.999760	0.998684	1.000000	0.999825
第 51 节龙身 (m/s)	0.999345	0.998641	0.995137	1.000000	1.000000
第 101 节龙身 (m/s)	0.999090	0.998248	0.994452	1.000000	1.000000
第 151 节龙身 (m/s)	0.998944	0.998047	0.994160	1.000000	1.000000
第 201 节龙身 (m/s)	0.998849	0.997925	0.993998	1.000000	1.000000
龙尾 (后) (m/s)	0.998817	0.997886	0.993948	1.000000	1.000000

5.6 问题五的建立与求解

5.6.1 把手间的速度成比例

由于舞龙队的各个把手只能在给定的螺线上进行，因此任意时刻整个舞龙队的位置都是确定的。我们考虑单位位移的变化。当龙头移动单位距离 dl 时，由于整个舞龙队“首尾相接”排列紧密，整个图形在几何上是固定的（即只要给出龙头的位置，整个舞龙队的位置确定，不存在其他解。）因此各个把手的位移 dx 也确定。dl 与 dx 成比例。因此龙头速度 $v_{0,t}$ 与龙身或龙尾把手的速度 $v_{i,t}$ 也成比例，我们有：

$$\begin{aligned} dl &= v_{0,t} \cdot dt \\ v_{i,t} &= \frac{dx}{dt} \\ &= \frac{dx}{dl} \cdot v_{0,t} \\ &= \eta_{i,t} \cdot v_{0,t} \end{aligned} \tag{37}$$

其中，$\eta_{i,t}$ 为 t 时刻 i 号把手的速度 $v_{i,t}$ 与龙头前把手速度 $v_{0,t}$ 的比。

5.6.2 遍历求解

基于上述分析，为了找到使得各个舞龙队把手速度不超过 2m/s 时最大的龙头速度，我们只需要保证任何时间点舞龙队最快的把手的速度低于 2m/s 即可。也即找到各个舞龙队把手与龙头把手最大的速度比 η_{max} 即可。我们采用遍历的思想，利用问题四中的方法与程序，对进入调头区域前后 100s 的各个时刻，求出所有的 $v_{i,t}$，并将其与 $v_{0,t}$ 做比，记录下所有的 $\eta_{i,t}$，并找出该时刻最大的 $\eta_{m,t}$。经过对所有时刻的遍历，我们找出所有 $\eta_{m,t}$，对其取最大值可得到我们所期望的 η_{max}。

$$\eta_{max} = \max_{t} \{\eta_{m,t}\} \tag{38}$$

当找到 η_{max} 后，我们可以计算出龙头速度 $v_{0,t}$:

$$v_{0,t} = v_{0,0} = \frac{2}{\eta_{max}} \tag{39}$$

最终，我们解得 $\eta_{max} = 1.0256$，$v_{0,t} = 1.9500$。

六、 模型的评价与改进

6.1 模型的优点

1. 使用变步长搜索，大大简化了代码，运行时间短、运行效率高。
2. 大量基于解析几何建模的代码，代码可调试性强，结果计算较为精确。
3. 丰富的可视化效果，对舞龙队运动过程制作了动画，结果可读性强。
4. 大量可调整的参数，如计算速度时用到的时间偏移量 Δt，判定位置关系时用到的容忍限度 $tolerance$ 等，通过调整参数能够得到不同精度的结果，代码可调节性强。

6.2 模型的缺点

1. 在解析几何中，需要多次求解复杂的超越方程，无法用一般代数几何方法求解，而在求数值解的过程中容易出现报错或无法求解的情况，代码的整体鲁棒性较弱。
2. 几何模型过于复杂，微小的扰动可能会对图形造成巨大的波动，某些结论无法用严格的数学语言证明，因此只能通过增加搜索量来减少复杂模型带来的影响。

6.3 模型的改进

1. 在仿真计算时可以适当减小时间间隔，增加数据点，以获得更连续的运动轨迹和队伍姿态。
2. 对代码中解方程的部分，可以建立数值保护机制，利用科学的计算方法，确保能够得到合理的解。

七、 参考文献与引用

参考文献

[1] 百度百科，板凳龙，
https://baike.baidu.com/item/ /24581696?fr=ge_ala

[2] spiderlion. 板凳龙实况录像 [EB/OL]. 哔哩哔哩, 2024-02-17[2024-09-07]. https://b23.tv/wnf5KDO

[3] 邵一恒. 基于数学软件的阿基米德螺线切线计算与分析. 新课程 (中), 2018, 01:118. ISSN 1673-2162.

附录

1 文件列表

文件名	功能描述
result1.xlsx	问题一结果
result2.xlsx	问题二结果
result4.xlsx	问题四结果
q1.m	问题一程序代码
q2.m	问题二程序代码
q3.m	问题三程序代码
q4_route.m	问题四程序代码 1
q4_before.m	问题四程序代码 2
q4_after.m	问题四程序代码 3
q4_before_delta.m	问题四程序代码 4
q4_after_delta.m	问题四程序代码 5
q4_show.m	问题四程序代码 6
q5.m	问题五程序代码
generating_position.m	函数文件 1
if_coordinates.m	函数文件 2
calc_collision_r.m	函数文件 3
find_circle_intersections.m	函数文件 4
position2route.m	函数文件 5
route2position.m	函数文件 6
q4_xy_before.mat	问题四结果数据 1
q4_xy_after.mat	问题四结果数据 2
q4_xy_before_delta.mat	问题四结果数据 3
q4_xy_after_delta.mat	问题四结果数据 4
q4_v.mat	问题四结果数据 5

2 代码

q1.m

```
clear
l = 0.55; % 螺距
d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
a = l / (2 * pi); % 螺线参数
v = 1; % 速度
theta_0 = 32*pi; % 初始角度

% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));

% 初始螺线长度
l_theta_0 = l_theta(theta_0);

% 时间步长
t_max = 100;
dt = 1;
delta_t = 0.01; % 用于计算瞬时速度的时间增量

% 初始化存储数组，448行 (x,y 交替)，100列 (t_max)
result_xy = zeros(448, t_max+1);
result_xy_deltax = zeros(448, t_max+1); % 用于存储delta_t后的位置
result_v = zeros(224, t_max+1); % 用于存储每个把手的瞬时速度

% 初始化极坐标图
polar_axes = polaraxes;
hold on;
theta_vals = linspace(0, theta_0, 1000);
r_vals = a * theta_vals;
polarplot(polar_axes, theta_vals, r_vals, 'k--'); % 绘制参考螺线

% 初始化龙头轨迹
x_1_history = []; % 存储第一个把手的轨迹
x_1 = [a * theta_0, theta_0]; % 龙头的极坐标
x_cartesian = [x_1(1) * cos(x_1(2)), x_1(1) * sin(x_1(2))]; % 直角坐标

% 逐步计算时间t时刻的螺线长度、各个把手的位置及其极坐标
for t = 0:dt:t_max
    l_t = l_theta_0 - v * t; % t时刻螺线长度
    theta_t = fzero(@(theta) l_theta(theta) - l_t, theta_0); % 利用螺线长度求解 theta_t
    x_t = [a * theta_t, theta_t]; % 当前时间 t 时的极坐标

    % 记录龙头轨迹
    x_1_history = [x_1_history; x_t(2), x_t(1)];

    % 清除上一帧的把手和连接线
```

```
cla(polar_axes);

% 绘制参考螺线
polarplot(polar_axes, theta_vals, r_vals, 'k--');

% 更新并绘制龙头位置（红色点）和轨迹
if size(x_1_history, 1) > 1
    polarplot(polar_axes, x_1_history(:, 1), x_1_history(:, 2), 'w', 'LineWidth', 1.5); %
        红色轨迹
end
polarplot(polar_axes, x_t(2), x_t(1), 'ro', 'MarkerFaceColor', 'r'); % 红色点标记龙头

% 计算当前把手的位置并存储
x_i = x_t; % 当前把手位置
result_xy(1, t+1) = x_t(1) * cos(x_t(2)); % x 坐标
result_xy(2, t+1) = x_t(1) * sin(x_t(2)); % y 坐标

% 计算 delta_t 后的螺线长度
l_t_deltat = l_theta_0 - v * (t+delta_t); % delta_t 时间后的螺线长度
% 计算 delta_t 后的位置
theta_t_deltat = fzero(@(theta) l_theta(theta) - l_t_deltat, delta_t + theta_0); % 计算新的
    theta
x_t_deltat = [a * theta_t_deltat, theta_t_deltat];
x_i_deltat = x_t_deltat;
result_xy_deltax(1, t+1) = x_t_deltat(1) * cos(x_t_deltat(2)); % x 坐标
result_xy_deltax(2, t+1) = x_t_deltat(1) * sin(x_t_deltat(2)); % y 坐标
% 计算第一个把手的瞬时速度
delta_x_1 = sqrt((x_t_deltat(1) * cos(x_t_deltat(2)) - x_t(1) * cos(x_t(2)))^2 + ...
              (x_t_deltat(1) * sin(x_t_deltat(2)) - x_t(1) * sin(x_t(2)))^2);
result_v(1, t+1) = delta_x_1 / delta_t; % 存储第一个把手的瞬时速度

for i = 2:224
    d = (i == 2) * d1 + (i > 2) * d2; % 判断使用 d1 还是 d2

    % 计算 beta_i
    beta_i = fzero(@(beta) (x_i(1)^2 + (x_i(1) + a * beta)^2 - d^2) - 2 * x_i(1) * (x_i(1) +
        a * beta) * cos(beta), 0.1);

    % 计算第 i 个把手的极坐标
    prev_x_i = x_i; % 保存上一个把手的位置
    x_i = [x_i(1) + a * beta_i, x_i(2) + beta_i];
    x_i_cartesian = [x_i(1) * cos(x_i(2)), x_i(1) * sin(x_i(2))];
    prev_x_i_cartesian = [prev_x_i(1) * cos(prev_x_i(2)), prev_x_i(1) * sin(prev_x_i(2))];

    % 绘制把手连接线
    polarplot(polar_axes, [prev_x_i(2), x_i(2)], [prev_x_i(1), x_i(1)], 'g-', 'LineWidth',
        1);
```

```
        % 用蓝色点表示把手的位置
        polarplot(polar_axes, x_i(2), x_i(1), 'bo', 'MarkerFaceColor', 'b');

        result_xy(2*i-1, t+1) = x_i_cartesian(1); % x 坐标（奇数行）
        result_xy(2*i, t+1) = x_i_cartesian(2); % y 坐标（偶数行）

        % 计算 beta_i_deltat
        beta_i_deltat = fzero(@(beta) (x_i_deltat(1)^2 + (x_i_deltat(1) + a * beta)^2 - d^2) - 2
            * x_i_deltat(1) * (x_i_deltat(1) + a * beta) * cos(beta), 0.1);

        % 计算 delta_t 后的位置
        x_i_deltat = [x_i_deltat(1)+a*beta_i_deltat,x_i_deltat(2)+beta_i_deltat];
        x_i_deltat_cartesian = [x_i_deltat(1) * cos(x_i_deltat(2)), x_i_deltat(1) *
            sin(x_i_deltat(2))];

        % 存储 delta_t 后的位置
        result_xy_deltax(2*i-1, t_max+1-t) = x_i_deltat_cartesian(1); % x 坐标（奇数行）
        result_xy_deltax(2*i, t+1-t) = x_i_deltat_cartesian(2); % y 坐标（偶数行）

        % 计算瞬时速度
        delta_x = sqrt((x_i_deltat_cartesian(1) - x_i_cartesian(1))^2 + (x_i_deltat_cartesian(2)
            - x_i_cartesian(2))^2);
        result_v(i, t+1) = delta_x / delta_t; % 存储瞬时速度
    end

    % 显示时间
    text(1/4*pi, 2* max(r_vals), ['t = ' num2str(t),'s'], 'HorizontalAlignment', 'center',
        'FontSize', 12);

    % 暂停以展示动画效果
    pause(0.001);
end

hold off;
```

q2.m

```
l = 0.55; % 螺距
d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
a = l / (2 * pi); % 螺线参数
v = 1; % 速度
theta_0 = 32 * pi; % 初始角度

% 搜索起点为300s，时间步长为10s
t_start = 300;
delta = 10;
```

```
% 搜索终点为440s，因为螺线到A点最多只有442米
t_end = 440;
result_xy_10 = generating_position(t_start, t_end, delta, l);
n = (t_end - t_start) / delta + 1;
flag_lst = zeros(1, n);

% 循环遍历每个时间步
for k = 1:n
    flag = 0; % 初始化当前时间步的标志位

    % 遍历前32个把手
    for i = 1:32
        lst = result_xy_10(:, k)';

        % 获取第i个把手的坐标 (x1, y1), (x2, y2)
        x1 = lst(2*i-1);
        y1 = lst(2*i);
        if i < 224
            x2 = lst(2*i+1);
            y2 = lst(2*i+2);
        end

        % 遍历后续把手，判断是否可能发生碰撞
        for j = i+2:223
            % 获取第j个把手的坐标 (x3, y3), (x4, y4)
            x3 = lst(2*j-1);
            y3 = lst(2*j);
            if j < 224
                x4 = lst(2*j+1);
                y4 = lst(2*j+2);
            end

            % 计算四点之间的最小距离
            d1 = sqrt((x1 - x3)^2 + (y1 - y3)^2);
            d2 = sqrt((x1 - x4)^2 + (y1 - y4)^2);
            d3 = sqrt((x2 - x3)^2 + (y2 - y3)^2);
            d4 = sqrt((x2 - x4)^2 + (y2 - y4)^2);
            d_min = min([d1, d2, d3, d4]);

            % 如果最小距离小于1，检查是否碰撞
            if d_min < 1
                flag = if_coordinates(x1, y1, x2, y2, x3, y3, x4, y4);
            end

            % 如果发生碰撞，跳出循环
            if flag
                break
```

```
            end
        end

        % 如果发生碰撞，跳出外层循环
        if flag
            break
        end
    end

    % 存储当前时间步的碰撞标志位
    flag_lst(k) = flag;
end

time_lst = linspace(t_start, t_end, n);
for i = 1:n
    if flag_lst(i)
        t_end = time_lst(i);
        break
    end
end
t_start = t_end - 10;

% 新的搜索范围，步长为0.1
delta_fine = 0.1; % 细致搜索的步长为0.1
result_xy_fine = generating_position(t_start, t_end, delta_fine, l);
n_fine = (t_end - t_start) / delta_fine + 1;
flag_lst_fine = zeros(1, n_fine);

% 循环遍历每个时间步，进行精细搜索
for k = 1:n_fine
    flag = 0; % 初始化当前时间步的标志位

    % 遍历前32个把手
    for i = 1:32
        lst = result_xy_fine(:, k)';

        % 获取第i个把手的坐标 (x1, y1), (x2, y2)
        x1 = lst(2*i-1);
        y1 = lst(2*i);
        if i < 224
            x2 = lst(2*i+1);
            y2 = lst(2*i+2);
        end

        % 遍历后续把手，判断是否可能发生碰撞
        for j = i+2:223
            % 获取第j个把手的坐标 (x3, y3), (x4, y4)
```

```
            x3 = lst(2*j-1);
            y3 = lst(2*j);
            if j < 224
                x4 = lst(2*j+1);
                y4 = lst(2*j+2);
            end

            % 计算四点之间的最小距离
            d1 = sqrt((x1 - x3)^2 + (y1 - y3)^2);
            d2 = sqrt((x1 - x4)^2 + (y1 - y4)^2);
            d3 = sqrt((x2 - x3)^2 + (y2 - y3)^2);
            d4 = sqrt((x2 - x4)^2 + (y2 - y4)^2);
            d_min = min([d1, d2, d3, d4]);

            % 如果最小距离小于1，检查是否碰撞
            if d_min < 1
                flag = if_coordinates(x1, y1, x2, y2, x3, y3, x4, y4);
            end

            % 如果发生碰撞，跳出循环
            if flag
                i;
                j;
                break
            end
        end

        % 如果发生碰撞，跳出外层循环
        if flag
            break
        end
    end

    % 存储当前时间步的碰撞标志位
    flag_lst_fine(k) = flag;
end

% 更新最终时间
time_lst_fine = linspace(t_start, t_end, n_fine);
for i = 1:n_fine
    if flag_lst_fine(i)
        t_end_fine = time_lst_fine(i);
        break
    end
end
```

```
result_xy_final = generating_position(t_end_fine, t_end_fine, 0, 1);
delta_t = 0.05;
result_xy_final_delta = generating_position(t_end_fine + delta_t, t_end_fine + delta_t, 0, 1);
result_v_xy = (result_xy_final_delta - result_xy_final) / delta_t;
result_xy = [];
result_v = [];
for i =1:224
    result_xy = [result_xy; result_xy_final(2*i-1, 1), result_xy_final(2*i, 1)];
    v_x = result_v_xy(2*i-1, 1);
    v_y = result_v_xy(2*i, 1);
    v = sqrt(v_x^2 + v_y^2);
    result_v = [result_v; v];
end

t_end_fine;
temp1 = result_xy_final(1,1);
temp2 = result_xy_final(2,1);
sqrt(temp1 * temp1 + temp2 * temp2);
```

q3.m

```
R = 4.5;
L_upper = 0.70;
L_lower = 0.28;
L_lst = linspace(L_lower, L_upper, 47);
r_lst = [];
for L = L_lst
    r = calc_collision_r(L);
    r_lst = [r_lst, r];
end

L_upper = 0.46;
L_lower = 0.43;
L_lst_fine = L_lower:0.0001:L_upper;
r_lst_fine = [];
for L = L_lst_fine
    r = calc_collision_r(L);
    r_lst_fine = [r_lst_fine, r];
end


% 创建一个新的图形窗口
figure;
% 绘制第一个子图
plot(L_lst, r_lst, 'o-', 'MarkerSize', 2); % 绘制折线
hold on
plot([0.25, 0.75], [4.5, 4.5], 'Color', 'r', 'LineWidth', 1, 'LineStyle', '--')
```

```
xlim([0.28, 0.70])
xlabel('螺距 (m)');     % 设置X轴标签
ylabel('最小碰撞半径');     % 设置Y轴标签
grid on;                % 显示网格

figure
% 绘制第一个子图
plot(L_lst_fine, r_lst_fine, 'o-', 'MarkerSize', 2);
hold on
plot([0.25, 0.75], [4.5, 4.5], 'Color', 'r', 'LineWidth', 1, 'LineStyle', '--')
xlim([0.43, 0.46])
xlabel('螺距 (m)');     % 设置X轴标签
ylabel('最小碰撞半径');     % 设置Y轴标签
grid on;               % 显示网格
```

q4_route.m

```
% 初始化参数
l = 1.7; % 螺距为1.7m
a_temp = l / (2 * pi); % 螺线参数为a
R = 4.5; % 直径为9m
theta = R / a_temp;
tan_angle = 1 / theta;
angle = atan(tan_angle); % 盘入螺线在盘入处的切线与圆的切线存在一个较小的夹角

% 定义参考半径比和旋转角列表
k_lst = [1, 1.5, 2, 2.5, 3, 4]; % 参考半径比
alpha_lst = linspace(pi/2, pi, 6); % 参考旋转角

% 预分配结果矩阵
L_matrix = zeros(length(k_lst), length(alpha_lst));

% 双重循环遍历所有可能的k和alpha组合
for i = 1:length(alpha_lst)
    alpha = alpha_lst(i);
    for j = 1:length(k_lst)
        k = k_lst(j);
        a_temp = (1 + cos(alpha)) * k;
        b_temp = (1 - cos(alpha)) * R * (k + 1);
        c_temp = (cos(alpha) - 1) * R^2;
        delta = b_temp^2 - 4 * a_temp * c_temp;
        x = (-b_temp + sqrt(delta)) / (2 * a_temp);
        y = k * x;
        side1 = R - y;
        side2 = R - x;
        side3 = x + y;
        angle1 = pi - acos((side1^2 + side3^2 - side2^2) / (2 * side1 * side3));
```

```
        angle2 = pi + angle1 - alpha;
        L_matrix(j, i) = angle1 * y + angle2 * x; % 将结果存入矩阵
    end
end

L_matrix(:, 6) = pi * R;

% 转换 alpha 从弧度到度
alpha_lst_deg = alpha_lst * 180 / pi;

% 创建折线图
figure;
hold on;
for i = 1:length(k_lst)
    plot(alpha_lst_deg, L_matrix(i, :), '-o', 'DisplayName', sprintf('k = %.1f', k_lst(i)));
end

% 设置图例和标签
legend('Location', 'northwest');
xticks(alpha_lst_deg);
xticklabels(arrayfun(@(alpha) sprintf('%.0f', alpha), alpha_lst_deg, 'UniformOutput', false));
xlabel('盘出点圆心角  (°)');
ylabel('掉头曲线长度 (m)');

% 显示网格
grid on;
hold off;

% 最短掉头曲线的计算
% 定义参考半径比和旋转角
L_min = 0.45; % q3中的最小螺距
k = 2; % 用户输入的半径比
alpha = 2 * pi * L_min / l;

% 掉头曲线的各项参数
a_temp = (1 + cos(alpha)) * k;
b_temp = (1 - cos(alpha)) * R * (k + 1);
c_temp = (cos(alpha) - 1) * R^2;
delta = b_temp^2 - 4 * a_temp * c_temp;
x = (-b_temp + sqrt(delta)) / (2 * a_temp); % 较小圆弧的半径
y = k * x; % 较大圆弧的半径
side1 = R - y;
side2 = R - x;
side3 = x + y;
angle1 = pi - acos((side1^2 + side3^2 - side2^2) / (2 * side1 * side3));
angle2 = pi + angle1 - alpha
```

```
L_shortest = angle1 * y + angle2 * x;

% 参数初始化
k = 2; % 调头圆弧的的半径比
alpha = 2 * pi / 3; % 盘出点的圆心角
R = 4.5; % 掉头区域半径
x_radius = 2.805639223409097; % 调头曲线大圆半径
y_radius = 1.402819611704549; % 调头曲线小圆半径
angle1 = 2.450545798771806; % 半径较大的圆弧的圆心角
angle2 = 3.497743349968404; % 半径较小的圆弧的圆心角
L_best = 11.782050379839399; % 掉头曲线的长度

% 盘入螺线方程:r=a*theta
% 盘出螺线方程:r=a*(theta+alpha)

d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
l = 1.7; % 螺距
a = l / (2 * pi); % 螺线的参数a
theta_0 = 16 * pi;
theta_in = R / a; % 螺线盘入点的极角
theta_out = theta_in - alpha; % 螺线盘出点的极角
x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标
o1 = x_in * (R - x_radius) / R; % 大圆圆心
o2 = x_out * (R - y_radius) / R; % 小圆圆心


% 初始化笛卡尔坐标图
figure;
hold on;
axis equal;
grid on;
% 生成盘入螺线的角度和半径
theta_vals = linspace(theta_in, theta_0, 1000); % 生成螺线的角度
r_vals = a * theta_vals; % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k--', 'Color', 'black'); % 绘制参考螺线

theta_vals = linspace(theta - alpha, theta_0, 1000); % 生成螺线的角度
% 生成盘出螺线的角度和半径
r_vals = a * (theta_vals + alpha); % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k-', 'Color', 'b'); % 绘制参考螺线
```

```
% 设置笛卡尔坐标下的掉头区域
theta_circle = linspace(0, 2*pi, 100); % 0到2*pi生成圆的角度
r_circle = R * ones(size(theta_circle)); % 圆的半径为R
% 将掉头圆从极坐标转换为笛卡尔坐标
[x_circle, y_circle] = pol2cart(theta_circle, r_circle); % 极坐标到笛卡尔坐标的转换
plot(x_circle, y_circle, 'LineWidth', 2); % 绘制掉头区域的圆

% 生成极坐标下的大圆与小圆
% 大圆的坐标
theta_vals_large = linspace(theta_in - angle1, theta_in, 1000);
theta_vals_small = linspace(theta_out - angle2, theta_out, 1000);
x_large_circle = o1(1) + x_radius * cos(theta_vals_large); % 大圆的x坐标
y_large_circle = o1(2) + x_radius * sin(theta_vals_large); % 大圆的y坐标
% 小圆的坐标
x_small_circle = o2(1) + y_radius * cos(theta_vals_small); % 小圆的x坐标
y_small_circle = o2(2) + y_radius * sin(theta_vals_small); % 小圆的y坐标
plot(x_large_circle, y_large_circle, 'r-', 'LineWidth', 1);
plot(x_small_circle, y_small_circle, 'r-', 'LineWidth', 1);
xlim([-8, 8]);
ylim([-8, 8]);
% 在图上添加x_in点的标注
text(x_in(1)+2, x_in(2), '盘入点', 'VerticalAlignment', 'bottom', 'HorizontalAlignment',
    'right');
% 在图上添加x_out点的标注
text(x_out(1)-0.5, x_out(2)-0.3, '盘出点', 'VerticalAlignment', 'top', 'HorizontalAlignment',
    'left');
```

q4_before.m

```
l = 1.7; % 螺距
d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
a = l / (2 * pi); % 螺线参数
v = -1; % 速度
theta_0 = 16.6320; % 初始角度

% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));

% 初始螺线长度
l_theta_0 = l_theta(theta_0);

% 时间步长
t_max = 100;
dt = 1;

% 初始化存储数组，448行 (x,y 交替)，100列 (t_max)
```

```
result_xy = zeros(448, t_max+1);

% 初始化极坐标图
polar_axes = polaraxes;
hold on;
theta_vals = linspace(0, theta_0, 1000);
r_vals = a * theta_vals;
polarplot(polar_axes, theta_vals, r_vals, 'k--'); % 绘制参考螺线

% 初始化龙头轨迹
x_1_history = []; % 存储第一个把手的轨迹
x_1 = [a * theta_0, theta_0]; % 龙头的极坐标
x_cartesian = [x_1(1) * cos(x_1(2)), x_1(1) * sin(x_1(2))]; % 直角坐标

% 逐步计算时间t时刻的螺线长度、各个把手的位置及其极坐标
for t = 0:dt:t_max
    l_t = l_theta_0 - v * t; % t时刻螺线长度
    theta_t = fzero(@(theta) l_theta(theta) - l_t, theta_0); % 利用螺线长度求解 theta_t
    x_t = [a * theta_t, theta_t]; % 当前时间 t 时的极坐标

    % 记录龙头轨迹
    x_1_history = [x_1_history; x_t(2), x_t(1)];

    % 清除上一帧的把手和连接线
    cla(polar_axes);

    % 绘制参考螺线
    polarplot(polar_axes, theta_vals, r_vals, 'k--');

    % 更新并绘制龙头位置（红色点）和轨迹
    if size(x_1_history, 1) > 1
        polarplot(polar_axes, x_1_history(:, 1), x_1_history(:, 2), 'w', 'LineWidth', 1.5); %
            红色轨迹
    end
    polarplot(polar_axes, x_t(2), x_t(1), 'ro', 'MarkerFaceColor', 'r'); % 红色点标记龙头

    % 计算每个把手的位置并连接
    x_i = x_t; % 当前把手位置
    % 逆向存储，第 t_max+1-t 列存储当前时间步的数据
    result_xy(1, t_max+1-t) = x_t(1) * cos(x_t(2)); % x 坐标
    result_xy(2, t_max+1-t) = x_t(1) * sin(x_t(2)); % y 坐标

    for i = 2:224
        d = (i == 2) * d1 + (i > 2) * d2; % 判断使用 d1 还是 d2

        % 计算 beta_i
        beta_i = fzero(@(beta) (x_i(1)^2 + (x_i(1) + a * beta)^2 - d^2) - 2 * x_i(1) * (x_i(1) +
```

```
            a * beta) * cos(beta), 0.1);

        % 计算第 i 个把手的极坐标
        prev_x_i = x_i; % 保存上一个把手的位置
        x_i = [x_i(1) + a * beta_i, x_i(2) + beta_i];
        x_i_cartesian = [x_i(1) * cos(x_i(2)), x_i(1) * sin(x_i(2))];
        prev_x_i_cartesian = [prev_x_i(1) * cos(prev_x_i(2)), prev_x_i(1) * sin(prev_x_i(2))];

        % 绘制把手连接线
        polarplot(polar_axes, [prev_x_i(2), x_i(2)], [prev_x_i(1), x_i(1)], 'g-', 'LineWidth',
            1);
        % 用蓝色点表示把手的位置
        polarplot(polar_axes, x_i(2), x_i(1), 'bo', 'MarkerFaceColor', 'b');

        % 逆向存储，第 t_max+1-t 列存储当前时间步的数据
        result_xy(2*i-1, t_max+1-t) = x_i_cartesian(1); % x 坐标（奇数行）
        result_xy(2*i, t_max+1-t) = x_i_cartesian(2); % y 坐标（偶数行）
    end

    % 显示时间
    text(1/4*pi, 2* max(r_vals), ['t = ' num2str(t),'s'], 'HorizontalAlignment', 'center',
        'FontSize', 12);

    % 暂停以展示动画效果
    pause(0.001);
end

hold off;
result_xy_before = result_xy;
% 存储结果到文件
save('points_data_448x100_reversed.mat', 'result_xy_before');
```

q4_after.m

```
% 参数初始化
k = 2; % 调头圆弧的的半径比
alpha = 2 * pi / 3; % 盘出点的圆心角
R = 4.5; % 掉头区域半径
x_radius = 2.805639223409097; % 调头曲线大圆半径
y_radius = 1.402819611704549; % 调头曲线小圆半径
angle1 = 2.450545798771806; % 半径较大的圆弧的圆心角
angle2 = 3.497743349968404; % 半径较小的圆弧的圆心角
L_best = 11.782050379839399; % 掉头曲线的长度

% 盘入螺线方程:r=a*theta
% 盘出螺线方程:r=a*(theta+alpha)
```

```
d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
l = 1.7; % 螺距
a = l / (2 * pi); % 螺线的参数a
theta_0 = 16 * pi;
theta_in = R / a; % 螺线盘入点的极角
theta_out = theta_in - alpha; % 螺线盘出点的极角
x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标
o1 = x_in * (R - x_radius) / R; % 大圆圆心
o2 = x_out * (R - y_radius) / R; % 小圆圆心


% 初始化笛卡尔坐标图
figure;
hold on;
axis equal;
grid on;
% 生成盘入螺线的角度和半径
theta_vals = linspace(theta_in, theta_0, 1000); % 生成螺线的角度
r_vals = a * theta_vals; % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k--', 'Color', 'black'); % 绘制参考螺线

theta_vals = linspace(theta - alpha, theta_0, 1000); % 生成螺线的角度
% 生成盘出螺线的角度和半径
r_vals = a * (theta_vals + alpha); % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k--', 'Color', 'b'); % 绘制参考螺线

% 设置笛卡尔坐标下的掉头区域
theta_circle = linspace(0, 2*pi, 100); % 0到2*pi生成圆的角度
r_circle = R * ones(size(theta_circle)); % 圆的半径为R
% 将掉头圆从极坐标转换为笛卡尔坐标
[x_circle, y_circle] = pol2cart(theta_circle, r_circle); % 极坐标到笛卡尔坐标的转换
plot(x_circle, y_circle, 'LineWidth', 2); % 绘制掉头区域的圆

% 生成极坐标下的大圆与小圆
% 大圆的坐标
theta_vals_large = linspace(theta_in - angle1, theta_in, 1000);
theta_vals_small = linspace(theta_out - angle2, theta_out, 1000);
x_large_circle = o1(1) + x_radius * cos(theta_vals_large); % 大圆的x坐标
y_large_circle = o1(2) + x_radius * sin(theta_vals_large); % 大圆的y坐标
% 小圆的坐标
x_small_circle = o2(1) + y_radius * cos(theta_vals_small); % 小圆的x坐标
```

```
y_small_circle = o2(2) + y_radius * sin(theta_vals_small); % 小圆的y坐标
plot(x_large_circle, y_large_circle, 'r-', 'LineWidth', 1);
plot(x_small_circle, y_small_circle, 'r-', 'LineWidth', 1);

% 在图上添加x_in点的标注
text(x_in(1), x_in(2), 'x_in', 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

% 在图上添加x_out点的标注
text(x_out(1), x_out(2), 'x_out', 'VerticalAlignment', 'top', 'HorizontalAlignment', 'left');




% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));
l_theta_0 = l_theta(theta_in);
t_lst = 1:100;
% 初始化龙头把手位置
result_xy_after = zeros(448,100);

% 初始化正时间内龙头把手位置
l_lst_after = 1:100;
for t = 1:100
    if t < L_best
        position = route2position(t);
        result_xy_after(1, t) = position(1);
        result_xy_after(2, t) = position(2);
    else
        l_temp = t - L_best + l_theta_0;
        theta_t = fzero(@(theta) l_theta(theta) - l_temp, theta_0);
        theta_temp = theta_t - alpha;
        r_temp = theta_t * a;
        x_temp = r_temp * cos(theta_temp);
        y_temp = r_temp * sin(theta_temp);
        result_xy_after(1, t) = x_temp;
        result_xy_after(2, t) = y_temp;
    end
end




tolerance = 0.1;
% 储存正时刻位置
for t = 1:100
    current_position = result_xy_after(1:2, t)';
    [theta_temp, r_temp] = cart2pol(current_position(1), current_position(2));
```

```
flag = 1;
for i = 2:224
    d = (i == 2) * d1 + (i > 2) * d2; % 判断使用d1还是d2

    % 计算是否在盘出螺线上，若在盘出螺线上，计算极角
    theta_real = r_temp / a - alpha;
    delta_theta = theta_real - theta_temp - fix((theta_real - theta_temp) / (2 * pi)) * 2 *
        pi;


    % 若出现一次不在盘出螺线上，则后续点不会在盘出螺线上，不用考虑盘出螺线的检验
    if abs(delta_theta) > tolerance
        flag = 0;
    end

    if r_temp <= R
        % 计算当前位置对应的路径长度
        l_answer = position2route(current_position);
        points12 = find_circle_intersections(current_position, d, o1, x_radius);
        points34 = find_circle_intersections(current_position, d, o2, y_radius);
        points1 = points12(1, :);
        points2 = points12(2, :);
        points3 = points34(1, :);
        points4 = points34(2, :);
        l1 = position2route(points1);
        l2 = position2route(points2);
        l3 = position2route(points3);
        l4 = position2route(points4);
        l_list = [l1, l2, l3, l4];
        for j = 1:4
            if l_list(j) == -1 || l_list(j) >= l_answer
                l_list(j) = 0;
            end
        end
        if max(l_list) > 0
            x_cartesian = route2position(max(l_list));
            % 存储当前把手的直角坐标到 result
            result_xy_after(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after(2*i, t) = x_cartesian(2); % 第i节龙身y
        else
            % 查找失败
            x_temp = current_position(1);
            y_temp = current_position(2);
            theta_temp = fzero(@(x) (a*x*cos(x)-x_temp)^2+(a*x*sin(x)-y_temp)^2-d^2,theta_in);
            r_temp = a * theta_temp;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after(2*i-1, t) = x_cartesian(1); % 第i节龙身x
```

```
            result_xy_after(2*i, t) = x_cartesian(2); % 第i节龙身y
        end
    elseif r_temp > R && flag
        % 计算 beta_i
        beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
            (r_temp+a*beta) * cos(beta), [0, pi/2]);
        theta_ = theta_real - beta;
        r_ = (theta_ + alpha) * a;
        if r_ >= R
            theta_temp = theta_;
            r_temp = r_;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after(2*i, t) = x_cartesian(2); % 第i节龙身y
        else
            % 计算当前位置对应的路径长度
            l_answer = L_best;
            points12 = find_circle_intersections(current_position, d, o1, x_radius);
            points34 = find_circle_intersections(current_position, d, o2, y_radius);
            points1 = points12(1, :);
            points2 = points12(2, :);
            points3 = points34(1, :);
            points4 = points34(2, :);
            l1 = position2route(points1);
            l2 = position2route(points2);
            l3 = position2route(points3);
            l4 = position2route(points4);
            l_list = [l1, l2, l3, l4];
            for j = 1:4
                if l_list(j) == -1
                    l_list(j) = 0;
                end
            end
            if max(l_list) > 0
                x_cartesian = route2position(max(l_list));
                % 存储当前把手的直角坐标到 result
                result_xy_after(2*i-1, t) = x_cartesian(1); % 第i节龙身x
                result_xy_after(2*i, t) = x_cartesian(2); % 第i节龙身y
            end
        end
    elseif r_temp > R
        % 计算 beta_i
        beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
            (r_temp+a*beta) * cos(beta), [0, pi/2]);
        theta_temp = theta_temp + beta;
        [x_temp, y_temp] = pol2cart(theta_temp, a * theta_temp);
        % 存储当前把手的直角坐标到 result
```

```
            result_xy_after(2*i-1, t) = x_temp; % 第i节龙身x
            result_xy_after(2*i, t) = y_temp; % 第i节龙身y
        end
        current_position(1) = result_xy_after(2*i-1, t);
        current_position(2) = result_xy_after(2*i, t);
        r_temp = sqrt(dot(current_position, current_position));
    end
end
```

q4_after_delta.m

```
% 参数初始化
k = 2; % 调头圆弧的的半径比
alpha = 2 * pi / 3; % 盘出点的圆心角
R = 4.5; % 掉头区域半径
x_radius = 2.805639223409097; % 调头曲线大圆半径
y_radius = 1.402819611704549; % 调头曲线小圆半径
angle1 = 2.450545798771806; % 半径较大的圆弧的圆心角
angle2 = 3.497743349968404; % 半径较小的圆弧的圆心角
L_best = 11.782050379839399; % 掉头曲线的长度

% 盘入螺线方程:r=a*theta
% 盘出螺线方程:r=a*(theta+alpha)

d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
l = 1.7; % 螺距
a = l / (2 * pi); % 螺线的参数a
theta_0 = 16 * pi;
theta_in = R / a; % 螺线盘入点的极角
theta_out = theta_in - alpha; % 螺线盘出点的极角
x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标
o1 = x_in * (R - x_radius) / R; % 大圆圆心
o2 = x_out * (R - y_radius) / R; % 小圆圆心

% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));
l_theta_0 = l_theta(theta_in);
t_lst = 1:100;
% 初始化龙头把手位置
result_xy_after_delta = zeros(448,100);



delta_time = 0.05;
% 初始化正时间内龙头把手位置
l_lst_after = 0.95:1:99.955;
```

```
for t = 1:100
    t_ = l_lst_after(t);
    if t_ < L_best
        position = route2position(t_);
        result_xy_after_delta(1, t) = position(1);
        result_xy_after_delta(2, t) = position(2);
    else
        l_temp = t_ - L_best + l_theta_0;
        theta_t = fzero(@(theta) l_theta(theta) - l_temp, theta_0);
        theta_temp = theta_t - alpha;
        r_temp = theta_t * a;
        x_temp = r_temp * cos(theta_temp);
        y_temp = r_temp * sin(theta_temp);
        result_xy_after_delta(1, t) = x_temp;
        result_xy_after_delta(2, t) = y_temp;
    end
end




tolerance = 0.1;
% 储存正时刻位置
for t = 1:100
    current_position = result_xy_after_delta(1:2, t)';
    [theta_temp, r_temp] = cart2pol(current_position(1), current_position(2));
    flag = 1;
    for i = 2:224
        d = (i == 2) * d1 + (i > 2) * d2; % 判断使用d1还是d2

        % 计算是否在盘出螺线上，若在盘出螺线上，计算极角
        theta_real = r_temp / a - alpha;
        delta_theta = theta_real - theta_temp - fix((theta_real - theta_temp) / (2 * pi)) * 2 *
            pi;


        % 若出现一次不在盘出螺线上，则后续点不会在盘出螺线上，不用考虑盘出螺线的检验
        if abs(delta_theta) > tolerance
            flag = 0;
        end

        if r_temp <= R
            % 计算当前位置对应的路径长度
            l_answer = position2route(current_position);
            points12 = find_circle_intersections(current_position, d, o1, x_radius);
            points34 = find_circle_intersections(current_position, d, o2, y_radius);
            points1 = points12(1, :);
```

```
        points2 = points12(2, :);
        points3 = points34(1, :);
        points4 = points34(2, :);
        l1 = position2route(points1);
        l2 = position2route(points2);
        l3 = position2route(points3);
        l4 = position2route(points4);
        l_list = [l1, l2, l3, l4];
        for j = 1:4
            if l_list(j) == -1 || l_list(j) >= l_answer
                l_list(j) = 0;
            end
        end
        if max(l_list) > 0
            x_cartesian = route2position(max(l_list));
            % 存储当前把手的直角坐标到 result
            result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
        else
            % 查找失败
            x_temp = current_position(1);
            y_temp = current_position(2);
            theta_temp = fzero(@(x) (a*x*cos(x)-x_temp)^2+(a*x*sin(x)-y_temp)^2-d^2,theta_in);
            r_temp = a * theta_temp;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
        end
    elseif r_temp > R && flag
        % 计算 beta_i
        beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
            (r_temp+a*beta) * cos(beta), [0, pi/2]);
        theta_ = theta_real - beta;
        r_ = (theta_ + alpha) * a;
        if r_ >= R
            theta_temp = theta_;
            r_temp = r_;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
        else
            % 计算当前位置对应的路径长度
            l_answer = L_best;
            points12 = find_circle_intersections(current_position, d, o1, x_radius);
            points34 = find_circle_intersections(current_position, d, o2, y_radius);
            points1 = points12(1, :);
            points2 = points12(2, :);
```

```
            points3 = points34(1, :);
            points4 = points34(2, :);
            l1 = position2route(points1);
            l2 = position2route(points2);
            l3 = position2route(points3);
            l4 = position2route(points4);
            l_list = [l1, l2, l3, l4];
            for j = 1:4
                if l_list(j) == -1
                    l_list(j) = 0;
                end
            end
            if max(l_list) > 0
                x_cartesian = route2position(max(l_list));
                % 存储当前把手的直角坐标到 result
                result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
                result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
            end
        end
    elseif r_temp > R
        % 计算 beta_i
        beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
            (r_temp+a*beta) * cos(beta), [0, pi/2]);
        theta_temp = theta_temp + beta;
        [x_temp, y_temp] = pol2cart(theta_temp, a * theta_temp);
        % 存储当前把手的直角坐标到 result
        result_xy_after_delta(2*i-1, t) = x_temp; % 第i节龙身x
        result_xy_after_delta(2*i, t) = y_temp; % 第i节龙身y
    end
    current_position(1) = result_xy_after_delta(2*i-1, t);
    current_position(2) = result_xy_after_delta(2*i, t);
    r_temp = sqrt(dot(current_position, current_position));
  end
end
```

q4_after_delta.m

```
% 参数初始化
k = 2; % 调头圆弧的的半径比
alpha = 2 * pi / 3; % 盘出点的圆心角
R = 4.5; % 掉头区域半径
x_radius = 2.805639223409097; % 调头曲线大圆半径
y_radius = 1.402819611704549; % 调头曲线小圆半径
angle1 = 2.450545798771806; % 半径较大的圆弧的圆心角
angle2 = 3.497743349968404; % 半径较小的圆弧的圆心角
L_best = 11.782050379839399; % 掉头曲线的长度
```

```
% 盘入螺线方程:r=a*theta
% 盘出螺线方程:r=a*(theta+alpha)

d1 = 2.86; % 第一个和第二个把手的距离
d2 = 1.65; % 第i个把手与第i+1个把手的距离
l = 1.7; % 螺距
a = l / (2 * pi); % 螺线的参数a
theta_0 = 16 * pi;
theta_in = R / a; % 螺线盘入点的极角
theta_out = theta_in - alpha; % 螺线盘出点的极角
x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标
o1 = x_in * (R - x_radius) / R; % 大圆圆心
o2 = x_out * (R - y_radius) / R; % 小圆圆心

% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));
l_theta_0 = l_theta(theta_in);
t_lst = 1:100;
% 初始化龙头把手位置
result_xy_after_delta = zeros(448,100);


delta_time = 0.05;
% 初始化正时间内龙头把手位置
l_lst_after = 0.95:1:99.955;
for t = 1:100
    t_ = l_lst_after(t);
    if t_ < L_best
        position = route2position(t_);
        result_xy_after_delta(1, t) = position(1);
        result_xy_after_delta(2, t) = position(2);
    else
        l_temp = t_ - L_best + l_theta_0;
        theta_t = fzero(@(theta) l_theta(theta) - l_temp, theta_0);
        theta_temp = theta_t - alpha;
        r_temp = theta_t * a;
        x_temp = r_temp * cos(theta_temp);
        y_temp = r_temp * sin(theta_temp);
        result_xy_after_delta(1, t) = x_temp;
        result_xy_after_delta(2, t) = y_temp;
    end
end
```

```
tolerance = 0.1;
% 储存正时刻位置
for t = 1:100
    current_position = result_xy_after_delta(1:2, t)';
    [theta_temp, r_temp] = cart2pol(current_position(1), current_position(2));
    flag = 1;
    for i = 2:224
        d = (i == 2) * d1 + (i > 2) * d2; % 判断使用d1还是d2

        % 计算是否在盘出螺线上，若在盘出螺线上，计算极角
        theta_real = r_temp / a - alpha;
        delta_theta = theta_real - theta_temp - fix((theta_real - theta_temp) / (2 * pi)) * 2 *
            pi;


        % 若出现一次不在盘出螺线上，则后续点不会在盘出螺线上，不用考虑盘出螺线的检验
        if abs(delta_theta) > tolerance
            flag = 0;
        end

        if r_temp <= R
            % 计算当前位置对应的路径长度
            l_answer = position2route(current_position);
            points12 = find_circle_intersections(current_position, d, o1, x_radius);
            points34 = find_circle_intersections(current_position, d, o2, y_radius);
            points1 = points12(1, :);
            points2 = points12(2, :);
            points3 = points34(1, :);
            points4 = points34(2, :);
            l1 = position2route(points1);
            l2 = position2route(points2);
            l3 = position2route(points3);
            l4 = position2route(points4);
            l_list = [l1, l2, l3, l4];
            for j = 1:4
                if l_list(j) == -1 || l_list(j) >= l_answer
                    l_list(j) = 0;
                end
            end
            if max(l_list) > 0
                x_cartesian = route2position(max(l_list));
                % 存储当前把手的直角坐标到 result
                result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
                result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
            else
                % 查找失败
                x_temp = current_position(1);
```

```
            y_temp = current_position(2);
            theta_temp = fzero(@(x) (a*x*cos(x)-x_temp)^2+(a*x*sin(x)-y_temp)^2-d^2,theta_in);
            r_temp = a * theta_temp;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
        end
    elseif r_temp > R && flag
        % 计算 beta_i
        beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
            (r_temp+a*beta) * cos(beta), [0, pi/2]);
        theta_ = theta_real - beta;
        r_ = (theta_ + alpha) * a;
        if r_ >= R
            theta_temp = theta_;
            r_temp = r_;
            x_cartesian = [r_temp * cos(theta_temp), r_temp * sin(theta_temp)];
            result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
            result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
        else
            % 计算当前位置对应的路径长度
            l_answer = L_best;
            points12 = find_circle_intersections(current_position, d, o1, x_radius);
            points34 = find_circle_intersections(current_position, d, o2, y_radius);
            points1 = points12(1, :);
            points2 = points12(2, :);
            points3 = points34(1, :);
            points4 = points34(2, :);
            l1 = position2route(points1);
            l2 = position2route(points2);
            l3 = position2route(points3);
            l4 = position2route(points4);
            l_list = [l1, l2, l3, l4];
            for j = 1:4
                if l_list(j) == -1
                    l_list(j) = 0;
                end
            end
            if max(l_list) > 0
                x_cartesian = route2position(max(l_list));
                % 存储当前把手的直角坐标到 result
                result_xy_after_delta(2*i-1, t) = x_cartesian(1); % 第i节龙身x
                result_xy_after_delta(2*i, t) = x_cartesian(2); % 第i节龙身y
            end
        end
    elseif r_temp > R
        % 计算 beta_i
```

```
            beta = fzero(@(beta) (r_temp^2 + (r_temp-a*beta)^2 - d^2) - 2 * r_temp *
                (r_temp+a*beta) * cos(beta), [0, pi/2]);
            theta_temp = theta_temp + beta;
            [x_temp, y_temp] = pol2cart(theta_temp, a * theta_temp);
            % 存储当前把手的直角坐标到 result
            result_xy_after_delta(2*i-1, t) = x_temp; % 第i节龙身x
            result_xy_after_delta(2*i, t) = y_temp; % 第i节龙身y
        end
        current_position(1) = result_xy_after_delta(2*i-1, t);
        current_position(2) = result_xy_after_delta(2*i, t);
        r_temp = sqrt(dot(current_position, current_position));
    end
end
```

q4_show.m

```
load('q4_xy_before.mat')
load('q4_xy_after.mat')
load('q4_xy_before_delta.mat')
load('q4_xy_after_delta.mat')
delta_time = 0.05;
result_xy = [result_xy_before, result_xy_after];
result_xy_delta = [result_xy_before_delta, result_xy_after_delta];
result_v_xy = (result_xy_delta - result_xy) ./ delta_time;
result_v = zeros(224, 201);

% 设置异常值的阈值（根据实际情况调整）
v_threshold1 = 1.03;
v_threshold2 = 0.97;

for i = 1:224
    for j = 1:201
        v_x = result_v_xy(2*i-1, j);
        v_y = result_v_xy(2*i, j);
        v = sqrt(v_x^2 + v_y^2); % 计算速度大小

        % 判断是否为异常值
        if v > v_threshold1 || isnan(v) || isinf(v) || v < v_threshold2
            v = 1; % 异常值处理，将v设为1
        end

        % 存储速度值
        result_v(i,j) = v;
    end
end
```

```
% 参数初始化
k = 2; % 调头圆弧的的半径比
alpha = 2 * pi / 3; % 盘出点的圆心角
R = 4.5; % 掉头区域半径
x = 2.805639223409097; % 调头曲线大圆半径
y = 1.402819611704549; % 调头曲线小圆半径
angle1 = 2.450545798771806; % 半径较小的圆弧的圆心角
angle2 = 3.497743349968404; % 半径较大的圆弧的圆心角
L_best = 11.78205037983; % 掉头曲线的长度

l = 1.7; % 螺距
a = l / (2 * pi); % 螺线的参数a
theta_0 = 16 * pi;
theta_in = R / a; % 螺线盘入点的极角
theta_out = theta_in - alpha; % 螺线盘出点的极角
x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标
o1 = x_in * (R - x) / R; % 大圆圆心
o2 = x_out * (R - y) / R; % 小圆圆心

% 初始化笛卡尔坐标图
figure;
hold on;
axis equal;
grid on;

% 生成盘入螺线的角度和半径
theta_vals = linspace(theta_in, theta_0, 1000); % 生成螺线的角度
r_vals = a * theta_vals; % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k--', 'Color', 'black'); % 绘制参考螺线

theta_vals = linspace(theta_in - alpha, theta_0, 1000); % 生成螺线的角度
% 生成盘出螺线的角度和半径
r_vals = a * (theta_vals + alpha); % 生成对应的半径
% 将螺线从极坐标转换为笛卡尔坐标
[x_spiral, y_spiral] = pol2cart(theta_vals, r_vals); % 极坐标到笛卡尔坐标的转换
plot(x_spiral, y_spiral, 'k--', 'Color', 'b'); % 绘制参考螺线

% 设置笛卡尔坐标下的掉头区域
theta_circle = linspace(0, 2*pi, 100); % 0到2*pi生成圆的角度
r_circle = R * ones(size(theta_circle)); % 圆的半径为R
% 将掉头圆从极坐标转换为笛卡尔坐标
[x_circle, y_circle] = pol2cart(theta_circle, r_circle); % 极坐标到笛卡尔坐标的转换
plot(x_circle, y_circle, 'LineWidth', 2); % 绘制掉头区域的圆
```

```
% 生成极坐标下的大圆与小圆
% 大圆的坐标
theta_vals_large = linspace(theta_in - angle1, theta_in, 1000);
theta_vals_small = linspace(theta_out - angle2, theta_out, 1000);
x_large_circle = o1(1) + x * cos(theta_vals_large); % 大圆的x坐标
y_large_circle = o1(2) + x * sin(theta_vals_large); % 大圆的y坐标
% 小圆的坐标
x_small_circle = o2(1) + y * cos(theta_vals_small); % 小圆的x坐标
y_small_circle = o2(2) + y * sin(theta_vals_small); % 小圆的y坐标
plot(x_large_circle, y_large_circle, 'r-', 'LineWidth', 1);
plot(x_small_circle, y_small_circle, 'r-', 'LineWidth', 1);

% 动画循环
for t = 1:201
    % 设置坐标轴范围
    xlim([-10, 10]); % 调整 X 轴的显示范围
    ylim([-10, 10]); % 调整 Y 轴的显示范围
    % 清除之前绘制的图像
    cla;

    % 重新绘制背景圆与螺线
    plot(x_spiral, y_spiral, 'k--', 'Color', 'black');
    plot(x_circle, y_circle, 'LineWidth', 2);
    plot(x_large_circle, y_large_circle, 'r-', 'LineWidth', 1);
    plot(x_small_circle, y_small_circle, 'r-', 'LineWidth', 1);

    % 获取每个时间步 t 下的点的坐标
    x_coords = result_xy(1:2:end, t); % 获取所有点的横坐标
    y_coords = result_xy(2:2:end, t); % 获取所有点的纵坐标

    % 跳过 NaN 和 (0, 0) 坐标点
    valid_indices = ~isnan(x_coords) & ~isnan(y_coords) & ~(x_coords == 0 & y_coords == 0);
    x_coords = x_coords(valid_indices);
    y_coords = y_coords(valid_indices);

    % 绘制第一个点为红色点
    plot(x_coords(1), y_coords(1), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');

    % 绘制其余点为蓝色点
    plot(x_coords(2:end), y_coords(2:end), 'bo', 'MarkerSize', 5, 'MarkerFaceColor', 'b');

    % 用绿色线连接相邻的点
    for i = 2:length(x_coords)
        line([x_coords(i-1) x_coords(i)], [y_coords(i-1) y_coords(i)], 'Color', 'g',
            'LineWidth', 1.5);
    end
```

```
    % 显示当前时间
    time_text = ['Time: ', num2str(t-101), ' s'];
    text(-9, 9, time_text, 'FontSize', 12, 'Color', 'k', 'FontWeight', 'bold'); % 显示在左上角

    % 暂停一段时间以创建动画效果
    pause(0.1); % 可调整暂停时间以控制动画速度
end
```

q5.m

```
load('q4_v.mat');
k_max = 1;
for i = 1:201
    v = result_v(:, i);
    v0 = v(1, 1);
    v_max = max(v);
    k = v_max / v0;
    if k > k_max
        k_max = k;
    end
end

V_upper = 2; % 各节点全时刻最大允许速度 (m/s)
V_final = V_upper / k_max; % 龙头最大允许速度 (m/s)
k_max
V_final
```

generating_position.m

```
function result_xy = generating_position(start_t, end_t, h, L)
    % 初始化参数
    d1 = 2.86; % 第一个和第二个把手的距离
    d2 = 1.65; % 第i个把手与第i+1个把手的距离
    a = L / (2 * pi); % 螺线参数
    v = 1; % 速度
    theta_0 = 32 * pi; % 初始角度

    if start_t == end_t
        t_lst = [start_t];
        t_max = 1;
    else
        % 生成时间序列，步长为 h
        t_lst = start_t:h:end_t;
        t_max = length(t_lst); % 时间步数
    end
```

```
% 初始化螺线长度函数
l_theta = @(theta) a / 2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));

% 初始螺线长度
l_theta_0 = l_theta(theta_0);

% 初始化位置矩阵，224个把手的 x 和 y 坐标
result_xy = zeros(448, t_max); % 每一列存储某时间步所有把手的位置

% 循环遍历每一个时间步
for t_idx = 1:t_max
    current_time = t_lst(t_idx); % 获取当前时间

    % 计算 t 时刻螺线长度
    l_t = l_theta_0 - v * current_time;

    % 防止螺线长度出现负数
    if l_t < 0
        l_t = 0;
    end

    % 利用 fzero 解 t 时刻的 theta 值
    theta_t = fzero(@(theta) l_theta(theta) - l_t, theta_0);

    % 计算龙头的极坐标
    x_t = [a * theta_t, theta_t]; % 龙头的极坐标
    x_cartesian = [x_t(1) * cos(x_t(2)), x_t(1) * sin(x_t(2))]; % 转换为直角坐标

    % 记录龙头直角坐标到 result_xy
    result_xy(1, t_idx) = x_cartesian(1); % 龙头 x
    result_xy(2, t_idx) = x_cartesian(2); % 龙头 y

    % 计算每个把手的位置并存储到 result_xy
    x_i = x_t; % 当前把手位置
    for i = 2:224
        % 判断使用 d1 还是 d2
        d = (i == 2) * d1 + (i > 2) * d2;

        % 使用 fzero 求解 beta_i
        beta_i = fzero(@(beta) (x_i(1)^2 + (x_i(1) + a * beta)^2 - d^2) - 2 * x_i(1) *
            (x_i(1) + a * beta) * cos(beta), 0.1);

        % 更新第 i 个把手的极坐标
        x_i = [x_i(1) + a * beta_i, x_i(2) + beta_i];

        % 将当前把手的位置转换为直角坐标
        x_cartesian = [x_i(1) * cos(x_i(2)), x_i(1) * sin(x_i(2))];
```

```
            % 记录当前把手直角坐标到 result_xy
            result_xy(2*i-1, t_idx) = x_cartesian(1); % 第 i 把手的 x 坐标
            result_xy(2*i, t_idx) = x_cartesian(2); % 第 i 把手的 y 坐标
        end
    end
end
```

if_coordinates.m

```
function flag = if_coordinates(x1, y1, x2, y2, x3, y3, x4, y4)
    % 输入四个点的坐标，判定两块板是否会相撞
    % x1, y1, x2, y2 表示内侧第一块板的两端坐标
    % x3, y3, x4, y4 表示外侧第二块板的两端坐标

    % 计算向量
    vector1 = [x1 - x2, y1 - y2]; % 第一块板的向量
    d = sqrt(dot(vector1, vector1)); % 向量长度
    D1 = [x1, y1] + vector1 * 0.275 / d; % D1 为第一块板的一端
    D2 = [x2, y2] - vector1 * 0.275 / d; % D2 为第一块板的另一端
    vector2 = [y2 - y1, x1 - x2]; % 法向量
    vector2 = vector2 * 0.15 / d; % 调整法向量的大小

    % 初始化两个碰撞标志
    flag1 = 0;
    flag2 = 0;

    % 计算D1和D2距离
    distance_D1 = dot(D1, D1);
    distance_D2 = dot(D2, D2);

    % 计算C1
    if dot(D1 + vector2, D1 + vector2) > distance_D1
        C1 = D1 + vector2;
    else
        C1 = D1 - vector2;
    end

    % 计算C2
    if dot(D2 + vector2, D2 + vector2) > distance_D2
        C2 = D2 + vector2;
    else
        C2 = D2 - vector2;
    end

    % 第二块板的坐标
    B1 = [x3, y3];
```

```
    B2 = [x4, y4];
    d_B1B2 = 2.2 - 0.275 * 2; % 第二块板的端点距离

    % 计算C1能否碰撞
    vector1_cross = (B1(1) - C1(1)) * (B2(2) - C1(2)) - (B1(2) - C1(2)) * (B2(1) - C1(1)); %
        计算外积
    distance1 = abs(vector1_cross) / d_B1B2; % 距离计算
    if distance1 < 0.15
        B1C1 = C1 - B1;
        B1B2 = B2 - B1;
        lamda = dot(B1C1, B1B2) / dot(B1B2, B1B2);
        B1H = lamda * B1B2;
        B2H = -B1B2 + B1H;
        dot1 = dot(B1H, B2H);
        if dot1 < 0.275 * (0.275 + d_B1B2)
            flag1 = 1;
        end
    end

    % 计算C2能否碰撞
    vector2_cross = (B1(1) - C2(1)) * (B2(2) - C2(2)) - (B1(2) - C2(2)) * (B2(1) - C2(1)); %
        计算外积
    distance2 = abs(vector2_cross) / d_B1B2; % 距离计算
    if distance2 < 0.15
        B1C2 = C2 - B1;
        B1B2 = B2 - B1;
        lamda = dot(B1C2, B1B2) / dot(B1B2, B1B2);
        B1H = lamda * B1B2;
        B2H = -B1B2 + B1H;
        dot1 = dot(B1H, B2H);
        if dot1 < 0.275 * (0.275 + d_B1B2)
            flag2 = 1;
        end
    end

    flag = flag1 || flag2;
end
```

calc_collision_r.m

```
function r = calc_collision_r(L)
    % 输入螺距L，返回停止时龙头到原点的距离
    d1 = 2.86; % 第一个和第二个把手的距离
    d2 = 1.65; % 第i个把手与第i+1个把手的距离
    a = L / (2 * pi); % 螺线参数
    v = 1; % 速度
    theta_0 = 32 * pi; % 初始角度
```

```
% 计算螺线长度函数
l_theta = @(theta) a/2 * (theta .* sqrt(1 + theta.^2) + log(theta + sqrt(1 + theta.^2)));
% 初始螺线长度
l_theta_0 = l_theta(theta_0);

% 搜索起点为0，时间步长为10s
t_start = 0;
delta = 10;
% 搜索终点为螺线长度/龙头速度
t_end = l_theta_0;
result_xy_10 = generating_position(t_start, t_end, delta, L);
t_lst = t_start:delta:t_end;
n = length(t_lst);
t_start = t_lst(1);
t_end = t_lst(n);
flag_lst = zeros(1, n);

% 循环遍历每个时间步
for k = 1:n
    flag = 0; % 初始化当前时间步的标志位
% 遍历前32个把手
for i = 1:32
    lst = result_xy_10(:, k)';

    % 获取第i个把手的坐标 (x1, y1), (x2, y2)
    x1 = lst(2*i-1);
    y1 = lst(2*i);
    if i < 224
        x2 = lst(2*i+1);
        y2 = lst(2*i+2);
    end

    % 遍历后续把手，判断是否可能发生碰撞
    for j = i+2:223
        % 获取第j个把手的坐标 (x3, y3), (x4, y4)
        x3 = lst(2*j-1);
        y3 = lst(2*j);
        if j < 224
            x4 = lst(2*j+1);
            y4 = lst(2*j+2);
        end

        % 计算四点之间的最小距离
        d1 = sqrt((x1 - x3)^2 + (y1 - y3)^2);
        d2 = sqrt((x1 - x4)^2 + (y1 - y4)^2);
        d3 = sqrt((x2 - x3)^2 + (y2 - y3)^2);
        d4 = sqrt((x2 - x4)^2 + (y2 - y4)^2);
```

```
        d_min = min([d1, d2, d3, d4]);

        % 如果最小距离小于1，检查是否碰撞
        if d_min < 1
            flag = if_coordinates(x1, y1, x2, y2, x3, y3, x4, y4);
        end

        % 如果发生碰撞，跳出循环
        if flag
            break
        end
    end

    % 如果发生碰撞，跳出外层循环
    if flag
        break
    end
end

    % 存储当前时间步的碰撞标志位
    flag_lst(k) = flag;
end

for i = 1:n
    if flag_lst(i)
        t_end = t_lst(i);
        break
    end
end
t_start = t_end - 10;

% 新的搜索范围，步长为0.1
delta_fine = 0.1; % 细致搜索的步长为0.1
result_xy_fine = generating_position(t_start, t_end, delta_fine, L);
n_fine = (t_end - t_start) / delta_fine + 1;
flag_lst_fine = zeros(1, n_fine);

% 循环遍历每个时间步，进行精细搜索
for k = 1:n_fine
    flag = 0; % 初始化当前时间步的标志位

    % 遍历前32个把手
    for i = 1:32
        lst = result_xy_fine(:, k)';

        % 获取第i个把手的坐标 (x1, y1), (x2, y2)
        x1 = lst(2*i-1);
```

```
        y1 = lst(2*i);
        if i < 224
            x2 = lst(2*i+1);
            y2 = lst(2*i+2);
        end

        % 遍历后续把手，判断是否可能发生碰撞
        for j = i+2:223
            % 获取第j个把手的坐标 (x3, y3), (x4, y4)
            x3 = lst(2*j-1);
            y3 = lst(2*j);
            if j < 224
                x4 = lst(2*j+1);
                y4 = lst(2*j+2);
            end

            % 计算四点之间的最小距离
            d1 = sqrt((x1 - x3)^2 + (y1 - y3)^2);
            d2 = sqrt((x1 - x4)^2 + (y1 - y4)^2);
            d3 = sqrt((x2 - x3)^2 + (y2 - y3)^2);
            d4 = sqrt((x2 - x4)^2 + (y2 - y4)^2);
            d_min = min([d1, d2, d3, d4]);

            % 如果最小距离小于1，检查是否碰撞
            if d_min < 1
                flag = if_coordinates(x1, y1, x2, y2, x3, y3, x4, y4);
            end

            % 如果发生碰撞，跳出循环
            if flag
                break
            end
        end

        % 如果发生碰撞，跳出外层循环
        if flag
            break
        end
    end

    % 存储当前时间步的碰撞标志位
    flag_lst_fine(k) = flag;
end

% 更新最终时间
time_lst_fine = linspace(t_start, t_end, n_fine);
for i = 1:n_fine
```

```
        if flag_lst_fine(i)
            t_end_fine = time_lst_fine(i);
            break
        end
    end



    result_xy_final = generating_position(t_end_fine, t_end_fine, 0, L);
    x = result_xy_final(1, 1);
    y = result_xy_final(2, 1);
    r = sqrt(x^2 + y^2);
end
```

find_circle_intersections.m

```
function intersection_points = find_circle_intersections(center1, r1, center2, r2)
    % center1 = [x1, y1]: 第一个圆的圆心坐标
    % r1: 第一个圆的半径
    % center2 = [x2, y2]: 第二个圆的圆心坐标
    % r2: 第二个圆的半径

    % 解的输出初始化
    intersection_points = [-100, -100; -100, -100];

    % 获取圆心的坐标
    x1 = center1(1);
    y1 = center1(2);
    x2 = center2(1);
    y2 = center2(2);

    % 计算圆心之间的距离
    d = sqrt((x2 - x1)^2 + (y2 - y1)^2);

    % 检查是否有解的条件
    if d > (r1 + r2) || d < abs(r1 - r2)
        % 圆相离或一个圆在另一个圆内，没有交点
        return;
    elseif d == 0 && r1 == r2
        % 两个圆完全重合，无穷多个交点
        return;
    end

    % 计算交点的公式
    a = (r1^2 - r2^2 + d^2) / (2 * d);
    h = sqrt(r1^2 - a^2);
```

```
    % 圆心到交点的直线上的中间点
    x0 = x1 + a * (x2 - x1) / d;
    y0 = y1 + a * (y2 - y1) / d;

    % 交点1
    x_intersect1 = x0 + h * (y2 - y1) / d;
    y_intersect1 = y0 - h * (x2 - x1) / d;

    % 交点2
    x_intersect2 = x0 - h * (y2 - y1) / d;
    y_intersect2 = y0 + h * (x2 - x1) / d;

    % 输出解
    if d == r1 + r2 || d == abs(r1 - r2)
        % 圆相切，只有一个交点
        intersection_points = [x_intersect1, y_intersect1; -100, -100];
    else
        % 圆相交，有两个交点
        intersection_points = [x_intersect1, y_intersect1; x_intersect2, y_intersect2];
    end
end
```

position2route.m

```
function l = position2route(position)
    % 参数初始化
    R = 4.5; % 掉头区域半径
    L = 1.7; % 螺距
    a = L / (2 * pi); % 螺线的参数a
    alpha = 2 * pi / 3; % 盘出点的圆心角
    x_radius = 2.805639223409097; % 调头曲线大圆半径
    y_radius = 1.402819611704549; % 调头曲线小圆半径
    angle1 = 2.450545798771806; % 大圆弧的圆心角
    angle2 = 3.497743349968404; % 小圆弧的圆心角
    L_best = 11.782050379839399; % 最优曲线长度

    % 螺线盘入点和盘出点的极角
    theta_in = R / a; % 螺线盘入点的极角
    theta_out = theta_in - alpha; % 螺线盘出点的极角
    x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
    x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标

    % 圆心坐标
    o1 = x_in * (R - x_radius) / R; % 大圆的圆心
    o2 = x_out * (R - y_radius) / R; % 小圆的圆心

    % 计算大圆和小圆的弧长
```

```
    l1 = x_radius * angle1;
    l2 = y_radius * angle2;

    tolerance = 0.01; % 允许的计算精度误差

    vector1 = position - o1;
    vector2 = position - o2;
    d1 = sqrt(dot(vector1, vector1));
    d2 = sqrt(dot(vector2, vector2));
    if abs(d1 - x_radius) < tolerance
        vector1 = vector1 / d1;
        [theta_temp, ~] = cart2pol(vector1(1), vector1(2));
        angle = theta_in - theta_temp - floor((theta_in - theta_temp) / (2 * pi)) * 2 * pi;
        l = x_radius * angle;
    elseif abs(d2 - y_radius) < tolerance
        vector2 = vector2 / d2;
        [theta_temp, ~] = cart2pol(vector2(1), vector2(2));
        angle = theta_out - theta_temp - floor((theta_out - theta_temp) / (2 * pi)) * 2 * pi;
        l = L_best - y_radius * angle;
    else
        l = -1;
    end
end
```

route2position.m

```
function position = route2position(l)
    % 参数初始化
    R = 4.5; % 掉头区域半径
    L = 1.7; % 螺距
    a = L / (2 * pi); % 螺线的参数a
    alpha = 2 * pi / 3; % 盘出点的圆心角
    x_radius = 2.805639223409097; % 调头曲线大圆半径
    y_radius = 1.402819611704549; % 调头曲线小圆半径
    angle1 = 2.450545798771806; % 大圆弧的圆心角
    angle2 = 3.497743349968404; % 小圆弧的圆心角
    L_best = 11.782050379839399; % 最优曲线长度

    % 螺线盘入点和盘出点的极角
    theta_in = R / a; % 螺线盘入点的极角
    theta_out = theta_in - alpha; % 螺线盘出点的极角
    x_in = [R * cos(theta_in), R * sin(theta_in)]; % 螺线盘入点的坐标
    x_out = [R * cos(theta_out), R * sin(theta_out)]; % 螺线盘出点的坐标

    % 圆心坐标
    o1 = x_in * (R - x_radius) / R; % 大圆的圆心
    o2 = x_out * (R - y_radius) / R; % 小圆的圆心
```

```
    % 计算大圆和小圆的弧长
    l1 = x_radius * angle1;
    l2 = y_radius * angle2;

    % 判断路径在大圆还是小圆
    if l < l1
        % 在大圆上
        theta_l = theta_in - l / x_radius; % 计算对应的角度
        x_temp = o1(1) + x_radius * cos(theta_l);
        y_temp = o1(2) + x_radius * sin(theta_l);
        position = [x_temp, y_temp];
    else
        % 在小圆上
        remaining_length = L_best - l;
        theta_l = theta_out - remaining_length / y_radius; % 计算对应的角度

        x_temp = o2(1) + y_radius * cos(theta_l);
        y_temp = o2(2) + y_radius * sin(theta_l);
        position = [x_temp, y_temp];
    end
end
```