等距螺线下“板凳龙”的位速与碰撞模型

摘要

本文在等距螺线的背景下，通过建立一个简单的二维模型，利用物理之间的长度和速度约束等、以及包围盒碰撞算法，研究“板凳龙”（又名“盘龙”）的各个把手的速度、板凳的位形以及在考虑板凳的宽度后盘入、盘出时可能会发生的碰撞和完成对调头曲线的优化和速度的限制。

针对问题一，本文建立了一个简单的二维模型，建立**极坐标系**，将板凳的运动退化来研究把手的运动。根据物理上对把手间板凳长度和相邻把手之间速度的**约束**，通过建立代数方程用**递推迭代法**进行位置求解，并对得出的可能解进行一定的**限制来筛选出合理的位置解**，再根据上述约束得到速度解，并将结果保存在文件 result1.xlsx 中。

针对问题二，仍然采用问题一所建立的二维平面模型，将板凳的宽度和长度考虑进实际问题，将单独的一个板凳视为一个**二维平面上的矩形**，利用 **OBB 包围盒的算法**（利用各边的投影进行逻辑运算）来判断除相邻板凳外各个板凳和其他板凳碰撞情况，**对时间进行精细分化**，一步步来求出较为精确的数值解，最终求得的碰撞临界在 412.473838s 时，此时板凳龙的各把手速度位置关系被保存在 result2.xlsx 文件中。

针对问题三，仍然采用问题一所建立的二维平面模型，**沿用**问题二的碰撞算法，**建立轨迹螺距优化模型**。考察在龙头到达调头空间之前的时间里板凳发生碰撞的情况，以此为约束条件来求解等距螺线轨迹螺距的最小值。最终最小螺距的值约为 0.416466m。

针对问题四，根据调头时的位置，利用已知**圆弧半径比条件和几何约束**，确定调头曲线，以调头曲线最短为目标，以盘入、盘出时板凳间不发生碰撞为约束**建立优化模型**，利用**粒子群算法**得出调头的最佳位置，从而利用问题一中的物理长度和速度约束计算出调头曲线确定后的各个时刻的把手位置和速度，。

针对问题五，已知问题四的结果，以及把手位形确定后其速度之间的**线性关系**，通过找出全部时刻下把手速度不超过 2m/s 的最大值，根据线性关系对龙头速度进行计算，从而得到满足符合约束条件的值。

关键词：**二维平面 极坐标 递推迭代法 OBB 包围盒算法 优化模型**

一、问题重述

1.1 问题背景

“板凳龙”又称“盘龙”，是浙闽地区的传统地方民俗文化活动。人们将板凳首尾相连形成板凳龙。盘龙时，龙头在前，龙身和龙尾相随，整体呈圆盘状。一般来说，在舞龙队能够自如地盘入和盘出的前提下，盘龙所需要的面积越小、行进速度越快，则观赏性越好。如何通过数学建模求各板凳的位置和速度，避免各板凳间的碰撞和舞龙队的调头问题成为了我们亟待解决的问题。

1.2 问题重述

问题一：舞龙队沿螺距 55cm 的等距螺线顺时针盘入，各把手中心均位于螺线上，龙头前把手速度保持 1m/s。计算从初始时刻到 300s 为止，每秒整个舞龙队的位置和速度，并按要求将一部分数据写在本文中。

问题二：计算舞龙队盘入的终止时刻，使得板凳之间不发生碰撞，并给出此时舞龙队的位置和速度，同时按要求将一部分数据写在本文中。

问题三：舞龙队需在以螺线中心为圆心、直径为 9m 的圆形区域的调头空间内逆时针盘出。计算最小螺距，使得龙头前把手能够盘入到调头空间的边界。

问题四：盘入螺线的螺距为 1.7m，盘出螺线与盘入螺线关于中心对称，舞龙队的调头路径是由两段圆弧相切连接而成的 S 形曲线，前一段圆弧的半径是后一段的 2 倍，它与盘入、盘出螺线均相切。画出圆弧曲线，使得调头曲线最短。龙头前把手速度保持 1m/s。以 2 调头开始时间为零时刻，计算出从-100s 开始到 100s 为止，每秒舞龙队的位置和速度。同时按要求将一部分数据写在本文中。

问题五：舞龙队按问题四设定的路径行进，龙头行进速度保持不变，计算出龙头的最大行进速度，使得舞龙队各把手的速度均不超过 2m/s。

二、问题分析

2.1 问题一的分析

本文建立了一个二维平面模型，建立极坐标系，将板凳的运动退化来研究把手的运动。首先，通过螺距求出螺线方程，然后，运用板凳长度不变的假设和递推迭代法逐一求出各把手的位置，且已知龙头速度始终保持 1m/s，速度沿板凳方向的分量相等，再次利用递推迭代法求出任意时刻各个把手的速度，并对得出的可能解进行一定的限制来筛选出合理的位置解，并将结果保存在文件 result1.xlsx 中，并且按要求完成表格。

2.2 问题二的分析

仍采用问题一所建立的二维平面模型，将板凳的宽度和长度考虑进实际问题，将板凳视为二维平面上的矩形，本题研究两个板凳之间是否发生碰撞，从

几何上或者从计算机图形学上来判断给定位形的两个矩阵是否发生重叠。利用OBB 包围盒算法来判断除相邻板凳外各个板凳和其他板凳碰撞情况。然后再利用问题一的递推迭代法和约束条件计算出舞龙队各个把手的位置和速度。

2.3 问题三的分析

仍采用问题一所建立的二维平面模型，沿用问题二的碰撞算法，建立轨迹螺距优化模型。已知调头空间的大小，且在调头空间之前不能发生碰撞，以此为约束条件来求解等距螺线螺距的最小值。

2.4 问题四的分析

按照圆弧相切，半径倍数关系等几何条件粗略确定调头曲线，以盘入、盘出时板凳间不发生碰撞为约束，以调头曲线最短为目标，得出调头的最佳位置，从而利用问题一的板凳长度和速度约束计算出确定后的调头曲线的各个时刻的把手位置和速度。

2.5 问题五的分析

已知问题四的结果，以及把手位形确定后其速度之间的线性关系，通过找出全部时刻下把手速度不超过 2m/s 的最大值，根据线性关系对龙头速度进行计算，从而得到满足约束条件的结果。

三、模型假设

1、每一个演员都按部就班的使把手在等距螺线上行进，不会产生任何的偏移，这也同时限制了演员在表演时动态调整的可能性，是一个静态模型。

2、忽略各个板凳的厚度和演员身高产生的差异，假定板凳的运动始终发生在一个二维平面上，且板面和该平面始终平行，在此基础上来研究运动和碰撞的问题。

3、假定龙头把手始终按匀速率行进且不会改变，各把手位置可以灵活的转动。

4、板凳不会产生长度上的形变，这同时也符合实际。

5、假定舞龙队初始时刻按照螺线轨迹排列。

四、符号说明及名词定义

符号	说明	符号	说明
x_i	第 i 个把手的横坐标	r_i	第 i 个把手的径长
y_i	第 i 个把手的纵坐标	θ_i	第 i 个把手的极角
$\overrightarrow{v_i}$	第 i 个把手的速度	$\overrightarrow{l_i}$	第 i 张板凳前后把手的长度矢量
$\overrightarrow{k_{i1}}$	第 i 个矩形沿长度方向的方向向量	$\overrightarrow{k_{i2}}$	第 i 个矩形沿宽度方向的方向向量

$\overrightarrow{v_{i1}}$	第 i 个矩形沿长度方向的长度向量	$\overrightarrow{v_{i2}}$	第 i 个矩形沿宽度方向的长度向量
$\overrightarrow{d_{ij}}$	第 i 和第 j 个矩形中心的位移	l_i	第 i 个矩形的长度
w_i	第 i 个矩形的宽度	w_{ij}^{flag}	第 i 和第 j 个矩形在第 i 个矩形宽度上的投影判据
l_{ij}^{flag}	第 i 和第 j 个矩形在第 i 个矩形长度上的投影判据	α_i	第 i 个矩形长度方向向量与 x 轴夹角
$iscollide_{ij}$	第 i 和第 j 个矩形是否碰撞	R	调头空间半径
r_{head}^{turn}	掉头时龙头前把手的径矢	$r_{collide}$	碰撞时刻龙头前把手的径矢
x_{ni}	第 n 个粒子的第 i 个方向位移分量	v_{ni}	第 n 个粒子的第 i 个方向速度分量
w	粒子群算法的惯性权重	c_i	粒子群算法的学习因子
p_{id}	个体已知最优解	p_{gd}	种群已知最优解

五、模型建立与求解

5.1 问题一的模型建立与求解

5.1.1 问题一模型的建立

为了问题求解的方便性，本文建立平面二维模型，在二维平面内引入二维极坐标系，等距螺线极坐标的表示方程为

$$r = a + b \cdot \theta \tag{1}$$

若以螺线中心为极坐标原点，则方程可以将常数项舍去，写为

$$r = b \cdot \theta \tag{2}$$

其中 b = p / 2π，p 为等距螺线的螺距。

各把手在等距螺线上运动，鉴于板凳长度不变的假设，各个时刻板凳各把手的位置可以由各个时刻龙头的位置逐一递推迭代确定（由第一个把手可知第二个，再由第二个知第三个，以此类推），同时龙头的速度已知，且不随时间改变，根据与位置求法一样的递推迭代法可以得到任一时刻下个把手的速度。

本文设龙头前把手的坐标在直角坐标下的表示为（x_0，y_0），在极坐标下为（r_0，θ_0），其他把手的坐标点在直角坐标系中以此类推为（x_i，y_i），在极坐标下为（r_i，θ_i），i =1，2，3，…，223，直角坐标与极坐标之间有如下变换关系

$$\begin{cases} x = r \cdot cos\theta \\ y = r \cdot sin\theta \end{cases} \qquad \begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \sqrt{x^2 + y^2}/b \end{cases}$$

设各个板凳两把手之间的长度分别记作 L_i，其中 L_1 = 3.41-0.275*2=2.86m，L_i=2.20-2*0.275=1.65m，i=2，3，…，223 对板凳的两个把手之间有长度约束

$$(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 = {L_{i+1}}^2 \tag{3}$$

另外还有速度约束，如图 1 所示，由于把手始终在等距螺线上运动，所以每个把手的速度方向确定，沿着等距螺线上一点的切线，又由于假设板凳长不变，两把手的速度沿把手沿线的分量应保持不变，设前一把手速度向量 $\overrightarrow{v_i}$ ，后一把手的速度向量为$\overrightarrow{v_{i+1}}$ ，第 i 个把手到第 i+1 个把手的位置向量为$\overrightarrow{l_{i+1}}$ ，有速度约束应有

$$\overrightarrow{v_i} \cdot \overrightarrow{l_{i+1}} = \overrightarrow{v_{i+1}} \cdot \overrightarrow{l_{i+1}} \tag{4}$$

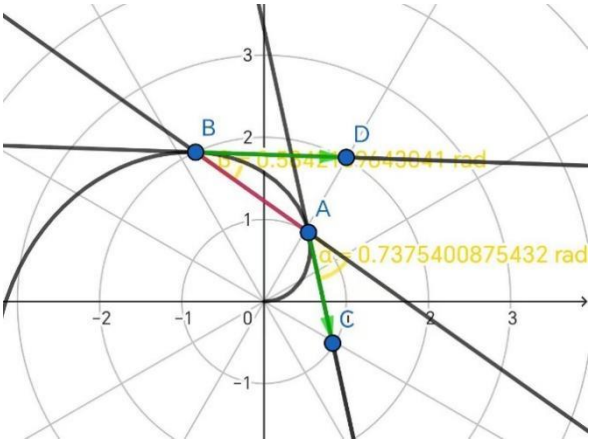


图 1 速度约束关系图

5.1.2 问题一模型的求解

在求解的过程中，为了方便统一在极坐标下进行表示，将长度约束的勾股定理转化为极坐标下的余弦定理

$$r_{i+1}^2 + r_i^2 - 2r_{i+1}r_i\cos(\theta_{i+1} - \theta_i) = L_{i+1}^2$$

由于龙头是 v_0 匀速运动，故其所走的路程

$$S = v_0 \cdot t$$

同时龙头所走的路程因为轨迹确定，是一个由初末位置确定的函数，即

$$S = \int_0^t dS = \int_\theta^{\theta_0} \sqrt{dr^2 + (rd\theta)^2} = \int_\theta^{\theta_0} b\sqrt{1+\theta^2}d\theta$$

由上面两式相等得到 t 与龙头位置 θ 的关系，即

$$v_0 \cdot t = \int_\theta^{\theta_0} b\sqrt{1+\theta^2}d\theta$$

从而在给定时刻 t 时可以反解出龙头的位置参量 θ。

按照 5.1.1 中的思路，利用 Python 中的 scipy 库，调用 fsolve 函数进行各时刻各把手位置的求解，在求解的过程中选择合适的解，合理解的约束暂定为

$$0 < \theta_{i+1} - \theta_i < \frac{\pi}{2}$$

利用递推迭代法可以求出各时刻把手位置和速度。结果如表 1 和表 2 所示

表 1 位置结果

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.8e+00	5.799209e+00	-4.084887e+00	-2.963609e+00	2.594494e+00	4.420274e+00
龙头 y (m)	0	-5.771092e+00	-6.304479e+00	6.094780e+00	-5.356743e+00	2.320429e+00
第 1 节龙身 x (m)	8.363824e+00	7.456758e+00	-1.445473e+00	-5.237118e+00	4.821221e+00	2.459489e+00
第 1 节龙身 y (m)	2.826544e+00	-3.440399e+00	-7.405883e+00	4.359627e+00	-3.561949e+00	4.402476e+00
第 51 节龙身 x (m)	-9.518732e+00	-8.686317e+00	-5.54315e+00	2.890455e+00	5.980011e+00	-6.301346e+00
第 51 节龙身 y (m)	1.341137e+00	2.540108e+00	6.377946e+00	7.249289e+00	-3.827758e+00	4.65829e-01
第 101 节龙身 x(m)	2.913983e+00	5.687116e+00	5.361939e+00	1.898794e+00	-4.917371e+00	-6.237722e+00
第 101 节龙身 y (m)	-9.918311e+00	-8.001384e+00	-7.557638e+00	-8.471614e+00	-6.379874e+00	3.936008e+00
第 151 节龙身 x (m)	[10.86172628]	[6.68231146]	[2.38875716]	[1.00515398]	[2.96537811]	[7.04074023]
第 151 节龙身 y (m)	[1.82875346]	[8.13454395]	[9.72741133]	[9.42475098]	[8.39972055]	[4.39301313]
第 201 节龙身 x (m)	[4.55510223]	[-6.61966363]	[-10.62721053]	[-9.28771984]	[-7.45715121]	[-7.45866207]
第 201 节龙身 y (m)	[10.72511774]	[9.02557032]	[1.35984719]	[-4.2466729]	[-6.18072612]	[-5.26338411]
龙尾（后）x (m)	[-5.30544404]	[7.36455722]	[10.97434827]	[7.38389563]	[3.24105108]	[1.78503273]
龙尾（后）y (m)	[-10.67658431]	[-8.79799165]	[0.84347322]	[7.49237046]	[9.46933643]	[9.30116402]

利用板凳把手位置，递推求解速度大小

表 2 速度结果

	0s	60s	120s	180s	240s	300s
龙头 (m/s)	1	1	1	1	1	1

第 1 节龙身 (m/s)	9.999710e-01	9.999611e-01	9.999451e-01	9.999166e-01	9.998586e-01	9.997095e-01
第 51 节龙身 (m/s)	9.997419e-01	9.996620e-01	9.995382e-01	9.993306e-01	9.989411e-01	9.980647e-01
第 101 节龙身（m/s）	9.99575e-01	9.994531e-01	9.99269e-01	9.989707e-01	9.984353e-01	9.973018e-01
第 151 节龙身（m/s）	9.99448e-01	9.992988e-01	9.990777e-01	9.987272e-01	9.981148e-01	9.968613e-01
第 201 节龙身（m/s）	9.993481e-01	9.991803e-01	9.989347e-01	9.985514e-01	9.978935e-01	9.965743e-01
龙尾（后）（m/s）	9.993106e-01	9.991364e-01	9.988827e-01	9.984887e-01	9.978165e-01	9.964775e-01

5.2 问题二的模型建立与求解

5.2.1 问题二模型的建立

仍然采用问题一建立的模型，但是将板凳的宽度和总长纳入考虑的范围，将每一个板凳表示为一个平面上的矩形，即考虑二维几何图形的分布情况。而且要考察两个板凳之间是否发生碰撞，从几何上或从计算机图形学上来判断给定位形的两个矩形是否发生重叠。由于难以推断真实情况下第一次碰撞发生的位置，本文采用计算机图形学上的二维 OBB 包围盒碰撞检测算法来判断某一时刻下任意两个板凳矩形的重叠情况。先对时间进行粗略划分，在碰撞的临界时间点附近对时间再进行细分来提高精度。

OBB 碰撞检测算法：如图 2，设两个被检测矩形的半长、半宽矢量分别为$\overrightarrow{v_1}$、$\overrightarrow{v_2}$、$\overrightarrow{v_3}$、$\overrightarrow{v_4}$，两矩形中心之间的矢量为$\vec{d}$，其核心思想是比较两个矩形的半对角线矢量的和与中心向量在边上四个方向上投影值的大小。图中所表示的 α 和 β 分别是两个矩形长度方向与直角坐标 x 轴的夹角。$\overrightarrow{k_1}$、$\overrightarrow{k_2}$、$\overrightarrow{k_3}$、$\overrightarrow{k_4}$分别是两个矩形四条边的方向向量，有如下的关系式：

$$\begin{cases}\overrightarrow{k_1} = （\cos\alpha , \sin\alpha） \\ \overrightarrow{k_2} = (sin\alpha , -cos\alpha)\end{cases} \quad \begin{cases}\overrightarrow{v_1} = \overrightarrow{k_1} \cdot l_1/2 \\ \overrightarrow{v_2} = \overrightarrow{k_2} \cdot w_1/2\end{cases}$$

$$\begin{cases}\overrightarrow{k_3} = （\cos\beta , \sin\beta） \\ \overrightarrow{k_4} = (sin\beta , -cos\beta)\end{cases} \quad \begin{cases}\overrightarrow{v_3} = \overrightarrow{k_3} \cdot l_2/2 \\ \overrightarrow{v_4} = \overrightarrow{k_4} \cdot w_2/2\end{cases}$$

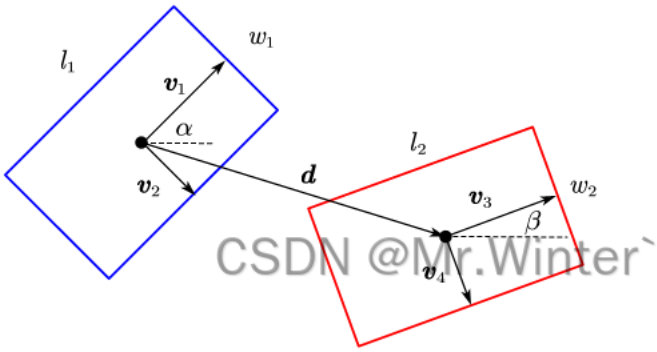


图 2 OBB 说明图（1）

以图 3 中在$\overrightarrow{v_1}$方向投影为例，r_1^-和 r_2^-分别是第一个矩形和第二个矩形的半对角线矢量在$\overrightarrow{v_1}$方向上的投影大小，d 是中心位置矢量$\vec{d}$在该方向上的投影大小。

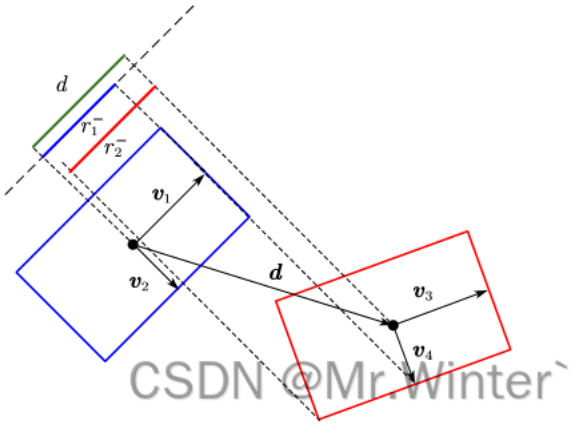


图 3 OBB 说明图（2）

只要在一个投影方向上两半对角线矢量分离，则表明两个矩形没有重叠的部分即不发生碰撞；反之，若四个边投影方向上都表明重叠，则两个矩形一定发生重叠，故能表示成如下图 4 的逻辑：

Function: OBB$(rect_1, rect_2)$

Input: 待检测的矩形$rect_1$与$rect_2$

Output：$rect_1$与$rect_2$间是否存在碰撞

[l_1-axis overlap]

$l_1^{flag} = |\boldsymbol{d}^T \boldsymbol{k}_1| < l_1^- + |\boldsymbol{v}_3^T \boldsymbol{k}_1| + |\boldsymbol{v}_4^T \boldsymbol{k}_1|$

[w_1-axis overlap]

$w_1^{flag} = |\boldsymbol{d}^T \boldsymbol{k}_2| < w_1^- + |\boldsymbol{v}_3^T \boldsymbol{k}_2| + |\boldsymbol{v}_4^T \boldsymbol{k}_2|$

[l_2-axis overlap]

$l_2^{flag} = |\boldsymbol{d}^T \boldsymbol{k}_3| < l_2^- + |\boldsymbol{v}_1^T \boldsymbol{k}_3| + |\boldsymbol{v}_2^T \boldsymbol{k}_3|$

[w_2-axis overlap]

$w_2^{flag} = |\boldsymbol{d}^T \boldsymbol{k}_4| < w_2^- + |\boldsymbol{v}_1^T \boldsymbol{k}_4| + |\boldsymbol{v}_2^T \boldsymbol{k}_4|$

return l_1^{flag} and w_1^{flag} and l_2^{flag} and w_2^{flag}

图 4 OBB 伪代码

5.2.2 问题二模型的求解

在板凳龙的实际碰撞中，由于不确定碰撞刚开始发生的位置，所以采用遍历算法将任意两个板凳之间的碰撞情况均做一个判断，但这里要注意的是，相邻的两个板凳是必然重叠的，然而并不是碰撞的情况，所以要从该比较对象之后的第二个板凳来比较。下面给出数学表示：设考查的是第 i 个板凳和第 j 个板凳的碰撞情况，不失一般性，不妨有$j > i + 1$, 设$\overrightarrow{d_{ij}}$是由第 i 个矩形中心指向第 j 个矩形中心的向量，$\overrightarrow{k_{i1}}$、$\overrightarrow{k_{i2}}$、$\overrightarrow{k_{j1}}$、$\overrightarrow{k_{j2}}$分别是第 i 和第 j 个矩形长度和宽度上的方向向量，剩下所设的变量均在符号说明当中，按照伪代码中的逻辑，可以写出：

$$l_i^{flag} = \left|\overrightarrow{d_{ij}} \cdot \overrightarrow{k_{i1}}\right| < \frac{l_i}{2} + \left|\overrightarrow{v_{j1}} \cdot \overrightarrow{k_{i1}}\right| + \left|\overrightarrow{v_{j2}} \cdot \overrightarrow{k_{i1}}\right|$$

$$w_i^{flag} = \left|\overrightarrow{d_{ij}} \cdot \overrightarrow{k_{i2}}\right| < \frac{w_i}{2} + \left|\overrightarrow{v_{j1}} \cdot \overrightarrow{k_{i2}}\right| + \left|\overrightarrow{v_{j2}} \cdot \overrightarrow{k_{i2}}\right|$$

$$l_j^{flag} = \left|\overrightarrow{d_{ij}} \cdot \overrightarrow{k_{j1}}\right| < \frac{l_j}{2} + \left|\overrightarrow{v_{i1}} \cdot \overrightarrow{k_{j1}}\right| + \left|\overrightarrow{v_{i2}} \cdot \overrightarrow{k_{j1}}\right|$$

$$w_j^{flag} = \left|\overrightarrow{d_{ij}} \cdot \overrightarrow{k_{j2}}\right| < \frac{w_j}{2} + \left|\overrightarrow{v_{i1}} \cdot \overrightarrow{k_{j2}}\right| + \left|\overrightarrow{v_{i2}} \cdot \overrightarrow{k_{j2}}\right|$$

$$iscollide_{ij} = l_i^{flag}\ and\ w_i^{flag}\ and\ l_j^{flag}\ and\ w_j^{flag}$$

在某一时刻下遍历任意不相邻的两个矩形得到在这一时刻的碰撞情况，划分时间节点，找到碰与不碰的临界，在附近再仔细划分时间得到高精度解，最后得到的碰撞临界时间是 412.473838s 左右，是龙头和第 8 块板碰撞。将临界碰撞时刻个把手的位速情况保存在 result2.xlsx，按题目要求将部分结果展示如下表 3。

表 3 碰撞临界时特定把手的位置、速度关系

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头	1.209931e+00	1.942784e+00	9.915508e-01
第 1 节龙身	-1.643791e+00	1.753399e+00	9.905163e-01
第 51 节龙身	1.281201e+00	4.326588e+00	9.767794e-01
第 101 节龙身	-5.362465e-01	-5.880138e+00	9.745239e-01
第 151 节龙身	9.688399e-01	-6.957479e+00	9.735952e-01
第 201 节龙身	-7.893161e+00	-1.230763e+00	9.730884e-01
龙尾（后）	[0.95621736]	[8.32273593]	9.729316e-01

5.3 问题三的模型建立与求解

5.3.1 问题三模型的建立

经定性分析可以知道，调头空间半径确定的情况下，等距螺线轨迹的螺距越小，在调头的地点越容易发生碰撞。所以要想题目所给的调头空间内调头且不发生碰撞，螺距最小的位置应该取在刚到达题目所给调头空间的边缘就发生碰撞的情况。由于约束十分复杂，可以将问题三化为一个优化问题，并建立以下优化模型：

$$\min z = p = 2\pi b$$
$$s.t. \quad r_{collide} \leq R$$

5.3.2 问题三模型的求解

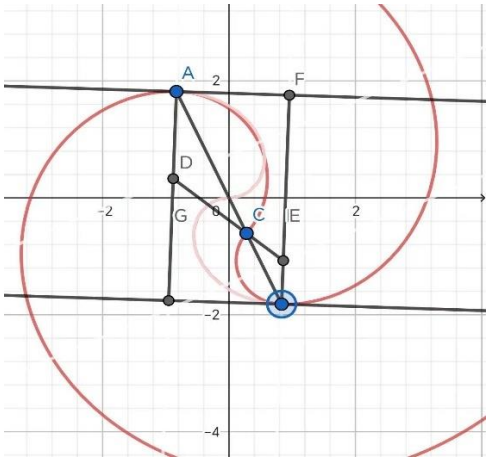
在求解问题的过程中，采用一种容易得到但较为繁杂的方法，即逐步搜索逼近的方法。方法的核心就是将要求量 b 的只按照一定的初始步长逐步计算出碰撞的时刻龙头的位置与调头空间的差值，再逐步缩小步长减小误差，最终得到一个可以接受的在要求精确度下的解。按照如上的繁杂方法计算得到最小的螺距约为 0.416466m(在碰撞半径误差为 1e-6 时)。

当然，鉴于上述方法的参数调节复杂且难度较大，还可以按建立的优化模型利用启发式算法如模拟退火算法、粒子群算法等方法来得到最小螺距的近似数值解。

5.4 问题四的模型建立与求解

5.4.1 问题四模型的建立

问题四经过分析，可以从几何上得到，当开始掉头的位置确定后，由于流入和流出曲线中心对称、两个圆弧的半径比确定，以及三个点的相切约束，可以发现整个调头的过程已经被确定，调头曲线的轨迹被确定，即调头曲线的长度也被确定。具体的几何求解如下图，有一对容易得到的相似三角形。



根据等距螺线的性质，可以利用数学分析的知识，设等距上点的切线与径矢法向的夹角 φ，调头位置（r, θ），两者有如下关系：

$$tan\phi = 1/\theta$$

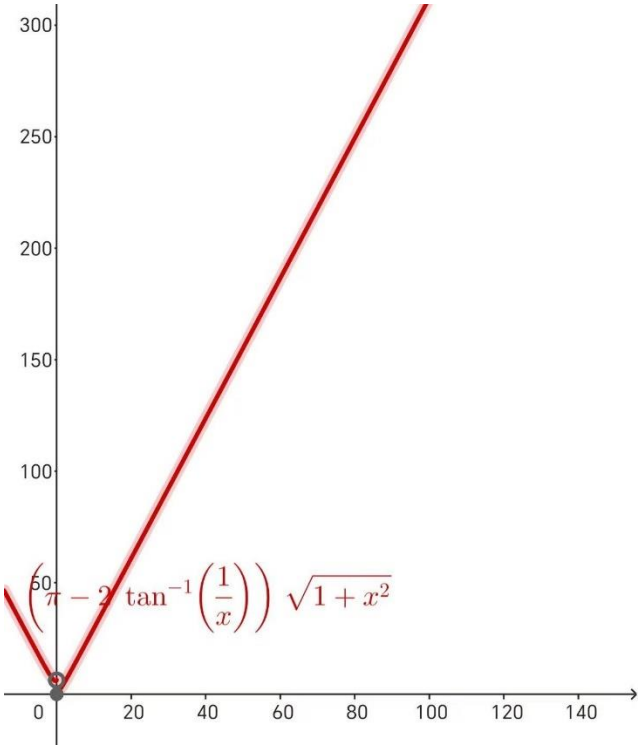
根据几何关系可以得到最终调头曲线的长度 s 在调头位置确定时，s 和 θ 的关系如下：

$$s = b(\pi \pm 2\phi)\sqrt{1+\theta^2}$$

由于要求的是最小值，必然要取负号，同时将 ϕ 和 θ 的关系带入得到

$$s = b(\pi - 2\arctan(1/\theta))\sqrt{1+\theta^2}$$

可以利用画图软件画出上述函数在自变量取正值时的函数，如下图：



可以看到调头曲线的长度是一个随调头位置 θ 单调递增的函数，可以知道调头越晚调头曲线越短。但必须还要考虑的是板凳龙不能再盘入、判处的时候不发生碰撞，定性分析知道向内盘入越深越容易发生碰撞，更无法避免盘出的板凳流与盘入的板凳流发生碰撞，所以调头要足够早才可以，所以调头曲线的长度一定有一个下限。

由以上的分析可以得到，问题四本质上也是一个优化问题，以调头曲线的长度为目标建立优化模型，可以表述为如下的形式：

$$\min z = s$$

$$s.t.\begin{cases} r_{head}^{turn} \le R \\ \prod_{i,j>i+1} iscollide_{ij} = 0\ ,in\ entire\ process \end{cases}$$

5.4.2 问题四模型的求解

由于参数和过程的复杂性，本文不再采用第三问的参数调节方法来求解本

问的优化问题，而是采用启发式算法中的粒子群算法来求解，粒子群算法是一种群体算法。

粒子群算法（也称粒子群优化算法（particle swarm optimization, PSO）），模拟鸟群随机搜索食物的行为。粒子群算法中，每个优化问题的潜在解都是搜索空间中的一只鸟，叫做“粒子”。所有的粒子都有一个由被优化的函数决定的适应值（fitness value），每个粒子还有一个速度决定它们“飞行”的方向和距离。

粒子群算法初始化为一群随机的粒子（随机解），然后根据迭代找到最优解。每一次迭代中，粒子通过跟踪两个极值来更新自己：第 1 个是粒子本身所找到的最优解，这个称为个体极值；第 2 个是整个种群目前找到的最优解，这个称为全局极值。也可以不用整个种群，而是用其中的一部分作为粒子的邻居，称为局部极值。

假设在一个 D 维搜索空间中，有 N 个粒子组成一个群落，其中第 i 个粒子表示为一个 D 维的向量：

$$X_i = (x_{i1}, x_{i2}, \dots, x_{iD})\ i = 1,2, \dots N$$

第 i 个粒子的速度表示为：

$$V_i = (v_{i1}, v_{i2}, \dots, v_{iD})\ i = 1,2, \dots N$$

还要保存每个个体的已经找到的最优解，和一个整个群落找到的最优解。

第 i 个粒子根据下面的公式更新自己的速度和位置：

$$v_{id} = w \times v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

其中,r_1 ,r_2 是[0,1]范围内的随机数。

六、模型评估

6.1 模型的优点

1. 物理约束明确，公式准确、直观，规避了数值代数的方法，尽量减小了工程上的误差

2. 采用 OBB 碰撞检测算法，使得求解问题变得更加简单

3. 第四问采用粒子群算法，对优化问题进行迭代，避免了复杂的调参过程

6.2 模型的缺点

1. 物理约束的公式过于复杂，导致向计算机输入困难，不直观且不便于使用推广

2. 在第三问中未用启发式算法，容易陷入参数调节的黑箱实验中

七、参考文献

［1］https://blog.csdn.net/FRIGIDWINTER/article/details/141790258

[2] https://www.jianshu.com/p/85960e68c9dd

附录

附录一 支撑材料的文件列表

文件一 问题一的结果文件 result1.xlsx

文件二 问题二的结果文件 result2.xlsx

文件三 问题四的结果文件 result4.xlsx

文件四 源程序代码

附录二 源程序代码

本文采用 Python 进行编程

1.问题一代码

第一问：计算速度.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd
from openpyxl import load_workbook


# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i


# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-
```

```
10):
    """
    Find the polar coordinates of a point on the spiral r = b *
theta that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be
equal to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance,
                                                full_output=True)
    # print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution
within the given tolerance.")

        # Calculate the corresponding r_tail value
```

```
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail


def angtoxy(theta, r):
    return [r * np.cos(theta), r * np.sin(theta)]


# 定义了一个类，它能计算两个旋转矩形是否碰撞

class RotatedRectangle:
    def __init__(self, center, angle, width, height):
        self.center = center  # 中心坐标 (x, y)
        self.angle = angle  # 旋转角度（弧度）
        self.width = width  # 宽度
        self.height = height  # 高度

        # 预先计算矩形的顶点
        self.update_vertices()

    def update_vertices(self):
        # 定义矩形的轴
        axis_x = [np.cos(self.angle), np.sin(self.angle)]
        axis_y = [-np.sin(self.angle), np.cos(self.angle)]

        # 计算矩形的四个顶点
        self.vertices = [
            [self.center[0] + self.width / 2 * axis_x[0] +
self.height / 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] +
self.height / 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] +
self.height / 2 * axis_y[0],
```

```
            self.center[1] - self.width / 2 * axis_x[1] +
self.height / 2 * axis_y[1]],
           [self.center[0] - self.width / 2 * axis_x[0] -
self.height / 2 * axis_y[0],
            self.center[1] - self.width / 2 * axis_x[1] -
self.height / 2 * axis_y[1]],
           [self.center[0] + self.width / 2 * axis_x[0] -
self.height / 2 * axis_y[0],
            self.center[1] + self.width / 2 * axis_x[1] -
self.height / 2 * axis_y[1]]
       ]

    def dot(self, v, w):
        return v[0] * w[0] + v[1] * w[1]

    def project(self, a, b):
        return (self.dot(a, b) / self.dot(b, b)) * b

    def on_segment(self, p, q, r):
        return q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0])
and \
            q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])

    def orientation(self, p, q, r):
        val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1]
- q[1])
        if val == 0:
            return 0  # collinear
        elif val > 0:
            return 1  # clock
        else:
            return 2  # counterclock

    def do_intersect(self, p1, q1, p2, q2):
        o1 = self.orientation(p1, q1, p2)
        o2 = self.orientation(p1, q1, q2)
        o3 = self.orientation(p2, q2, p1)
        o4 = self.orientation(p2, q2, q1)

        if o1 != o2 and o3 != o4:
            return True

        if o1 == 0 and self.on_segment(p1, p2, q1):
            return True
```

```
        if o2 == 0 and self.on_segment(p1, q2, q1):
            return True
        if o3 == 0 and self.on_segment(p2, p1, q2):
            return True
        if o4 == 0 and self.on_segment(p2, q1, q2):
            return True

        return False

    def collides_with(self, other):
        # 检查边-边碰撞
        for i in range(4):
            for j in range(4):
                if self.do_intersect(self.vertices[i],
self.vertices[(i + 1) % 4],
                                     other.vertices[j], other.vertices[(j
+ 1) % 4]):
                    return True

        return False

def velocitycal(velupper,theta,thetaplus1):   # 注意 plus1 意味着向龙尾
靠近 1
    xiplus1,yiplus1 = angtoxy(thetaplus1,rot(thetaplus1,b))
    xi,yi = angtoxy(theta,rot(theta,b))
    ki = ((xiplus1-xi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))),(yiplus1-
yi)/(np.sqrt(np.square(xiplus1-xi)+np.square(yiplus1-yi))))
    viplus1 = ((np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))/np.sqrt(np.square(np.cos(thetaplus1)
-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetapl
us1*np.cos(thetaplus1))),(np.sin(thetaplus1)+thetaplus1*np.cos(thet
aplus1))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetapl
us1*np.cos(thetaplus1))))
    vi = ((np.cos(theta)-
theta*np.sin(theta))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))),
(np.sin(theta)+theta*np.cos(theta))/np.sqrt(np.square(np.cos(theta)
-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))))

    resultup = 0
```

```
    resultdown = 0
    for i in range(2):
        resultup += ki[i]*vi[i]
        resultdown += ki[i]*viplus1[i]

    return abs(resultup*velupper/resultdown)

if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改b的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成theta的值



    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    # 定义特定点xy变量名
    headx, heady, bodyx, bodyy = 0, 0, 0, 0
    # 下面这俩是用来求速度的
    bodyx1 = []
    bodyy1 = []

    bodytheta1 = []
    bodyr1 = []
    bodytheta2 = []
    bodyr2 = []
    #  下面是保存数据的部分
    veloall = []
    savepath = 'C:\\Users\\31827\\Downloads\\result1.xlsx'
    book = load_workbook(savepath)
    writer = pd.ExcelWriter(savepath, engine='openpyxl')
    writer._book = book

    timemark = 0
    timeinterval = 0.7
    # 碰撞时刻：412.473838
    for i in range(301):  # 计算300个点
        timemark = i
        dt = np.linspace(timemark - timeinterval, timemark +
timeinterval, 2)
        tail = []
```

```
        bodytheta1 = []
        bodyr1 = []
        bodytheta2 = []
        bodyr2 = []
        ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True,xtol=1e-8)
        if message == 'The solution converged.':
            headtheta.append(ans)
            search_initial = ans  # 更新初始猜测为当前解
            upperangle = ans
            headx, heady = angtoxy(ans, rot(ans, b))
            #tail.append(ans)
            print("龙首坐标:", [headx, heady])
        else:
            print(f"Warning: fsolve did not, converge for i={i}")
        #对 dt 求速度
        for j in range(1, 225):
            if j == 1:
                L = 2.86
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                tail.append(tailtheta)
            else:
                L = 1.65
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                tail.append(tailtheta)
            upperangle = tailtheta

        print(len(tail))
        velocity = []
        velocity.append(1)
        velotem = 0
        inherit = 0
        for j in range(0,224):
            if j == 0:
                velotem = velocitycal(1,ans,tail[j])
            else:
                velotem = velocitycal(inherit,tail[j-1],tail[j])
            inherit = velotem
            velotem = np.format_float_scientific(velotem,precision=6)
            velocity.append(velotem)
        veloall.append(velocity)
    print('lens=',len(veloall))
```

```
    savedict = {f"{l} s":veloall[l] for l in range(301)}
    df = pd.DataFrame(savedict)
    print(df)
    df.to_excel(writer, sheet_name='速度', startcol=1,
header=1,index=False)
    writer._save()
    writer.close()
```

第一问：计算位置.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd

from openpyxl import load_workbook

# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i

# 定义把手和把手距离函数并求解
```

```
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-
10):
    """
    Find the polar coordinates of a point on the spiral r = b *
theta that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be
equal to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance, full_output=True)
    #print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution
within the given tolerance.")

        # Calculate the corresponding r_tail value
```

```
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        plt.show()
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail
def angtoxy(theta,r):
    return [r*np.cos(theta),r*np.sin(theta)]

#定义了一个类，它能计算两个旋转矩形是否碰撞

class RotatedRectangle:
    def __init__(self, center, angle, width, height):
        self.center = center  # 中心坐标 (x, y)
        self.angle = angle  # 旋转角度（弧度）
        self.width = width  # 宽度
        self.height = height  # 高度

        # 预先计算矩形的顶点
        self.update_vertices()

    def update_vertices(self):
        # 定义矩形的轴
        axis_x = [np.cos(self.angle), np.sin(self.angle)]
        axis_y = [-np.sin(self.angle), np.cos(self.angle)]

        # 计算矩形的四个顶点
        self.vertices = [
            [self.center[0] + self.width / 2 * axis_x[0] +
self.height / 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] +
self.height / 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] +
self.height / 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] +
self.height / 2 * axis_y[1]],
```

```
            [self.center[0] - self.width / 2 * axis_x[0] -
self.height / 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] -
self.height / 2 * axis_y[1]],
            [self.center[0] + self.width / 2 * axis_x[0] -
self.height / 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] -
self.height / 2 * axis_y[1]]
        ]

    def dot(self, v, w):
        return v[0] * w[0] + v[1] * w[1]

    def project(self, a, b):
        return (self.dot(a, b) / self.dot(b, b)) * b

    def on_segment(self, p, q, r):
        return q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0])
and \
            q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])

    def orientation(self, p, q, r):
        val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1]
- q[1])
        if val == 0:
            return 0  # collinear
        elif val > 0:
            return 1  # clock
        else:
            return 2  # counterclock

    def do_intersect(self, p1, q1, p2, q2):
        o1 = self.orientation(p1, q1, p2)
        o2 = self.orientation(p1, q1, q2)
        o3 = self.orientation(p2, q2, p1)
        o4 = self.orientation(p2, q2, q1)

        if o1 != o2 and o3 != o4:
            return True

        if o1 == 0 and self.on_segment(p1, p2, q1):
            return True
        if o2 == 0 and self.on_segment(p1, q2, q1):
            return True
```

```
        if o3 == 0 and self.on_segment(p2, p1, q2):
            return True
        if o4 == 0 and self.on_segment(p2, q1, q2):
            return True

        return False

    def collides_with(self, other):
        # 检查边-边碰撞
        for i in range(4):
            for j in range(4):
                if self.do_intersect(self.vertices[i],
self.vertices[(i + 1) % 4],
                                     other.vertices[j], other.vertices[(j
+ 1) % 4]):
                    return True

        return False

def velocitycal(velupper,theta,thetaplus1):   # 注意 plus1 意味着向龙尾
靠近 1
    xiplus1,yiplus1 = angtoxy(thetaplus1,rot(thetaplus1,b))
    xi,yi = angtoxy(theta,rot(theta,b))
    ki = np.array((xiplus1-xi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))),(yiplus1-
yi)/(np.sqrt(np.square(xiplus1-xi)+np.square(yiplus1-yi))))
    viplus1 = np.array((np.cos(thetaplus1-
thetaplus1*np.sin(thetaplus1)))/np.sqrt(np.square(np.cos(thetaplus1
)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetapl
us1*np.cos(thetaplus1))),(np.sin(thetaplus1)+thetaplus1*np.cos(thet
aplus1))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetapl
us1*np.cos(thetaplus1))))
    vi = np.array((np.cos(theta-
theta*np.sin(theta)))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))),
(np.sin(theta)+theta*np.cos(theta))/np.sqrt(np.square(np.cos(theta)
-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))))
    return np.dot(ki,vi)*velupper/np.dot(ki,viplus1)

if __name__ == '__main__':
    p = 0.55
```

```
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值

    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    #定义特定点 xy 变量名

    # 下面这俩是用来求速度的
    bodyx = []
    bodyy = []
    #saveform = []
    # 定义特定点速度


    timemark = 0
    dt = np.linspace(timemark-0.000001,timemark+0.000001,2)
    print(dt)


    # 定义特定点速度

    timemark = 0
    dt = np.linspace(timemark - 0.000001, timemark + 0.000001, 2)
    print(dt)
    savepath = 'C:\\Users\\31827\\Downloads\\result1.xlsx'
    book = load_workbook(savepath)
    writer = pd.ExcelWriter(savepath, engine='openpyxl')
    writer._book = book
    saveformall = []
    # 碰撞时刻：412.473838
    for i in range(301):  # 计算 300 个点

        # 使用 fsolve 找到满足条件的 theta

        tail = []

        saveform =[]
        ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True)
        if message == 'The solution converged.':
            headtheta.append(ans)
```

```
            search_initial = ans  # 更新初始猜测为当前解
            upperangle = ans
            headx, heady = angtoxy(ans, rot(ans, b))
            print("龙首坐标:", [headx, heady])
            saveform.append(headx)
            saveform.append(heady)
        else:
            print(f"Warning: fsolve did not, converge for i={i}")
        # ax.plot(ans,rot(ans,b),color='red')
        count = 0
        for j in range(1, 224):

            if j == 1:
                L = 2.86
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                tail.append(tailtheta)
            else:
                L = 1.65
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                tail.append(tailtheta)

            upperangle = tailtheta
            count = count + 1
            # print(count)


        for j in range(len(tail)):

saveform.append(np.format_float_scientific(angtoxy(tail[j],rot(tail
[j],b))[0],precision=6))

saveform.append(np.format_float_scientific(angtoxy(tail[j],rot(tail
[j],b))[1],precision=6))

        #saved1ct = {"横坐标 x (m)":saveformx,"纵坐标 y (m)":saveformy}
        #df = pd.DataFrame(saved1ct)
        #df.to_excel(writer, sheet_name='Sheet1', startcol=1,
header=1,index=False)
        #writer._save()
```

```
        saveformall.append(saveform)
    # 利用字典生成式生成表内容并保存
    savedict = {f"{l} s": saveformall[l] for l in range(0, 301)}
    df = pd.DataFrame(savedict)
    df.to_excel(writer, sheet_name='位置', startcol=1,
header=1,index=False)
    #print(bodyx, bodyy)
    writer._save()
    writer.close()
    # print(tail)
```

2.问题二代码

prc(1).py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd
from pandas import DataFrame
from openpyxl import load_workbook
import math
# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
```

```
    return S(theta, theta1, b, fun) - i

# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-10):
    """
    Find the polar coordinates of a point on the spiral r = b * theta
that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be equal
to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance, full_output=True)
    #print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution within
the given tolerance.")
```

```
        # Calculate the corresponding r_tail value
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        plt.show()
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail
def angtoxy(theta,r):
    return [r*np.cos(theta),r*np.sin(theta)]

#定义了一个类，它能计算两个旋转矩形是否碰撞


class OBB:
    def __init__(self, center, angle,width, height):
        self.center = np.array(center)
        self.width = width
        self.height = height
        self.angle = angle
        self.half_width = width / 2
        self.half_height = height / 2
        self.orientation = np.array([np.cos(angle), np.sin(angle)])
        self.corners = self._get_corners()

    def _get_corners(self):
        cos_angle = np.cos(self.angle)
        sin_angle = np.sin(self.angle)
        dx = self.half_width * cos_angle
        dy = self.half_width * sin_angle
        hx = self.half_height * sin_angle
        hy = self.half_height * cos_angle

        corners = [
            self.center + np.array([dx - hx, dy + hy]),
            self.center + np.array([dx + hx, dy - hy]),
```

```
            #self.center + np.array([-dx - hx, -dy + hy]),
            self.center + np.array([-dx + hx, -dy - hy]),
            self.center + np.array([-dx - hx, -dy + hy])
        ]
        return np.array(corners)

    def _get_axes(self):
        edges = [self.corners[i] - self.corners[i - 1] for i in
range(4)]
        return [np.array([-edge[1], edge[0]]) / np.linalg.norm(edge)
for edge in edges]

    def _project_onto_axis(self, axis):
        axis = axis.flatten()
        projections = [np.dot(corner.flatten(), axis) for corner in
self.corners]
        return [min(projections), max(projections)]

    def _overlap(self, projection1, projection2):
        return projection1[0] <= projection2[1] and projection2[0] <=
projection1[1]

    def is_colliding(self, other):
        axes = self._get_axes() + other._get_axes()
        for axis in axes:
            if not self._overlap(self._project_onto_axis(axis),
other._project_onto_axis(axis)):
                return False
        return True




if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值

    plt.figure(figsize=(10, 7))
    ax = plt.subplot(111, polar=True)  # 创建极坐标图
    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测
```

```
    # 下面这些是为计算矩形位置的
    handtheta = []#记录每个把手的 theta 角度
    rect = [] #记录每个矩形的中心点位置
    rectangle = [] #记录每个矩形的旋转角度

    #定义特定点 xy 变量名

    # 下面这俩是用来求速度的
    bodyx = []
    bodyy = []




    # 定义特定点速度

    ax.set_title("Equidistant Spiral")


    #print(dt)

    for i in range(4124738370,4500000000):  # 计算 300 个点
        ax.clear()
        i = i/10000000
        ax.plot(theta, rot(theta, b), 'b-')
        # 使用 fsolve 找到满足条件的 theta

        tail = []
        ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True)
        if message == 'The solution converged.':


            ax.plot(ans, rot(ans, b), 'ro')  # 绘制每个点
            search_initial = ans  # 更新初始猜测为当前解
            upperangle = ans
            headx, heady = angtoxy(ans, rot(ans, b))
            #print("龙首坐标:", [headx, heady])
        else:
            print(f"Warning: fsolve did not, converge for i={i}")
        # ax.plot(ans,rot(ans,b),color='red')
        handtheta.append(ans)
        count = 0
```

```
        numberinrow = 0
        for j in range(1, 225):

            if j == 1:
                L = 2.86
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
            else:
                L = 1.65
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                tail.append(tailtheta)
            ax.scatter(tailtheta, rot(tailtheta, b))
            upperangle = tailtheta
            handtheta.append(tailtheta)
            count = count + 1
            if tailtheta - ans < np.pi*3:
                numberinrow += 1
            # print(count)
            plt.draw()
        print(numberinrow)
        #print(handtheta)
        # 接下来处理每个矩形的坐标、位置
        # 由于第 i 个矩形由第 i 和第 i+1 个把手的位置所确定，因此需要一些计算
        # 把手的位置来源于以上的 head 和 tail
        # handtheta 是每个把手的 theta 角度
        # 由于每个矩形的位置是由两个把手的位置所确定的，因此需要一些计算
        rect = []
        rectangle = []
        for j in range(len(handtheta)-1):
            # 计算矩形的中心点位置
            #print(j)
            midposupper = angtoxy(handtheta[j],rot(handtheta[j],b))
            midposlater = angtoxy(handtheta[j+1],rot(handtheta[j+1],b))

            center_x = (midposupper[0]+midposlater[0])/2
            center_y = (midposupper[1]+midposlater[1])/2
            rect.append((center_x,center_y))

            # 计算矩形的旋转角度(弧度制？)
            turningangle = np.arctan2(midposlater[1]-
midposupper[1],midposlater[0]-midposupper[0])
            rectangle.append(turningangle)
            #print((midposupper[0]+midposlater[0])/2)
```

```
        handtheta.clear()
        # 现在记录了每个矩形的位置和旋转角度
        # 接下来就是对是否碰撞进行检测
        # 由于之前已验证 300s 前不会碰撞，因此从 300s 后再开始检测
        #print(len(rect))

        if i > 0:
            for j in range(0,3):
                if j == 0:
                    rect1 = OBB(rect[j],rectangle[j],3.41,0.3)
                else:
                    rect1 = OBB(rect[j],rectangle[j],2.20,0.3)

                for k in range(j+2,len(rect)):

                    if k == 0:
                        rect2 = OBB(rect[k],rectangle[k],3.41,0.3)
                    else:
                        rect2 = OBB(rect[k],rectangle[k],2.20,0.3)
                    if rect1.is_colliding(rect2):
                        print(f"Warning: Collision detected at i(s)={i},
j(head)={j}, k={k}")
                        print("The R = ",rot(ans,b))
                        a = input("PAUSE")
                        break


        plt.pause(0.01)
```

计算位置.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd

from openpyxl import load_workbook

# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
```

```
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i

# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-10):
    """
    Find the polar coordinates of a point on the spiral r = b * theta
that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be equal
to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition
```

```
    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance, full_output=True)
    #print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution within
the given tolerance.")

        # Calculate the corresponding r_tail value
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        plt.show()
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail
def angtoxy(theta,r):
    return [r*np.cos(theta),r*np.sin(theta)]

#定义了一个类，它能计算两个旋转矩形是否碰撞

class RotatedRectangle:
    def __init__(self, center, angle, width, height):
        self.center = center  # 中心坐标 (x, y)
        self.angle = angle  # 旋转角度（弧度）
        self.width = width  # 宽度
        self.height = height  # 高度

        # 预先计算矩形的顶点
        self.update_vertices()

    def update_vertices(self):
        # 定义矩形的轴
        axis_x = [np.cos(self.angle), np.sin(self.angle)]
```

```
        axis_y = [-np.sin(self.angle), np.cos(self.angle)]

        # 计算矩形的四个顶点
        self.vertices = [
            [self.center[0] + self.width / 2 * axis_x[0] + self.height
/ 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] + self.height
/ 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] + self.height
/ 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] + self.height
/ 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] - self.height
/ 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] - self.height
/ 2 * axis_y[1]],
            [self.center[0] + self.width / 2 * axis_x[0] - self.height
/ 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] - self.height
/ 2 * axis_y[1]]
        ]

    def dot(self, v, w):
        return v[0] * w[0] + v[1] * w[1]

    def project(self, a, b):
        return (self.dot(a, b) / self.dot(b, b)) * b

    def on_segment(self, p, q, r):
        return q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0]) and
\
            q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])

    def orientation(self, p, q, r):
        val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] -
q[1])
        if val == 0:
            return 0  # collinear
        elif val > 0:
            return 1  # clock
        else:
            return 2  # counterclock

    def do_intersect(self, p1, q1, p2, q2):
```

```
        o1 = self.orientation(p1, q1, p2)
        o2 = self.orientation(p1, q1, q2)
        o3 = self.orientation(p2, q2, p1)
        o4 = self.orientation(p2, q2, q1)

        if o1 != o2 and o3 != o4:
            return True

        if o1 == 0 and self.on_segment(p1, p2, q1):
            return True
        if o2 == 0 and self.on_segment(p1, q2, q1):
            return True
        if o3 == 0 and self.on_segment(p2, p1, q2):
            return True
        if o4 == 0 and self.on_segment(p2, q1, q2):
            return True

        return False

    def collides_with(self, other):
        # 检查边-边碰撞
        for i in range(4):
            for j in range(4):
                if self.do_intersect(self.vertices[i], self.vertices[(i
+ 1) % 4],
                                     other.vertices[j], other.vertices[(j +
1) % 4]):
                    return True

        return False

def velocitycal(velupper,theta,thetaplus1):   # 注意 plus1 意味着向龙尾靠
近 1
    xiplus1,yiplus1 = angtoxy(thetaplus1,rot(thetaplus1,b))
    xi,yi = angtoxy(theta,rot(theta,b))
    ki = np.array((xiplus1-xi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))),(yiplus1-yi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))))
    viplus1 = np.array((np.cos(thetaplus1-
thetaplus1*np.sin(thetaplus1)))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetaplus
1*np.cos(thetaplus1))),(np.sin(thetaplus1)+thetaplus1*np.cos(thetaplu
s1))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetaplus
```

```
1*np.cos(thetaplus1))))
    vi = np.array((np.cos(theta-
theta*np.sin(theta)))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))),(n
p.sin(theta)+theta*np.cos(theta))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))))
    return np.dot(ki,vi)*velupper/np.dot(ki,viplus1)

if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值

    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    #定义特定点 xy 变量名

    # 下面这俩是用来求速度的
    bodyx = []
    bodyy = []
    #saveform = []
    # 定义特定点速度


    timemark = 0
    dt = np.linspace(timemark-0.000001,timemark+0.000001,2)
    print(dt)


    # 定义特定点速度

    timemark = 0
    dt = np.linspace(timemark - 0.000001, timemark + 0.000001, 2)
    print(dt)
    savepath = 'C:\\Users\\31827\\Downloads\\result2.xlsx'
    book = load_workbook(savepath)
    writer = pd.ExcelWriter(savepath, engine='openpyxl')
    writer._book = book
    saveformall = []
    # 碰撞时刻：412.473838
    for i in range(300):  # 计算 300 个点
```

```
        # 使用 fsolve 找到满足条件的 theta

        tail = []

        saveformx = []
        saveformy = []
        ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True)
        if message == 'The solution converged.':
            headtheta.append(ans)


            search_initial = ans  # 更新初始猜测为当前解
            upperangle = ans
            headx, heady = angtoxy(ans, rot(ans, b))
            print("龙首坐标:", [headx, heady])
            saveformx.append(headx)
            saveformy.append(heady)
        else:
            print(f"Warning: fsolve did not, converge for i={i}")
        # ax.plot(ans,rot(ans,b),color='red')
        count = 0
        for j in range(1, 224):

            if j == 1:
                L = 2.86
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                tail.append(tailtheta)
            else:
                L = 1.65
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                tail.append(tailtheta)

            upperangle = tailtheta
            count = count + 1
            # print(count)


        for j in range(len(tail)):
            saveformx.append(angtoxy(tail[j],rot(tail[j],b))[0])
            saveformy.append(angtoxy(tail[j],rot(tail[j],b))[1])
```

```
            saveformx[j] =
np.format_float_scientific(saveformx[j],precision=6)
            saveformy[j] = np.format_float_scientific(saveformy[j],
precision=6)

        saved1ct = {"横坐标 x (m)":saveformx,"纵坐标 y (m)":saveformy}
        df = pd.DataFrame(saved1ct)
        df.to_excel(writer, sheet_name='Sheet1', startcol=1,
header=1,index=False)
        writer._save()

    #saveformall =
np.format_float_scientific(saveformall,precision=6)
    # 利用字典生成式生成表内容并保存
    #savedict = {f"{l} s": saveformall[l] for l in range(0, 301)}
    #df = pd.DataFrame(savedict)
    #df.to_excel(writer, sheet_name='位置', startcol=1, header=1)
    #print(bodyx, bodyy)
    #writer._save()
    writer.close()
    # print(tail)
```

计算速度.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd
from openpyxl import load_workbook


# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result
```

```
# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i


# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-10):
    """
    Find the polar coordinates of a point on the spiral r = b * theta
that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be equal
to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
```

```
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance,
                                            full_output=True)
    # print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution within
the given tolerance.")

        # Calculate the corresponding r_tail value
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail


def angtoxy(theta, r):
    return [r * np.cos(theta), r * np.sin(theta)]


# 定义了一个类，它能计算两个旋转矩形是否碰撞

class RotatedRectangle:
    def __init__(self, center, angle, width, height):
        self.center = center  # 中心坐标 (x, y)
        self.angle = angle  # 旋转角度（弧度）
        self.width = width  # 宽度
        self.height = height  # 高度

        # 预先计算矩形的顶点
        self.update_vertices()

    def update_vertices(self):
```

```
        # 定义矩形的轴
        axis_x = [np.cos(self.angle), np.sin(self.angle)]
        axis_y = [-np.sin(self.angle), np.cos(self.angle)]

        # 计算矩形的四个顶点
        self.vertices = [
            [self.center[0] + self.width / 2 * axis_x[0] + self.height
/ 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] + self.height
/ 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] + self.height
/ 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] + self.height
/ 2 * axis_y[1]],
            [self.center[0] - self.width / 2 * axis_x[0] - self.height
/ 2 * axis_y[0],
             self.center[1] - self.width / 2 * axis_x[1] - self.height
/ 2 * axis_y[1]],
            [self.center[0] + self.width / 2 * axis_x[0] - self.height
/ 2 * axis_y[0],
             self.center[1] + self.width / 2 * axis_x[1] - self.height
/ 2 * axis_y[1]]
        ]

    def dot(self, v, w):
        return v[0] * w[0] + v[1] * w[1]

    def project(self, a, b):
        return (self.dot(a, b) / self.dot(b, b)) * b

    def on_segment(self, p, q, r):
        return q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0]) and
\
            q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])

    def orientation(self, p, q, r):
        val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] -
q[1])
        if val == 0:
            return 0  # collinear
        elif val > 0:
            return 1  # clock
        else:
            return 2  # counterclock
```

```
    def do_intersect(self, p1, q1, p2, q2):
        o1 = self.orientation(p1, q1, p2)
        o2 = self.orientation(p1, q1, q2)
        o3 = self.orientation(p2, q2, p1)
        o4 = self.orientation(p2, q2, q1)

        if o1 != o2 and o3 != o4:
            return True

        if o1 == 0 and self.on_segment(p1, p2, q1):
            return True
        if o2 == 0 and self.on_segment(p1, q2, q1):
            return True
        if o3 == 0 and self.on_segment(p2, p1, q2):
            return True
        if o4 == 0 and self.on_segment(p2, q1, q2):
            return True

        return False

    def collides_with(self, other):
        # 检查边-边碰撞
        for i in range(4):
            for j in range(4):
                if self.do_intersect(self.vertices[i], self.vertices[(i
+ 1) % 4],
                                     other.vertices[j], other.vertices[(j +
1) % 4]):
                    return True

        return False

def velocitycal(velupper,theta,thetaplus1):   # 注意 plus1 意味着向龙尾靠
近 1
    xiplus1,yiplus1 = angtoxy(thetaplus1,rot(thetaplus1,b))
    xi,yi = angtoxy(theta,rot(theta,b))
    ki = ((xiplus1-xi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))),(yiplus1-yi)/(np.sqrt(np.square(xiplus1-
xi)+np.square(yiplus1-yi))))
    viplus1 = ((np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetaplus
1*np.cos(thetaplus1))),(np.sin(thetaplus1)+thetaplus1*np.cos(thetaplu
```

```
s1))/np.sqrt(np.square(np.cos(thetaplus1)-
thetaplus1*np.sin(thetaplus1))+np.square(np.sin(thetaplus1)+thetaplus
1*np.cos(thetaplus1))))
    vi = ((np.cos(theta)-
theta*np.sin(theta))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))),(n
p.sin(theta)+theta*np.cos(theta))/np.sqrt(np.square(np.cos(theta)-
theta*np.sin(theta))+np.square(np.sin(theta)+theta*np.cos(theta))))

    resultup = 0
    resultdown = 0
    for i in range(2):
        resultup += ki[i]*vi[i]
        resultdown += ki[i]*viplus1[i]

    return abs(resultup*velupper/resultdown)

if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值



    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    # 定义特定点 xy 变量名
    headx, heady, bodyx, bodyy = 0, 0, 0, 0
    # 下面这俩是用来求速度的
    bodyx1 = []
    bodyy1 = []

    bodytheta1 = []
    bodyr1 = []
    bodytheta2 = []
    bodyr2 = []
    #  下面是保存数据的部分
    veloall = []
    savepath = 'C:\\Users\\31827\\Downloads\\result2.xlsx'
    book = load_workbook(savepath)
    writer = pd.ExcelWriter(savepath, engine='openpyxl')
```

```
    writer._book = book

    timemark = 0
    timeinterval = 0.7
    # 碰撞时刻：412.473838
    for i in range(300):  # 计算 300 个点
        timemark = i
        dt = np.linspace(timemark - timeinterval, timemark +
timeinterval, 2)
        tail = []
        bodytheta1 = []
        bodyr1 = []
        bodytheta2 = []
        bodyr2 = []
        ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True,xtol=1e-8)
        if message == 'The solution converged.':
            headtheta.append(ans)
            search_initial = ans  # 更新初始猜测为当前解
            upperangle = ans
            headx, heady = angtoxy(ans, rot(ans, b))
            #tail.append(ans)
            print("龙首坐标:", [headx, heady])
        else:
            print(f"Warning: fsolve did not, converge for i={i}")
        #对 dt 求速度
        for j in range(1, 225):
            if j == 1:
                L = 2.86
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                tail.append(tailtheta)
            else:
                L = 1.65
                tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                tail.append(tailtheta)
            upperangle = tailtheta

        print(len(tail))
        velocity = []
        velotem = 0
        inherit = 0
        for j in range(0,224):
```

```
            if j == 0:
                velotem = velocitycal(1,ans,tail[j])
            else:
                velotem = velocitycal(inherit,tail[j-1],tail[j])
            inherit = velotem
            velotem = np.format_float_scientific(velotem,precision=6)
            velocity.append(velotem)

    print('lens=',len(veloall))

    savedict = {"速度 (m/s)":velocity}
    df = pd.DataFrame(savedict)
    print(df)
    df.to_excel(writer, sheet_name='Sheet1', startcol=3,
header=1,index=False)
    writer._save()
    writer.close()
```

3.问题三代码

优化螺距.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd
from pandas import DataFrame
from openpyxl import load_workbook
import math
# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
```

```
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i

# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-10):
    """
    Find the polar coordinates of a point on the spiral r = b * theta
that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be equal
to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance, full_output=True)
    #print(theta_tail)
```

```
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution within
the given tolerance.")

        # Calculate the corresponding r_tail value
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        plt.show()
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail
def angtoxy(theta,r):
    return [r*np.cos(theta),r*np.sin(theta)]

#定义了一个类，它能计算两个旋转矩形是否碰撞


class OBB:
    def __init__(self, center, angle,width, height):
        self.center = np.array(center)
        self.width = width
        self.height = height
        self.angle = angle
        self.half_width = width / 2
        self.half_height = height / 2
        self.orientation = np.array([np.cos(angle), np.sin(angle)])
        self.corners = self._get_corners()

    def _get_corners(self):
        cos_angle = np.cos(self.angle)
        sin_angle = np.sin(self.angle)
        dx = self.half_width * cos_angle
        dy = self.half_width * sin_angle
        hx = self.half_height * sin_angle
```

```
        hy = self.half_height * cos_angle

        corners = [
            self.center + np.array([dx - hx, dy + hy]),
            self.center + np.array([dx + hx, dy - hy]),
            #self.center + np.array([-dx - hx, -dy + hy]),
            self.center + np.array([-dx + hx, -dy - hy]),
            self.center + np.array([-dx - hx, -dy + hy])
        ]
        return np.array(corners)

    def _get_axes(self):
        edges = [self.corners[i] - self.corners[i - 1] for i in
range(4)]
        return [np.array([-edge[1], edge[0]]) / np.linalg.norm(edge)
for edge in edges]

    def _project_onto_axis(self, axis):
        axis = axis.flatten()
        projections = [np.dot(corner.flatten(), axis) for corner in
self.corners]
        return [min(projections), max(projections)]

    def _overlap(self, projection1, projection2):
        return projection1[0] <= projection2[1] and projection2[0] <=
projection1[1]

    def is_colliding(self, other):
        axes = self._get_axes() + other._get_axes()
        for axis in axes:
            if not self._overlap(self._project_onto_axis(axis),
other._project_onto_axis(axis)):
                return False
        return True




if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值
```

```
    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    # 下面这些是为计算矩形位置的
    handtheta = []#记录每个把手的 theta 角度
    rect = [] #记录每个矩形的中心点位置
    rectangle = [] #记录每个矩形的旋转角度

    #定义特定点 xy 变量名

    # 下面这俩是用来求速度的
    bodyx = []
    bodyy = []

    b = 0.06798999999999998
    while b > 0:
        iscounter = False
        iscollision = False
        isfind = False
        f = 412
        for i in range(f*10):  # 计算 300 个点
            # 使用 fsolve 找到满足条件的 theta
            i = i/10
            tail = []
            ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True)
            if message == 'The solution converged.':

                search_initial = ans  # 更新初始猜测为当前解
                upperangle = ans
                headx, heady = angtoxy(ans, rot(ans, b))
                # print("龙首坐标:", [headx, heady])
            else:
                print(f"Warning: fsolve did not, converge for i={i}")
            # ax.plot(ans,rot(ans,b),color='red')
            handtheta = [ans]
            count = 0
            numberinrow = 0
            if abs(rot(ans, b) - 4.5) < 1e-2 or rot(ans, b) < 4.5:
                iscounter = True
```

```
            for j in range(1, 225):

                if j == 1:
                    L = 2.86
                    tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                else:
                    L = 1.65
                    tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                    tail.append(tailtheta)

                upperangle = tailtheta
                handtheta.append(tailtheta)
                count = count + 1
                if tailtheta - ans < np.pi * 3:
                    numberinrow += 1
                # print(count)

            # print(numberinrow)
            # print(handtheta)
            # 接下来处理每个矩形的坐标、位置
            # 由于第 i 个矩形由第 i 和第 i+1 个把手的位置所确定，因此需要一些计算
            # 把手的位置来源于以上的 head 和 tail
            # handtheta 是每个把手的 theta 角度
            # 由于每个矩形的位置是由两个把手的位置所确定的，因此需要一些计算
            rect = []
            rectangle = []
            for j in range(len(handtheta) - 1):
                # 计算矩形的中心点位置
                # print(j)
                midposupper = angtoxy(handtheta[j], rot(handtheta[j],
b))
                midposlater = angtoxy(handtheta[j + 1], rot(handtheta[j
+ 1], b))

                center_x = (midposupper[0] + midposlater[0]) / 2
                center_y = (midposupper[1] + midposlater[1]) / 2
                rect.append((center_x, center_y))

                # 计算矩形的旋转角度(弧度制？)
                turningangle = np.arctan2(midposlater[1] -
midposupper[1], midposlater[0] - midposupper[0])
                rectangle.append(turningangle)
```

```
                # print((midposupper[0]+midposlater[0])/2)
            handtheta.clear()
            # 现在记录了每个矩形的位置和旋转角度
            # 接下来就是对是否碰撞进行检测
            # 由于之前已验证 300s 前不会碰撞，因此从 300s 后再开始检测
            # print(len(rect))

            if iscounter:
                for j in range(0, 3):
                    if j == 0:
                        rect1 = OBB(rect[j], rectangle[j], 3.41, 0.3)
                    else:
                        rect1 = OBB(rect[j], rectangle[j], 2.20, 0.3)

                    for k in range(j + 2, len(rect)):

                        if k == 0:
                            rect2 = OBB(rect[k], rectangle[k], 3.41, 0.3)
                        else:
                            rect2 = OBB(rect[k], rectangle[k], 2.20, 0.3)
                        if rect1.is_colliding(rect2):
                            print(f"Warning: Collision detected at
i(s)={i}, j(head)={j}, k={k}")
                            print("The R = ", rot(ans, b))
                            iscollision = True
                            break
                    if iscollision == True:
                        break
                break
        if not(iscounter == True and iscollision == False):
            isfind = True

        # 0.001 精度发现 b = 0.0675352
        # 0.0001 b = 0.0679351
        # 0.00001 b= 0.06799000000000001
        # 0.000001 b = 0.06798899999999998
        # 0.0000001 b = 0.0679895
        print("b =", b)
        if isfind == True:
            print("临界点发现！")
            btrue = b + 0.0000001
            print("此时 b=", btrue)
            output = np.pi * 2 * btrue
            print("螺距为",output)
```

```
            break
        b -= 0.0000001
```

优化螺距-高精度.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize
import pandas as pd
from pandas import DataFrame
from openpyxl import load_workbook
import math
# 这是计算路程里的被积函数
def curvedif(theta):
    return (1 + theta ** 2) ** 0.5


# 这是计算龙头路程的函数，采用积分的方法
def S(theta0, theta1, b, fun):
    result, _ = integrate.quad(fun, theta0, theta1)
    return b * result


# 这是等距螺线的极坐标方程
def rot(theta, b):
    return b * theta


# 定义用于求解的函数
def Si(theta, i, theta1, b, fun):
    return S(theta, theta1, b, fun) - i

# 定义把手和把手距离函数并求解
def find_point_on_spiral(theta_head, r_head, L, b, tolerance=1e-10):
    """
    Find the polar coordinates of a point on the spiral r = b * theta
that is a distance L away from the head.

    Parameters:
    - theta_head: float, the polar angle of the head.
    - r_head: float, the radial distance of the head (should be equal
to b * theta_head if on the spiral).
    - L: float, the linear distance from the head to the desired
point.
    - b: float, the parameter of the spiral r = b * theta.
    - tolerance: float, the tolerance for the convergence of fsolve
```

```
(default is 1e-6).

    Returns:
    - theta_tail: float, the polar angle of the found point.
    - r_tail: float, the radial distance of the found point.
    """

    # Define a function that returns the difference between the
actual distance and the target distance L
    def distance_diff(theta_tail):
        r_tail = b * theta_tail
        distance = np.sqrt(r_head ** 2 + r_tail ** 2 - 2 * r_head *
r_tail * np.cos((theta_head - theta_tail)))
        return distance - L

        # Use fsolve to find the theta_tail that satisfies the
distance condition

    theta_tail_initial_guess = theta_head  # Initial guess (may need
adjustment based on the spiral)
    theta_tail, _, _, message = optimize.fsolve(distance_diff,
theta_tail_initial_guess, xtol=tolerance, full_output=True)
    #print(theta_tail)
    # Check if fsolve converged successfully
    if message != 'The solution converged.':
        plt.show()
        raise ValueError("fsolve did not converge to a solution within
the given tolerance.")

        # Calculate the corresponding r_tail value
    r_tail = b * theta_tail

    # Ensure theta_tail is in the expected range (theta_head <
theta_tail)
    # Note: We do not check for theta_tail < 2*pi here because the
spiral can extend beyond 2*pi
    #       If you want to restrict the solution to a single
revolution, you can add this check.
    if not (theta_head < theta_tail):
        plt.show()
        raise ValueError("The found θ_tail is not greater than
θ_head.")

    return theta_tail, r_tail
```

```
def angtoxy(theta,r):
    return [r*np.cos(theta),r*np.sin(theta)]

#定义了一个类，它能计算两个旋转矩形是否碰撞


class OBB:
    def __init__(self, center, angle,width, height):
        self.center = np.array(center)
        self.width = width
        self.height = height
        self.angle = angle
        self.half_width = width / 2
        self.half_height = height / 2
        self.orientation = np.array([np.cos(angle), np.sin(angle)])
        self.corners = self._get_corners()

    def _get_corners(self):
        cos_angle = np.cos(self.angle)
        sin_angle = np.sin(self.angle)
        dx = self.half_width * cos_angle
        dy = self.half_width * sin_angle
        hx = self.half_height * sin_angle
        hy = self.half_height * cos_angle

        corners = [
            self.center + np.array([dx - hx, dy + hy]),
            self.center + np.array([dx + hx, dy - hy]),
            #self.center + np.array([-dx - hx, -dy + hy]),
            self.center + np.array([-dx + hx, -dy - hy]),
            self.center + np.array([-dx - hx, -dy + hy])
        ]
        return np.array(corners)

    def _get_axes(self):
        edges = [self.corners[i] - self.corners[i - 1] for i in
range(4)]
        return [np.array([-edge[1], edge[0]]) / np.linalg.norm(edge)
for edge in edges]

    def _project_onto_axis(self, axis):
        axis = axis.flatten()
        projections = [np.dot(corner.flatten(), axis) for corner in
self.corners]
```

```
        return [min(projections), max(projections)]

    def _overlap(self, projection1, projection2):
        return projection1[0] <= projection2[1] and projection2[0] <=
projection1[1]

    def is_colliding(self, other):
        axes = self._get_axes() + other._get_axes()
        for axis in axes:
            if not self._overlap(self._project_onto_axis(axis),
other._project_onto_axis(axis)):
                return False
        return True




if __name__ == '__main__':
    p = 0.55
    pi = np.pi
    b = p / (2 * pi)  # 修改 b 的计算
    theta = np.linspace(0, 32 * pi, 2000)  # 生成 theta 的值


    upperangle = 0
    headtheta = []
    search_initial = 0  # 初始猜测

    # 下面这些是为计算矩形位置的
    handtheta = []#记录每个把手的 theta 角度
    rect = [] #记录每个矩形的中心点位置
    rectangle = [] #记录每个矩形的旋转角度

    #定义特定点 xy 变量名

    # 下面这俩是用来求速度的
    bodyx = []
    bodyy = []

    b = 0.0662829
    while b > 0:
        iscounter = False
        iscollision = False
        isfind = False
```

```
        f = 412
        for i in range(f*10):  # 计算 300 个点
            # 使用 fsolve 找到满足条件的 theta
            i = i/10
            tail = []
            ans, _, _, message = optimize.fsolve(Si, search_initial,
args=(i, 32 * pi, b, curvedif), full_output=True)
            if message == 'The solution converged.':

                search_initial = ans  # 更新初始猜测为当前解
                upperangle = ans
                headx, heady = angtoxy(ans, rot(ans, b))
                # print("龙首坐标:", [headx, heady])
            else:
                print(f"Warning: fsolve did not, converge for i={i}")
            # ax.plot(ans,rot(ans,b),color='red')
            handtheta = [ans]
            count = 0
            numberinrow = 0
            if abs(rot(ans, b) - 4.5) < 1e-6 or rot(ans, b) < 4.5:
                iscounter = True

            for j in range(1, 225):

                if j == 1:
                    L = 2.86
                    tailtheta, _ = find_point_on_spiral(upperangle,
rot(ans, b), L, b)
                else:
                    L = 1.65
                    tailtheta, _ = find_point_on_spiral(upperangle,
rot(upperangle, b), L, b)
                    tail.append(tailtheta)

                upperangle = tailtheta
                handtheta.append(tailtheta)
                count = count + 1
                if tailtheta - ans < np.pi * 3:
                    numberinrow += 1
                # print(count)

            # print(numberinrow)
            # print(handtheta)
            # 接下来处理每个矩形的坐标、位置
```

```
            # 由于第 i 个矩形由第 i 和第 i+1 个把手的位置所确定，因此需要一些计算
            # 把手的位置来源于以上的 head 和 tail
            # handtheta 是每个把手的 theta 角度
            # 由于每个矩形的位置是由两个把手的位置所确定的，因此需要一些计算
            rect = []
            rectangle = []
            for j in range(len(handtheta) - 1):
                # 计算矩形的中心点位置
                # print(j)
                midposupper = angtoxy(handtheta[j], rot(handtheta[j],
b))
                midposlater = angtoxy(handtheta[j + 1], rot(handtheta[j
+ 1], b))

                center_x = (midposupper[0] + midposlater[0]) / 2
                center_y = (midposupper[1] + midposlater[1]) / 2
                rect.append((center_x, center_y))

                # 计算矩形的旋转角度(弧度制？)
                turningangle = np.arctan2(midposlater[1] -
midposupper[1], midposlater[0] - midposupper[0])
                rectangle.append(turningangle)
                # print((midposupper[0]+midposlater[0])/2)
            handtheta.clear()
            # 现在记录了每个矩形的位置和旋转角度
            # 接下来就是对是否碰撞进行检测
            # 由于之前已验证 300s 前不会碰撞，因此从 300s 后再开始检测
            # print(len(rect))

            if iscounter:
                for j in range(0, 3):
                    if j == 0:
                        rect1 = OBB(rect[j], rectangle[j], 3.41, 0.3)
                    else:
                        rect1 = OBB(rect[j], rectangle[j], 2.20, 0.3)

                    for k in range(j + 2, len(rect)):

                        if k == 0:
                            rect2 = OBB(rect[k], rectangle[k], 3.41, 0.3)
                        else:
                            rect2 = OBB(rect[k], rectangle[k], 2.20, 0.3)
                        if rect1.is_colliding(rect2):
                            print(f"Warning: Collision detected at
```

```
i(s)={i}, j(head)={j}, k={k}")
                        print("The R = ", rot(ans, b))
                        iscollision = True
                        break
                if iscollision == True:
                    break
            break
        if not(iscounter == True and iscollision == False):
            isfind = True

        # 0.001 精度发现 b = 0.06653521870054242
        # 0.0001 b = 0.06579999999999997
        # 0.00001 b= 0.06586000000000002
        # 0.000001 b = 0.06585499999999997
        # 0.0000001 b = 0.0679895
        print("b =", b)
        if isfind == True:
            print("临界点发现！")
            btrue = b + 0.0000001
            print("此时 b=", btrue)
            output = np.pi * 2 * btrue
            print("螺距为",output)
            break
        b -= 0.0000001
```