“板凳龙”表演的运动学分析

摘要

本文从对“板凳龙”的建模抽象出发，结合极坐标系对“板凳龙”表演过程中的运动学性质进行分析。板凳位置的求解均可转化为曲线的联立求解，再在位置计算方法的基础上建立了基于微元法的瞬时速度计算方法，为后续分析提供了定量计算的理论依据。接下来又分析了“板凳龙”前进时的碰撞条件、调头路线设计以及行进中各板凳之间的速度关系，帮助在盘入盘出过程中减小盘龙所需面积，加快盘龙速度，提高观赏性。

对于问题一，建立**极坐标系**并在此基础上建立了“板凳龙”的**折线模型**。龙头前把手初始位置已知。对于每个位置待定的把手，根据把手极角可唯一确定从原点沿等距螺线到把手的弧长，进而确定把手的位置。对于每个把手，使用**二分法**求出其后继把手的位置和沿等距螺线运动一段时间后的位置，进而递推出每个时间点整条“板凳龙”的位置，详细结果见表 1。对于把手速度，选择一个足够短的时间段，在满足精度要求的前提下以平均速度代替瞬时速度得到结果，详细结果见表 2。

对于问题二，在问题一的基础上加入板凳宽度进行建模。经过分析，我们只需求出碰撞临界点时**从原点沿等距螺线到龙头前把手的弧长**即可。于是使用二分求出龙头前把手运动的弧长，再由龙头前把手的速度求得中止时刻为开始运动后的 412.473838s，进而求出龙头前把手及后续把手的位置，并使用与第一问类似的方法求解各把手的速度。详细结果见表 3。对于判断两长方形是否碰撞的问题，可以转化为构成两长方形的线段是否相交的问题。

对于问题三，问题需要求解使得龙头前把手沿着螺线盘入到调头空间边界的最小螺距。经过分析，采用与问题二相同的建模方式，使用二分法求解最小螺距。由于算法只能判断离散时间点上的碰撞，采用多次二分，对每次二分逐渐缩小离散时间点的间隔，同时通过观察其收敛趋势调整二分算法的边界，在保证精度的基础上优化算法耗时。

对于问题四，问题提到调整圆弧，说明调整后的曲线必定也是圆弧。经过证明可以得出两圆弧组成的调头路线长度恒定。取题设所示圆弧进行建模，将盘入螺线、调头路线、盘出螺线通过以龙头前把手走过的弧长为自变量的函数联系起来，将完整路线视为一条曲线，即可将问题转变为问题一。适当调整短时间段取值来获取表现更好的方案。最终求得题目所需的时间段里各个板凳的位置和速度，详细结果见表 4。

对于问题五，由于问题四的计算结果只含整数时间采样点，直接等比例缩放并不能取到准确结果。观察问题四计算结果表格，发现速度最大值只会出现在龙头从调头路线进入盘出螺线时的第 1 节龙身，且最大速度关于龙头速度单调，于是改为使用二分法求解速度。同时使用问题三类似的多次二分方式优化答案。最终得到龙头的最大行进速度为 1.283843m/s。

关键词：运动学分析、二分法、微元法、采样、碰撞检测

一、 问题重述

“板凳龙”作为我国的传统地方民俗文化活动，是我国传统文化传承的组成部分，有着独特的文化内涵。对此，我们需要研究“板凳龙”活动中的行进路线和速度来提高观赏性。

“板凳龙”结构组成：“板凳龙”由包含龙头、龙尾、龙身的多节板凳组成。相邻板凳之间由一次穿过两板凳的把手连接。

“板凳龙”表演原理：进行“板凳龙”盘龙表演时，舞龙队举起把手带动板凳按规定路线运动。由表演者控制龙的运动轨迹以及速度。

问题一：已知舞龙的初始位置、舞龙的路线以及龙头前把手的行进速度。建立“板凳龙”的运动模型，求出从出发起 300 秒内每一秒各板凳的前把手以及最后一个板凳的后把手的位置和速度。

问题二：在舞龙过程中，板凳之间不能碰撞，所以不能无限制盘入。在第一问的运动约束条件下，确定盘入的中止时刻，以及中止时各个板凳的前后把手的运动学参数。

问题三：为了实现不发生碰撞的从盘入到盘出的掉头操作，我们以螺线中心为圆心确定一个圆形范围作为调头空间用于调头。要实现这一过程，至少需要龙头前把手可以在碰撞前沿着螺线到达调头空间边界。而螺距越小越容易发生碰撞，所以需要求出能实现这一过程的最小螺距。

问题四：对于问题三的调头空间，考虑具体调头过程，要求调头后“板凳龙”能沿着与原盘入螺线中心对称的一条盘出螺线进行盘出。为了实现调头，需要在调头空间内设计一段调头曲线。最初给定两段圆弧作为调头曲线，使得“板凳龙”完整的运动路径各部分相切。在给定调头路径的基础上，我们需要分析是否能够通过调整圆弧，使得调整后的调头曲线相较原曲线缩短，并在确定龙头前把手运动速度与调头方案后，计算出调头前后一段时间内“板凳龙”的各把手的运动情况。

问题五：考虑“板凳龙”在运动过程中各把手都存在最大速度。试求龙头的最大行进速度，使得所有把手在问题四的运动过程中不超过给定的速度上限。

二、 问题分析

2.1 问题一分析

问题一需要确定“板凳龙”在整个运动过程中，各把手在给定时刻的位置与瞬时速度，为此我们将整条“板凳龙”转化为以**把手为端点的折线段**来简化分析。

考虑不同时间各把手的坐标。首先在**极坐标**中利用定积分与**二分答案**计算龙头前把

手运动 t s 后的位置，再由每个板凳前后把手中心的距离固定，将下一个把手的位置求解转化为螺线与半径为把手距离的圆的交点问题，**递推**地求出每个龙身前把手以及龙尾后把手的位置。

再考虑不同时间各把手的瞬时速度。在这里我们选用**微元法**，以研究位置附近**一个足够小时间邻域内的平均速度**表示该位置的瞬时速度。

2.2 问题二分析

问题二需要确定按问题一的路线盘入时，板凳之间第一次发生碰撞的时刻。不同于第一问，板凳间是否碰撞需要考虑整个板凳矩形的边界，即与板凳外边沿的位置相关。所以，在这里我们考虑把整条“板凳龙”抽象为**各板凳边沿矩形构成的一系列矩形**，再结合问题一设计的算法分析其运动情况。

经过对顺时针盘入过程的模拟，我们发现：随着运动轨迹的曲率半径逐渐减小，整条“板凳龙”会在我们所求的第一次碰撞前后从无重叠转变为有重叠。为了计算碰撞时刻以及该时刻各板凳的运动状态，我们使用**二分查找算法**来求解第一次发生碰撞的时刻。

为了在二分算法中确定如何更新二分边界，我们需要对特定的时刻判断碰撞是否发生。对此，我们利用**快速排斥实验**与**跨立实验**对板凳的各条边判断是否相交，再遍历所有可能的板凳碰撞情况，确定整条“板凳龙”中是否发生碰撞。

2.3 问题三分析

问题三需要最小化螺距，使得“板凳龙”能够沿着螺线到达调头空间边界而不发生碰撞。因为螺距越小越容易在到达空间边界之前发生碰撞，所以使用**二分查找算法**确定碰撞临界条件，进而求得满足条件的最小螺距。

对于特定螺距的运动过程，为了判断是否会发生碰撞，我们对连续的运动过程进行**采样**，用离散的采样时刻模拟连续的运动过程，并以各采样点是否发生碰撞为依据，来近似判断连续运动过程中是否会发生碰撞。

采样间隔和**二分查找的上下界**是影响二分查找效率与结果的两个参数。我们利用**超参数优化**的思想优化参数选择，最终得到符合题目条件的答案。

2.4 问题四分析

首先考虑圆弧调整问题。我们认为调头曲线始终由两段圆弧构成，且“板凳龙”的运动曲线各部分保持相切。因为圆弧调头曲线的起点、终点以及两点上的切线方向均确定，所以**给定两段圆弧的半径比，即可唯一确定调头曲线**。

再通过勾股定理列出方程分别求解两段圆弧的圆心角，即可得到圆弧调头曲线的长度。同时根据计算结果，调头曲线的长度**与两段圆弧半径的比值无关**。因此我们直接否认了缩短调头曲线的可行性，即无法通过调整圆弧使得总调头曲线长度减小。

接下来以问题四题设所示路径进行建模，计算要求时间范围内各把手的位置及速度。以龙头前把手走过路径长度为索引，将盘入螺线、调头曲线、盘出螺线串联为一整条曲线，简化模型。至此，我们可以根据问题一的算法计算题目所需时间段内各把手的位置以及速度。

2.5 问题五分析

由于本问题继承问题四的运动路径，因此只需观察前问表格，即可分析得到运动过程中速度最大值出现的时刻与位置。观察结果表明，速度最大值只会出现在龙头进入盘出螺线时的第 1 节龙身。于是我们从盘出螺线的起点开始，逆时针方向取一小段盘出曲线为研究对象，研究龙头运动到该段曲线时第 1 节龙身的速度，并采用类似问题三的算法，对龙头的速度多次进行二分，从而得到优化后的答案。

三、 模型假设

- **板凳物理性质假设：**将板凳视为刚体，不考虑相邻板凳运动中的相对位移与弹性。
- **运动路径高度假设：**忽略板凳厚度以及把手连接处板凳上下重叠带来的影响，认为所有板凳的上表面在同一平面上。
- **调头路径假设：**假设“板凳龙”进入给定调头区域后立即进入调头曲线，且调头曲线始终由两段相切圆弧构成。

四、 符号说明

符号	意义
t	从出发起用时
v	龙头前把手速度
L	龙头的板长
l	龙身和龙尾的板长
w	板凳的板宽
s	把手孔中心到最近板头的距离
d	螺距
Ω	盘入螺线
Ω'	盘出螺线
θ_l	螺线段上点极角下界
θ_r	螺线段上点极角上界
$c(\theta_l, \theta_r)$	极角范围 $[\theta_l, \theta_r]$ 的螺线段的长度
$i = 1, 2, ...221$	第 i 个龙身的前把手标号
$i = 0$	龙头前把手中心点标号
$i = 222$	龙尾前把手中心点标号
$i = 223$	龙尾后把手中心点标号
$\theta_{i,t}$	t 时刻标号为 i 的把手极角
ε	计算时允许的数值误差

五、 模型的建立与求解

5.1 问题一模型的建立及求解

5.1.1 “板凳龙”运动模型建立

在本问中我们只关心把手的位置与速度。我们用把手中心点的位置来表示把手的位置。因为视板凳为刚体，所以同一个板凳的两个把手速度与板凳形状无关。又由于前一

个板凳的后把手就是后一个板凳的前把手，所以相邻板凳的前把手之间的唯一关联是同一个板凳两个把手中心的连线。由此可见，在运动模型建立的过程中，我们只需关注把手中心点的位置以及相邻把手中心点的连线。这些点以及连线得到了一条以把手中心为端点的折线段。该折线模型满足：

1. 折线上每一个端点都在给定的等距螺线上。于是我们可以由极角 $\theta_{i,t}$ 唯一确定某时刻某端点的位置。

2. 折线上每一条线段的长度都对应板凳前后把手中心的距离。

5.1.2 坐标系的建立

为了更好地描述，我们在题目给出的直角坐标系的基础上建立极坐标系 Ox, 其中 O 是极坐标系的极点，也是原直角坐标系的原点 O。以原直角坐标系的 x 轴正方向为极轴方向，以 $1m$ 为单位长度，逆时针方向为角度正方向，建立极坐标系。由有序数对 (ρ, θ) 表示极坐标系中的一点 M，其中 ρ 为极距，θ 为极角。由极坐标系定义可知 M 在原直角坐标系中的坐标为 $(\rho\cos\theta, \rho\sin\theta)$。

5.1.3 极坐标系下的等距螺线分析

对于顺时针盘入的等距螺线，其在极坐标系下的曲线方程为：

$$\rho = a\theta + b \ (a > 0)$$

又由于曲线过原点以及螺距已知为 $d = 0.55$（单位：m），于是可以得到：

$$\begin{cases} 0 = a \times 0 + b \\ d = a \times 2\pi + b \end{cases}$$

解得:

$$a = \frac{0.55}{2\pi},\ b = 0$$

也就得到盘入螺线的方程

$$\Omega:\ \rho = \rho(\theta) = a\theta = \frac{0.55}{2\pi}\theta$$

再考虑计算已知极角范围为 $[\theta_l, \theta_r]$ 的螺线段的长度。首先我们考虑极角范围为 $[0, \theta_r]$ 的螺线，它的长度为：

$$\begin{aligned} c(0, \theta_r) &= \int_0^{\theta_r} \sqrt{\rho(\theta)^2 + \rho'^2(\theta)}\mathrm{d}\theta \\ &= \int_0^{\theta_r} \sqrt{a^2\theta^2 + a^2}\mathrm{d}\theta \\ &= \frac{a}{2}(\theta_r\sqrt{\theta_r^2 + 1} + \ln(\theta_r + \sqrt{\theta_r^2 + 1})) \end{aligned}$$

类似地，我们也可以求出极角范围为 $[0, \theta_l]$ 的螺线长度为：

$$c(0, \theta_l) = \frac{a}{2}(\theta_l\sqrt{\theta_l^2 + 1} + \ln(\theta_l + \sqrt{\theta_l^2 + 1}))$$

于是我们可以求得已知极角范围的螺线段的长度

$$c(\theta_l, \theta_r) = c(0, \theta_r) - c(0, \theta_l)$$

5.1.4 龙头前把手运动分析

因为龙头前把手速度 $v = 1m/s$ 不变，所以只需求 t 时刻的龙头前把手极角 $\theta_{0,t}$，就可以得到对应的位置。由把手沿螺线运动知，其位移为一段螺线段的长度：

$$vt = c(\theta_{0,t}, \theta_{0,0})$$

其中 $\theta_{0,0}$ 为已知：

$$\theta_{0,0} = 32\pi$$

初始时刻“板凳龙”的位置如图 1 所示：

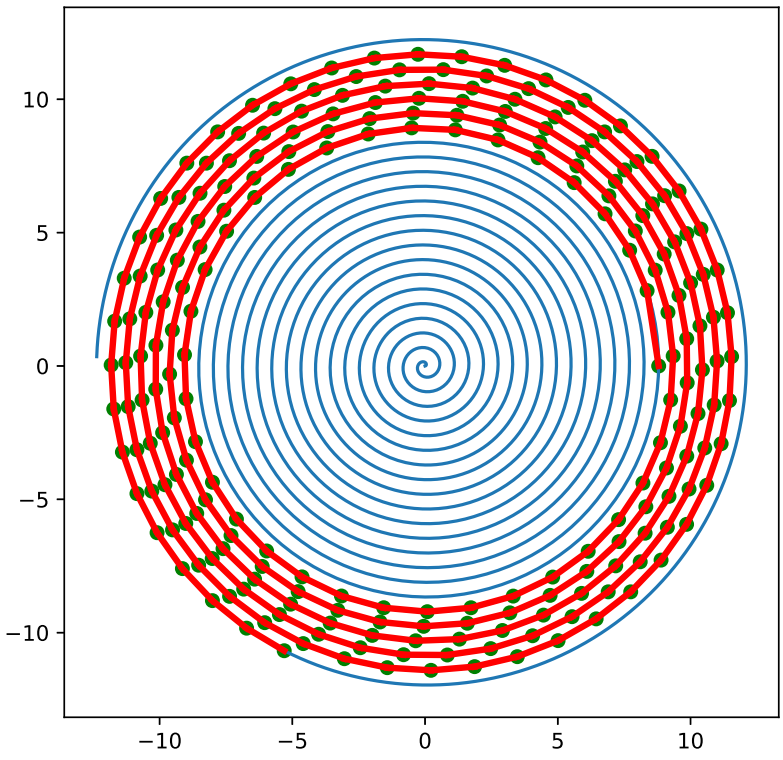

图 **1**　初始时刻“板凳龙”的位置

在已知 t 时由二分法即可求得 $\theta_{0,t}$ 的值：

Step1：转化为便于二分求解的问题形式：将问题转化为求方程 $f(\theta)=0$ 的解。这个解即为我们所求的 $\theta_{0,t}$。结合 5.1.3 节的分析，我们可得待研究方程为

$$f(\theta)=vt-c(\theta,\theta_{0,0})=0\ (t\leq 300)$$

Step2：验证 $f(\theta)$ 符合使用二分求解的条件：因为 $f(\theta)$ 是简单初等函数，所以在定义域上连续。又由式子几何意义知 $c(\theta,\theta_{0,0})$ 单调，结合 vt 是常数，得到 $f(\theta)$ 在定义域上单调。再注意到 $f(0)=vt-c(0,\theta_{0,0})\leq 300-442.5<0$，$f(32\pi)=vt>0$。至此，我们确认函数满足二分求解且解唯一的条件：

1. $f(\theta)$ 在区间 $[0,32\pi]$ 上连续且单调。

2. $f(0)f(32\pi)<0$.

Step3：确定二分中止条件：题目要求数值计算结果保留六位小数，为了保证达到题目允许的数值误差，我们设置计算时允许的数值误差 ε 满足

$$\varepsilon\leq 1\times 10^{-8}$$

为了让二分满足这个条件，我们以二分区间长度小于等于 ε 为中止条件。

Step4：按照下列伪代码求解：

Require: 方程 $f(\theta)=0$ 的解在 $[a,b]$ 范围内且范围内 $f(\theta)$ 单调递增，误差 ε 内求解
  while $a-b>\varepsilon$ **do**
    $mid\leftarrow\frac{a+b}{2}$
    if $f(mid)>0$ **then**
      $b\leftarrow mid$
    else
      $a\leftarrow mid$
    end if
  end while
  return mid

最后返回的结果即为方程 $f(\theta)=0$ 的解 $\theta_{0,t}$，由此确定了龙头前把手的位置。

5.1.5 后续把手位置的递推求解

由上一小节的推导，我们求得 $\theta_{0,t}$。在此基础上，若我们求得 $\theta_{i,t}$ 与 $\theta_{i+1,t}$ 之间的递推关系式，我们即可算出各把手在不同时间的位置。对此，考虑把手都在螺线上且把手

间距离已知且固定，所以我们可以以一个把手为圆心、把手间距离为半径构造辅助圆并与螺线方程联立求解另一个把手的位置。给出方程组如下：

$$\begin{cases} \rho_i = a\theta_i \\ x_i = \rho_i \cos\theta_i \\ y_i = \rho_i \sin\theta_i \\ \rho_{i+1} = a\theta_{i+1} \\ x_{i+1} = \rho_{i+1} \cos\theta_{i+1} \\ y_{i+1} = \rho_{i+1} \sin\theta_{i+1} \\ (x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 = r^2 \\ r = \begin{cases} L, i = 0 \\ l, i \neq 0 \end{cases} \end{cases}$$

其中 $i = 0, 1, ..., 222$. 解方程组即可得到答案。又因为 $r > 2d$，方程组会解出不止一组解。考虑到一个板凳的两个把手不可能绕螺线超过一圈，所以取 θ_{i+1} 的多个根里满足

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

的那个根作为答案。龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置如表 1 所示。

5.1.6 各把手速度求解

为了求各把手速度，我们使用微元法求解。给定某把手在某时刻的位置，我们选择前后各运动一段足够短的时间 Δt 形成的曲线，在精度范围内我们可以将这一小段的平均速度视作此时的瞬时速度：

$$v_{i,t} = \frac{c(\theta_{i,t+\Delta t}, \theta_{i,t-\Delta t})}{2\Delta t}$$

解得龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手在不同时刻的速度如表 2 所示。

5.2 问题二模型的建立及求解

5.2.1 板凳矩形边界碰撞模型

为了判断给定时刻板凳是否碰撞，我们需要解决板凳边沿矩形的重叠问题。对此，可以转化为矩形边界线段相交问题。我们需要快速排斥实验与跨立实验来解决这个问题。

表 **1**　把手在不同时刻的位置

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.800000	5.799208	-4.084890	-2.963605	2.594489	4.420277
龙头 y (m)	0.000000	-5.771093	-6.304477	6.094782	-5.356745	2.320423
第 1 节龙身 x (m)	8.363824	7.456757	-1.445476	-5.237115	4.821218	2.459494
第 1 节龙身 y (m)	2.826544	-3.440400	-7.405882	4.359630	-3.561953	4.402473
第 51 节龙身 x (m)	-9.518732	-8.686316	-5.543147	2.890460	5.980008	-6.301345
第 51 节龙身 y (m)	1.341137	2.540110	6.377948	7.249287	-3.827763	0.465835
第 101 节龙身 x (m)	2.913982	5.687113	5.361935	1.898789	-4.917376	-6.237719
第 101 节龙身 y (m)	-9.918311	-8.001385	-7.557641	-8.471615	-6.379870	3.936014
第 151 节龙身 x (m)	10.861727	6.682314	2.388762	1.005160	2.965384	7.040744
第 151 节龙身 y (m)	1.828752	8.134542	9.727410	9.424750	8.399718	4.393007
第 201 节龙身 x (m)	4.555105	-6.619660	-10.627210	-9.287723	-7.457156	-7.458667
第 201 节龙身 y (m)	10.725117	9.025573	1.359853	-4.246667	-6.180720	-5.263377
龙尾（后）x (m)	-5.305447	7.364554	10.974349	7.383901	3.241058	1.785041
龙尾（后）y (m)	-10.676583	-8.797995	0.843467	7.492365	9.469334	9.301162

表 **2**　把手在不同时刻的速度

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999971	0.999961	0.999945	0.999917	0.999859	0.999709
第 51 节龙身 (m/s)	0.999742	0.999662	0.999538	0.999331	0.998941	0.998065
第 101 节龙身 (m/s)	0.999575	0.999453	0.999269	0.998971	0.998435	0.997302
第 151 节龙身 (m/s)	0.999448	0.999299	0.999078	0.998727	0.998115	0.996861
第 201 节龙身 (m/s)	0.999348	0.999180	0.998935	0.998551	0.997893	0.996574
龙尾（后）(m/s)	0.999311	0.999136	0.998883	0.998489	0.997816	0.996478

Step1：求得给定时间给定板凳矩形的顶点：先由问题一的方法求得研究板凳的把手中心，再根据板凳形状不变，由相对位置关系在直角坐标系中求出给定板凳矩形的顶点，也就分别得到了两个矩形各自的四条边。接下来只需两两判断分属两个矩形的边是否相交。

Step2：由快速排斥实验初步检验线段是否相交：以线段为对角线，分别构造一个

各边平行于直角坐标系坐标轴的矩形，若构造出的两个矩形没有公共区域，则线段一定没有相交。

Step3：由跨立实验确定线段是否相交：在快速排斥实验的基础上，我们可以发现，若一条线段的两个端点分别在另一条线段的两侧，则两条线段相交。为了知道点在线段的哪一侧，我们可以利用向量叉乘的性质判断。要判断点 R,T 是否在线段 PQ 的同一侧，我们只需计算 $\overrightarrow{RP}\times\overrightarrow{PQ}$ 和 $\overrightarrow{TP}\times\overrightarrow{PQ}$，若二者符号不相同则说明两线段相交，否则没有相交。

5.2.2 二分查找初次碰撞时刻

使用伪代码表示这一过程如下：

Require: 最小化板凳发生碰撞的时刻 t
  $a \leftarrow 0$
  $b \leftarrow \frac{c(0,\theta_{0,0})}{v}$
  while $a-b>\varepsilon$ **do**
    $mid \leftarrow \frac{a+b}{2}$
    if mid 时刻板凳有碰撞 **then**
      $b \leftarrow mid$
    else
      $a \leftarrow mid$
    end if
  end while
  return mid

最终返回结果即为最初碰撞时刻 $t_{\min}=412.473838$s。再由第一问算法解出该时刻的把手位置以及速度，结果如表 3 所示。碰撞时前若干节板凳所处位置如图 2 所示。

5.2.3 正确性检验

由于模型在实际情况中可能存在随机误差，在满足题目给定条件的情况下，根据已解出的最初碰撞时刻 t_{min}，我们在 $[t_{min}-10,t_{min})$ 和 $[t_{min}-1,t_{min})$ 两个区间内分别随机生成了 100 个时刻，并检验模型的合理性。结果显示，在所有随机生成的时刻前，“舞龙队”板凳之间未发生碰撞，说明我们没有漏掉更小的解，同时模型的正确性得到验证。代码中我们将模型检验代码和模型代码放在同一个文件里，便于运行。

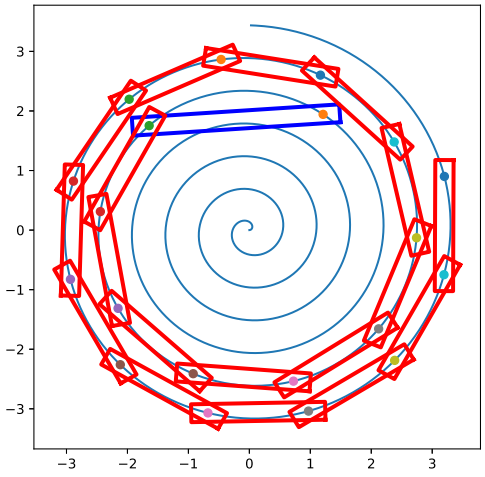

图 **2**　碰撞时前若干节板凳所处位置

表 **3**　把手在碰撞时的位置以及速度

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头	1.209931	1.942784	1.000000
第 1 节龙身	-1.643792	1.753399	0.991551
第 51 节龙身	1.281201	4.326588	0.976858
第 101 节龙身	-0.536246	-5.880138	0.974550
第 151 节龙身	0.968840	-6.957479	0.973608
第 201 节龙身	-7.893161	-1.230764	0.973096
龙尾（后）	0.956217	8.322736	0.972938

5.3 问题三模型的建立及求解

5.3.1 对连续的运动过程采样

问题二中的碰撞检验算法针对的是某一时刻的问题，而本问中所需要考察的运动过程则是连续时间下的问题。于是我们对运动过程进行**等时间间距采样**得到一系列时刻的碰撞检测问题，这个问题可以利用问题二中的算法求解。若经过检测，这一系列时刻均未发生碰撞，且设定的时间间隔足够小时，我们就可以认为在连续运动过程中没有发生碰撞。

5.3.2 二分查找符合题意的最小螺距

使用伪代码表示这一过程如下：

Require: 最小化螺距 d，使得螺距为 d 的行进路线中无碰撞
  while $a - b > \varepsilon$ **do**
    $mid \leftarrow \frac{a+b}{2}$
    if 螺距为 mid 的行进路线中无碰撞 **then**
      $b \leftarrow mid$
    else
      $a \leftarrow mid$
    end if
  end while
  return mid

返回值即为所求的最小螺距 d_{min}。

5.3.3 参数优化

此算法存在两个参数，分别为采样时间间隔和起始二分区间。对于采样时间间隔，当间隔过大时，更可能会错过碰撞时的状态导致误判，而间隔过小会导致计算量过大。对于起始二分区间，当区间过大时会导致计算量过大，而区间过小可能会错过答案。所以在计算过程中，我们需要依据计算结果以及计算效率，多次调整参数并运行程序，在保证计算精度的同时缩短计算时间。最终得到符合题目要求的最小螺距为

$$d_{min} = 0.450337m$$

5.3.4 正确性检验

与问题二进行的检验类似，在满足题目给定条件的情况下，根据模型已解出的最小螺距 d_{min}，我们在区间 $[d_{min} - 0.01, d_{min})$ 内随机做 100 次扰动测试，并检验其合理性。结果显示，在所有生成的螺距里，龙头前把手均不能够沿着相应的螺线盘入到调头空间的边界，说明我们求出的解是最优的，同时也验证了模型的正确性。由于代码中设定了时间阈值，我们在时间阈值内只计算龙头的前把手和后把手的位置，以优化运行时间。要进行更大的扰动，需要调小时间阈值来获取相应结果。

5.4 问题四模型的建立与求解

5.4.1 缩短圆弧调头路径的可行性判断与证明

1. 两段圆弧半径比确定时调头路径唯一

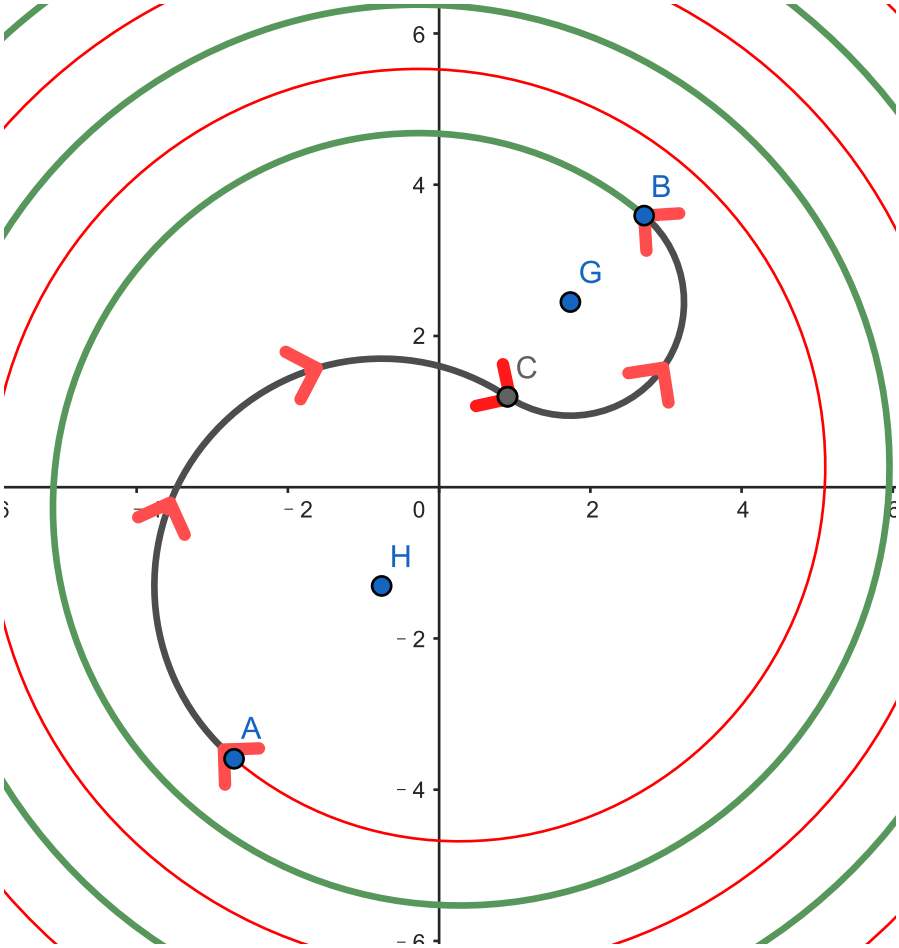

图 **3**　调头路线示意图

由示意图我们可以看出，因为要求圆弧与螺线相切，所以圆弧的圆心被限制在了圆弧与螺线连接点处的法线方向，两段圆弧所在的圆的位置只由半径的大小确定。又由于两段圆弧的半径比已经给定，再加上要求两段圆弧相切的约束条件，圆心距离被唯一确定，圆的位置也被唯一确定。最终我们唯一确定了在给定条件下，弧 ACB 即为“板凳龙”的调头曲线，如图 3 所示。

2. 两段圆弧任意半径比下调头路径长度恒定

图 4 中 A, B, C 分别是大圆弧与盘入螺线、小圆弧与盘出螺线以及大圆弧和小圆弧之间的切点，G, H 分别是小圆弧和大圆弧的圆心，延长 BG 交过大圆弧在 A 点处的切线于 D。设 $AD = a$，$BD = b$，$GC = r$，大圆弧与小圆弧的半径之比为 k，即 $CH = kr$，$\angle DGC = \alpha$。

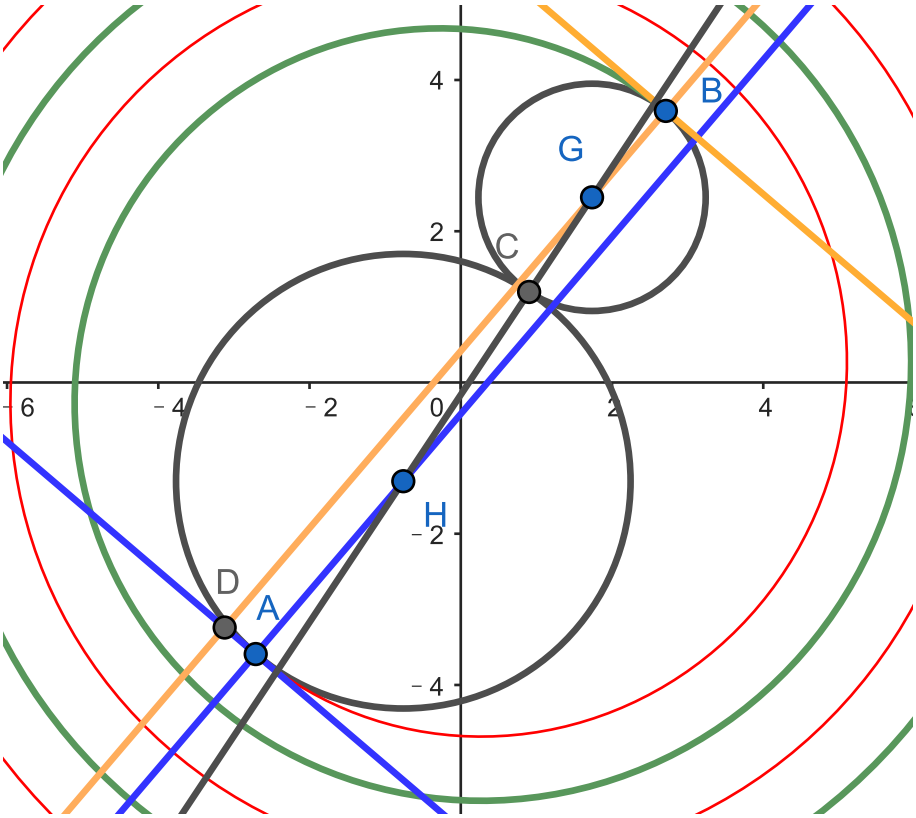

图 **4**　调头路线长度计算

由切线性质可知

$$AD \perp BD$$

由于盘入螺线和盘出螺线关于原点中心对称，因此两条螺线和圆弧的公共切线相互平行。由平行四边形的性质与勾股定理：

$$a^2 + (b - r - kr)^2 = (r + kr)^2$$

由三角函数性质可知

$$\sin\alpha = \frac{a}{(k+1)r}$$

由弧长公式知

$$C_{AB} = (\pi - \alpha)(k+1)r$$

联立解得

$$C_{AB} = (\pi - \arcsin\frac{2ab}{a^2+b^2})\frac{a^2+b^2}{2b}$$

这是与半径比 k 无关的定值。所以不能通过调整圆弧使得调头曲线变短。

5.4.2 对运动过程条件与问题的分析

我们已经证明任何用两段相切圆弧构造的调头曲线长度均为定值，因此在接下来的计算中我们选取问题四最初给定的调头曲线。

在本问题中，我们仍然需要关心的只有把手的位置和速度，因此在求解时只需沿用问题一建立的“板凳龙”运动模型即可。与第一问不同的是，待研究方程 $f(\theta)=0$ 中弧长项因为路径发生了从螺线到组合曲线的变化，龙头前把手位置的计算方式也不能只是简单地从极角得到。因此，我们需要建立一种新的算法计算龙头前把手在时刻 t 的位置。该算法的具体实现方式如下：

1. 确定 $t=0$ 时刻的位置

题目要求以调头开始时间为零时刻，所以此时位置基准点为调头开始时刻龙头前把手所处的位置，即图 4 中大圆弧与盘入螺线的切点 A。

2. 确定初始的参考基准点

由于题目要求从 $t=-100$s 时刻开始计算把手的位置与速度，所以我们选择该时刻龙头前把手所处的位置作为后续计算的参考基准点。

3. 根据龙头位置距离参考基准点的弧长计算位置

设龙头从基准点开始沿曲线向圆弧方向运动 c'm. 由于运动路程与位置能够形成单射，所以给定任意的运动路程，我们都能在运动曲线上唯一确定一个位置。现在已知龙头前把手的运动速度为 1m/s，设大圆弧和小圆弧的半径分别为 r_1 和 r_2，满足 $r_1=2r_2$，每段圆弧的圆心角相等，均为 β. 于是：

(1) 当 $c'\leq 100$ 时，龙头前把手位于盘入螺线上。此时直接在盘入螺线上利用二分算法寻找位置。

(2) 当 $c'>100$ 且 $c'\leq 100+r_1\beta$ 时，龙头前把手位于大圆弧上。此时计算从大圆弧起点 A 开始运动 $c-100$m 的位置即可。

(3) 当 $c'>100+r_1\beta$ 且 $c'\leq 100+r_1\beta+r_2\beta$ 时，龙头前把手位于小圆弧上。此时计算从小圆弧起点 C 开始运动 $c-100-r_1\beta$m 的位置即可。

(4) 当 $c'>100+r_1\beta+r_2\beta$ 时，龙头前把手位于盘出螺线上。此时龙头前把手沿盘出螺线向前运动了 $c'-(100+r_1\beta+r_2\beta)$m。根据中心对称的性质，我们可以视作龙头前把手从盘入螺线终点开始倒退 $c'-(100+r_1\beta+r_2\beta)$m 的路程，计算出这个位置并给出关于原点中心对称的位置即可。

在得到新的计算方法后，后续二分答案求解方程的过程与第一问完全相同，计算得到特定时刻下，把手的位置和速度结果分别见下表 4 和表 5 所示。

5.5 问题五模型的建立与求解

5.5.1 最大速度形成分析

在本问中，“板凳龙”的路径始终与问题四一致。因此运动到给定位置时，各把手间相对速度大小关系不会随着龙头速度改变而改变。所以，只需观察问题四条件下把手最大速度出现的位置，就可以推广到任意龙头速度下的情形。从问题四的结果中我们观

表 **4**　把手在调头过程前后的位置

	-100 s	-50 s	0 s	50 s	100 s
龙头 x (m)	7.778034	6.608301	-2.711856	1.332696	-3.157229
龙头 y (m)	3.717164	1.898865	-3.591078	6.175324	7.548511
第 1 节龙身 x (m)	6.209273	5.366911	-0.063534	3.862265	-0.346890
第 1 节龙身 y (m)	6.108521	4.475403	-4.670888	4.840828	8.079166
第 51 节龙身 x (m)	-10.608038	-3.629945	2.459962	-1.671385	2.095033
第 51 节龙身 y (m)	2.831491	-8.963800	-7.778145	-6.076713	4.033787
第 101 节龙身 x (m)	-11.922761	10.125787	3.008493	-7.591816	-7.288774
第 101 节龙身 y (m)	-4.802378	-5.972246	10.108539	5.175487	2.063875
第 151 节龙身 x (m)	-14.351032	12.974784	-7.002789	-4.605164	9.462514
第 151 节龙身 y (m)	-1.980994	-3.810357	10.337482	-10.386989	-3.540357
第 201 节龙身 x (m)	-11.952943	10.522509	-6.872842	0.336953	8.524374
第 201 节龙身 y (m)	10.566998	-10.807425	12.382609	-13.177610	8.606933
龙尾（后）x (m)	-1.011058	0.189809	-1.933627	5.859094	-10.980157
龙尾（后）y (m)	-16.527573	15.720588	-14.713128	12.612894	-6.770006

表 **5**　把手在调头过程前后的速度

	-100 s	-50 s	0 s	50 s	100 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999904	0.999762	0.998595	1.000363	1.000124
第 51 节龙身 (m/s)	0.999346	0.998641	0.995044	0.949949	1.003966
第 101 节龙身 (m/s)	0.999091	0.998248	0.994358	0.948496	1.096234
第 151 节龙身 (m/s)	0.998944	0.998048	0.994066	0.948052	1.095277
第 201 节龙身 (m/s)	0.998849	0.997925	0.993905	0.947837	1.094904
龙尾（后）(m/s)	0.998818	0.997886	0.993855	0.947774	1.094803

察到：速度最大值只会出现在龙头从调头路线进入盘出螺线时的第 1 节龙身，如图 5 所示。

5.5.2 二分求解龙头最大速度

使用伪代码表示这一过程如下：

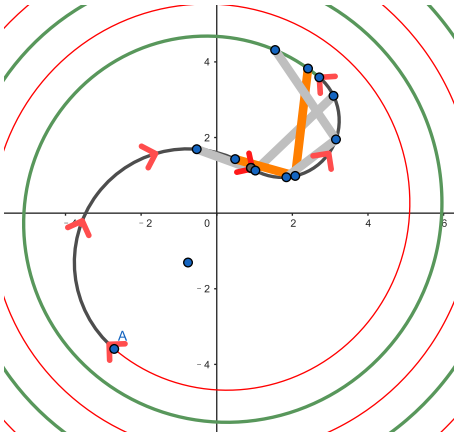

图 **5**　速度最大值出现位置示意图 **(**橙色折线处**)**

Require: 最大化龙头速度 v，并且在龙头从调头路线进入盘出螺线时的第 1 节龙身速度不超过 2m/s

while $a - b > \varepsilon$ **do**

$mid \leftarrow \frac{a+b}{2}$

if 第 1 节龙身速度不超过 2m/s **then**

$b \leftarrow mid$

else

$a \leftarrow mid$

end if

end while

return mid

其中为了判断第 1 节龙身的是否超过 2m/s，我们对进入盘出螺线时到之后 3m 的运动过程进行采样。采样间隔是 0.001s。判断每个采样点中第 1 节龙身的速度是否超过 2m/s。最终返回结果即为所求最大龙头速度

$$v_{max} = 1.283843 m/s.$$

5.5.3 正确性检验

与问题二、问题三进行的检验类似，在满足题目给定条件的情况下，模型已求出的龙头前把手最大速度为 v_{max}. 我们在区间 $(v_{max}, v_{max} + 0.01]$ 做 100 次扰动测试，并检验模型的合理性。结果显示，在所有随机产生的龙头里，至少存在一个把手的速度超过了 2 m/s，说明我们求出的最大速度是正确的，显示出模型的稳定性。

六、 模型的优缺点及改进

6.1 模型的优点

- 根据问题需要，采用不同的坐标系灵活分析，简化了模型运算，提高了建模分析的综合性以及全面性。
- 本题建模主要使用二分法，在解决问题的前提下，尽量减小了模型复杂度。
- 部分需要检验的模型通过了随机扰动检验，保证了模型正确性。

6.2 模型的缺点

- 使用采样的方式将连续运动问题转化为离散问题，在极端情况下可能会造成一定的误差，此时的误差会随着递推累积。
- 在超参数优化中，由于二分法的特性，我们可能需要多次调整参数来尽可能避免陷入局部最优解。

6.3 模型的改进

- 由于问题的特性，可以使用随机化算法以及智能算法辅助模型构建。
- 可能能找到一种算法找出问题的解析解，提高运算效率。

七、 参考文献

[1] 陆先森不怕鬼. 直角坐标与极坐标互相转换（Python&C++ 实现）[EB/OL].(2022-12-15)[2024-09-08].https://blog.csdn.net/weixin_44729155/article/details/128299755.

[2] 吉因克丝.Python 判断线段是否相交,快速排斥 + 叉乘 [EB/OL].(2022-04-02)[2024-09-08].https://blog.csdn.net/m0_37660632/article/details/123925503.

[3] 星夜孤帆.python 一个点绕另一个点旋转后的坐标 [EB/OL].(2018-11-19)[2024-09-08].https://blog.csdn.net/qq_38826019/article/details/84233397.

附录 A: 支撑文件目录

python 代码：

1.py

1speedfinal.py

2.py

3.py

32.py

33.py

34.py

3check.py

4.py

5.py

5check.py

plot.py

plot2.py

Excel 文件：

result1.xlsx

result2.xlsx

result4.xlsx

附录 B：题目代码

Listing 1: **1.py**

```python
import time
import numpy as np
import matplotlib.pyplot as plt
pi = 3.1415926535
# class point:
#     x = 0
#     y = 0
#     theta = 0
#     rho = 0

def Rectangular_to_Polar(x, y):  # 直角坐标转极坐标，输出的thata为角度值
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):  # 极坐标转直角坐标，输入的thata需为角度值
    # theta = theta * (np.pi / 180)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid):
    a = 0.55 / 2 / pi
    return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len):
    # rho, theta = Rectangular_to_Polar(x, y)
    # print(rho, theta)
    theta = otheta
    l = theta - 2 * pi
    r = theta
    a = 0.55 / 2 / pi
    for _ in  range(30):
        mid = (l + r) / 2
        if(arc_len(theta) - arc_len(mid) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
```

```
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = 0.55 / 2 / pi
    for _ in  range(30):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta


# print(get_nxt_point(x,y,theta),sep=",")

# plt.plot(x,y,marker=".")
# print(x,y,sep=",")
# 448 * 301

data = np.zeros((448, 301))
x = 0.55 * 16
y = 0
theta = 32 * pi
org_x = x
org_y = y
org_theta = theta
for t in  range(301):
    if(t != 0):
        x = org_x
        y = org_y
        theta = org_theta
        x,y,theta = get_nxt_point(x,y,theta,1)
        org_x = x
        org_y = y
        org_theta = theta
```

```
    data[0][t] = x
    data[1][t] = y
    # plt.plot(x,y,marker=".")
    x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55)
    data[2][t] = x
    data[3][t] = y
    # plt.plot(x,y,marker=".")
    for i in  range(222):
        x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55)
        data[(i + 2) * 2][t] = x
        data[(i + 2) * 2 + 1][t] = y
        # plt.plot(x,y,marker=".")

    # print(x,y,sep=",")
    # plt.plot(x,y,marker=".")
# for i in range(224):
#     print(f"{data[i * 2][0]}, {data[i * 2 + 1][0]}")
import pandas as pd
df = pd.DataFrame(data)
df.to_excel("output.xlsx",index=False)
print("Done.")
```

Listing 2: **1speedfinal.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
pi = 3.1415926535
# class point:
#     x = 0
#     y = 0
#     theta = 0
#     rho = 0

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    # theta = theta * (np.pi / 180)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid):
    a = 0.55 / 2 / pi
```

```
24      return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
25
26  def point_distance(x,y,a,b):
27      return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))
28
29  def get_nxt_point(x,y,otheta, len):
30      # rho, theta = Rectangular_to_Polar(x, y)
31      # print(rho, theta)
32      theta = otheta
33      l = theta - 2 * pi
34      r = theta
35      a = 0.55 / 2 / pi
36      for _ in  range(50):
37          mid = (l + r) / 2
38          if(arc_len(theta) - arc_len(mid) <  len):
39              r = mid
40          else:
41              l = mid
42      ntheta = l
43      nrho = a * ntheta
44      # print(nrho, ntheta)
45      nx, ny = Polar_to_Rectangular(nrho, ntheta)
46      # print(nx,ny)
47      return nx, ny, ntheta
48
49  def get_prev_point(x,y,otheta,lenn):
50      # rho, theta = Rectangular_to_Polar(x, y)
51      # print(rho, theta)
52      theta = otheta
53      l = theta
54      r = theta + 2 * pi
55      a = 0.55 / 2 / pi
56      for _ in  range(50):
57          mid = (l + r) / 2
58          if(arc_len(mid) - arc_len(theta) < lenn):
59              l = mid
60          else:
61              r = mid
62      ntheta = l
63      nrho = a * ntheta
64      # print(nrho, ntheta)
65      nx, ny = Polar_to_Rectangular(nrho, ntheta)
66      # print(nx,ny)
67      return nx, ny, ntheta
68
69  def get_prev_board(x,y,otheta, len):
70      theta = otheta
```

```
    l = theta
    r = theta + pi
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

# v = 1m/s
# t = t
# s = vt

# print(get_nxt_point(x,y,theta),sep=",")

# plt.plot(x,y,marker=".")
# print(x,y,sep=",")
# 448 * 301
T = 0.00001

data = np.zeros((448, 301))
data_nxt = np.zeros((448, 301))
data_prev = np.zeros((448, 301))
data_theta_prev = np.zeros((224, 301))
data_theta_nxt = np.zeros((224, 301))

speed = np.zeros((224, 301))

x = 0.55 * 16
y = 0
theta = 32 * pi
org_x = x
org_y = y
org_theta = theta
for t in  range(301):
    if(t != 0):
        x = org_x
        y = org_y
        theta = org_theta
```

```
        x,y,theta = get_nxt_point(x,y,theta,1)
        org_x = x
        org_y = y
        org_theta = theta
    data[0][t] = x
    data[1][t] = y
    data_prev[0][t], data_prev[1][t], data_theta_prev[0][t] = get_prev_point(x,y,theta,T)
    data_nxt[0][t], data_nxt[1][t], data_theta_nxt[0][t] = get_nxt_point(x,y,theta,T)
    # plt.plot(x,y,marker=".")
    x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55)
    data[2][t] = x
    data[3][t] = y
    data_prev[2][t], data_prev[3][t], data_theta_prev[1][t] = get_prev_board(data_prev[0][t],
        data_prev[1][t], data_theta_prev[0][t], 3.41 - 0.55)
    data_nxt[2][t], data_nxt[3][t], data_theta_nxt[1][t] = get_prev_board(data_nxt[0][t],
        data_nxt[1][t], data_theta_nxt[0][t], 3.41 - 0.55)
    # plt.plot(x,y,marker=".")
    for i in  range(222):
        x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55)
        data[(i + 2) * 2][t] = x
        data[(i + 2) * 2 + 1][t] = y
        data_prev[(i + 2) * 2][t], data_prev[(i + 2) * 2 + 1][t], data_theta_prev[i + 2][t] =
            get_prev_board(data_prev[(i + 2) * 2 - 2][t], data_prev[(i + 2) * 2 - 1][t],
            data_theta_prev[i+1][t], 2.2 - 0.55)
        data_nxt[(i + 2) * 2][t], data_nxt[(i + 2) * 2 + 1][t], data_theta_nxt[i + 2][t] =
            get_prev_board(data_nxt[(i + 2) * 2 - 2][t], data_nxt[(i + 2) * 2 - 1][t],
            data_theta_prev[i+1][t], 2.2 - 0.55)

        # data_prev[(i + 2) * 2][t], data_prev[(i + 2) * 2 + 1][t] = get_prev_point(x,y,theta,T)
        # data_nxt[(i + 2) * 2][t], data_nxt[(i + 2) * 2 + 1][t] = get_nxt_point(x,y,theta,T)

        # plt.plot(x,y,marker=".")

for t in  range(301):
    for i in  range(224):
        # print("prev",data_prev[(i*2)][t],data_prev[i*2+1][t])
        # print("just",data[(i*2)][t],data[i*2+1][t])
        # print("nxt",data_nxt[(i*2)][t],data_nxt[i*2+1][t])

        # print(arc_len(data_theta_prev[i][t]),arc_len(data_theta_nxt[i][t]))

        speed[i][t] = (arc_len(data_theta_prev[i][t]) - arc_len(data_theta_nxt[i][t])) / T / 2

    # print(x,y,sep=",")
    # plt.plot(x,y,marker=".")
# for i in range(224):
#     print(f"{data[i * 2][0]}, {data[i * 2 + 1][0]}")
```

```
159 import pandas as pd
160 df = pd.DataFrame(speed)
161 df.to_excel("output_speed.xlsx",index=False)
162 print("Done.")
```

Listing 3: **2.py**

```
1  import time
2  import numpy as np
3  import matplotlib.pyplot as plt
4  pi = 3.1415926535
5
6  class point:
7      x = 0
8      y = 0
9      def __init__(self,x,y):
10         self.x = x
11         self.y = y
12
13 class rect:
14     p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]
15
16     def __init__(self,a,b,c,d,e,f,g,h):
17         self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]
18
19     def print_point(self):
20         print("Rect{")
21         for i in  range(4):
22             print(f"({self.p[i].x},{self.p[i].y})")
23         print("}")
24
25 class vec:
26     x = 0
27     y = 0
28     def __init__(self,x,y):
29         self.x = x
30         self.y = y
31
32 def normalize(v):
33      len = np.sqrt(v.x*v.x+v.y*v.y)
34     return vec(v.x /  len, v.y /  len)
35
36 def Rectangular_to_Polar(x, y):
37     r = np.sqrt(np.square(x) + np.square(y))
38     theta = np.degrees(np.arctan(y / x))
39     return r, theta
40
```

```python
def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid):
    a = 0.55 / 2 / pi
    return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len):
    theta = otheta
    l = 0
    r = theta
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta) - arc_len(mid) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn):
    theta = otheta
    l = theta
    r = theta + 2 * pi
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid) - arc_len(theta) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
```

```
    r = theta + pi
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15

    point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
    point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15

    point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
    point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15

    return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)

def quick_judge(a,b,c,d): # 快速排斥，不相交返回 False，不能判断不相交返回 True
    if ( max(a.x,b.x) <  min(c.x,d.x) or
         max(c.x,d.x) <  min(a.x,b.x) or
         max(a.y,b.y) <  min(c.y,d.y) or
         max(c.y,d.y) <  min(a.y,b.y)) :
        return False
    else:
        return True

def xmult(a,b,c,d): # 叉乘
    vectorAx = b.x - a.x
    vectorAy = b.y - a.y
    vectorBx = d.x - c.x
    vectorBy = d.y - c.y
    return (vectorAx * vectorBy - vectorAy * vectorBx)
```

```
135 def cross(a,b,c,d):
136     if not quick_judge(a,b,c,d):
137         return False
138     xmult1 = xmult(c,d,c,a)
139     xmult2 = xmult(c,d,c,b)
140     xmult3 = xmult(a,b,a,c)
141     xmult4 = xmult(a,b,a,d)
142     if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
143         return True
144     else:
145         return False
146 
147 def collide_detect(s,t):
148     for i in  range(4):
149         for j in  range(4):
150             if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
151                 return True
152     return False
153 
154 def check(s):
155     data = np.zeros((448))
156     rect_array = []
157     x = 0.55 * 16
158     y = 0
159     theta = 32 * pi
160     x,y,theta = get_nxt_point(x,y,theta,s)
161     data[0] = x
162     data[1] = y
163     # print(s,x,y,theta)
164     x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55)
165     data[2] = x
166     data[3] = y
167     rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
168     # print(len(rect_array),rect_array[0].p[0].x)
169     # rect_array[0].print_point()
170     for i in  range(222):
171         x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55)
172         data[(i + 2) * 2] = x
173         data[(i + 2) * 2 + 1] = y
174         rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
175 
176     # for i in rect_array:
177     #     i.print_point()
178 
179     for i in  range( len(rect_array)):
180         for j in  range(i+2,  len(rect_array)):
181             if(collide_detect(rect_array[i],rect_array[j])):
```

```
                # print("true")
                return True
    # print("false")
    return False

L = 0
R = arc_len(32*pi)
# print(R)

for _ in  range(50):
    mid = (L + R) / 2
    if(check(mid)):
        R = mid
    else:
        L = mid

print(L)

import random
print("[L - 10, L) test:") # 稳定性测试 1
for _ in  range(1, 101):
    test_mid = L - 10 + random.random() * 10
    if check(test_mid):
        print(f"#{_} test = {test_mid}, collided, model failed.")
        exit(0)
    else:
        print(f"#{_} test = {test_mid}, pass")

print("[L - 1, L) test:") # 稳定性测试 2
for _ in  range(1, 101):
    test_mid = L - 1 + random.random() * 1
    if check(test_mid):
        print(f"#{_} test = {test_mid}, collided, model failed.")
        exit(0)
    else:
        print(f"#{_} test = {test_mid}, pass")

data = np.zeros((448))
data_nxt = np.zeros((448))
data_prev = np.zeros((448))
data_theta_prev = np.zeros((224))
data_theta_nxt = np.zeros((224))
speed = np.zeros((224))

rect_array = []

x = 0.55 * 16
```

```
y = 0
theta = 32 * pi
T = 0.00001

x,y,theta = get_nxt_point(x,y,theta,L)
data[0] = x
data[1] = y
data_prev[0], data_prev[1], data_theta_prev[0] = get_prev_point(x,y,theta,T)
data_nxt[0], data_nxt[1], data_theta_nxt[0] = get_nxt_point(x,y,theta,T)
x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55)
data[2] = x
data[3] = y
data_prev[2], data_prev[3], data_theta_prev[1] = get_prev_board(data_prev[0], data_prev[1],
    data_theta_prev[0], 3.41 - 0.55)
data_nxt[2], data_nxt[3], data_theta_nxt[1] = get_prev_board(data_nxt[0], data_nxt[1],
    data_theta_nxt[0], 3.41 - 0.55)
rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
for i in  range(222):
    x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55)
    data[(i + 2) * 2] = x
    data[(i + 2) * 2 + 1] = y
    data_prev[(i + 2) * 2], data_prev[(i + 2) * 2 + 1], data_theta_prev[i + 2] =
        get_prev_board(data_prev[(i + 2) * 2 - 2], data_prev[(i + 2) * 2 - 1],
        data_theta_prev[i+1], 2.2 - 0.55)
    data_nxt[(i + 2) * 2], data_nxt[(i + 2) * 2 + 1], data_theta_nxt[i + 2] =
        get_prev_board(data_nxt[(i + 2) * 2 - 2], data_nxt[(i + 2) * 2 - 1], data_theta_prev[i+1],
        2.2 - 0.55)
    rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))

rect_array[0].print_point()
rect_array[8].print_point()

for i in  range(224):
    speed[i] = (arc_len(data_theta_prev[i]) - arc_len(data_theta_nxt[i])) / T / 2

data_xlsx = np.zeros((224,3))

for i in  range(224):
    data_xlsx[i][0] = data[i * 2]
    data_xlsx[i][1] = data[i * 2 + 1]
    data_xlsx[i][2] = speed[i]

import pandas as pd
df = pd.DataFrame(data_xlsx)
df.to_excel("output2.xlsx",index=False)
print("Done.")
```

Listing 4: **3.py**

```python
import time
import numpy as np
import matplotlib.pyplot as plt
from tqdm import trange
pi = 3.1415926535

class point:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

class rect:
    p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]

    def __init__(self,a,b,c,d,e,f,g,h):
        self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]

    def print_point(self):
        print("Rect{")
        for i in  range(4):
            print(f"({self.p[i].x},{self.p[i].y})")
        print("}")
class vec:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

def normalize(v):
     len = np.sqrt(v.x*v.x+v.y*v.y)
    return vec(v.x /  len, v.y /  len)

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

```

```
46 def arc_len(mid,s):
47     a = s / 2 / pi
48     return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
49 
50 def point_distance(x,y,a,b):
51     return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))
52 
53 def get_nxt_point(x,y,otheta, len,s):
54     theta = otheta
55     l = 0
56     r = theta
57     a = s / 2 / pi
58     for _ in  range(50):
59         mid = (l + r) / 2
60         if(arc_len(theta,s) - arc_len(mid,s) <  len):
61             r = mid
62         else:
63             l = mid
64     ntheta = l
65     nrho = a * ntheta
66     nx, ny = Polar_to_Rectangular(nrho, ntheta)
67     return nx, ny, ntheta
68 
69 def get_prev_point(x,y,otheta,lenn,s):
70     theta = otheta
71     l = theta
72     r = theta + 2 * pi
73     a = s / 2 / pi
74     for _ in  range(50):
75         mid = (l + r) / 2
76         if(arc_len(mid,s) - arc_len(theta,s) < lenn):
77             l = mid
78         else:
79             r = mid
80     ntheta = l
81     nrho = a * ntheta
82     nx, ny = Polar_to_Rectangular(nrho, ntheta)
83     return nx, ny, ntheta
84 
85 def get_prev_board(x,y,otheta, len,s):
86     theta = otheta
87     l = theta
88     r = theta + pi
89     a = s / 2 / pi
90     for _ in  range(50):
91         mid = (l + r) / 2
92         tx, ty = Polar_to_Rectangular(a * mid, mid)
```

```
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15

    point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
    point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15

    point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
    point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15

    return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)

def quick_judge(a,b,c,d):
    if ( max(a.x,b.x) <  min(c.x,d.x) or
         max(c.x,d.x) <  min(a.x,b.x) or
         max(a.y,b.y) <  min(c.y,d.y) or
         max(c.y,d.y) <  min(a.y,b.y)) :
        return False
    else:
        return True

def xmult(a,b,c,d):
    vectorAx = b.x - a.x
    vectorAy = b.y - a.y
    vectorBx = d.x - c.x
    vectorBy = d.y - c.y
    return (vectorAx * vectorBy - vectorAy * vectorBx)
def cross(a,b,c,d):
    if not quick_judge(a,b,c,d):
        return False
    xmult1 = xmult(c,d,c,a)
    xmult2 = xmult(c,d,c,b)
```

```
    xmult3 = xmult(a,b,a,c)
    xmult4 = xmult(a,b,a,d)
    if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
        return True
    else:
        return False

def collide_detect(s,t):
    for i in  range(4):
        for j in  range(4):
            if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
                return True
    return False

def check(s):
    data = np.zeros((448))
    x = s * 16
    y = 0
    theta = 32 * pi
    org_x = x
    org_y = y
    org_theta = theta
    rect_array = []
    for t in  range(10000000000):
        if(t != 0):
            x = org_x
            y = org_y
            theta = org_theta
            if(point_distance(x,y,0,0)<4.5):
                print(f"{s} OK, True, t = {t}")
                return True
            x,y,theta = get_nxt_point(x,y,theta,1,s)
            org_x = x
            org_y = y
            org_theta = theta
        data[0] = x
        data[1] = y

        x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55,s)
        data[2] = x
        data[3] = y
        rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
        for i in  range(222):
            x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55,s)
            data[(i + 2) * 2] = x
            data[(i + 2) * 2 + 1] = y
            rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
```

```
        for i in  range( len(rect_array)):
            for j in  range(i+2,  len(rect_array)):
                if(collide_detect(rect_array[i],rect_array[j])):
                    print(f"{s} collided, False, t = {t}")
                    rect_array[i].print_point()
                    rect_array[j].print_point()

                    return False
        rect_array.clear()
    return False
L = 0.28125
R = 1

for _ in  range(30):
    mid = (L + R) / 2
    print(f"Now binary searching: L = {L}, R = {R}")
    if(check(mid)):
        R = mid
    else:
        L = mid

print(L)



# import pandas as pd
# df = pd.DataFrame(data_xlsx)
# df.to_excel("output2.xlsx",index=False)
# print("Done.")
```

Listing 5: **32.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
from tqdm import trange
pi = 3.1415926535

class point:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

class rect:
    p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]

```

```
    def __init__(self,a,b,c,d,e,f,g,h):
        self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]

    def print_point(self):
        print("Rect{")
        for i in  range(4):
            print(f"({self.p[i].x},{self.p[i].y})")
        print("}")
class vec:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

def normalize(v):
     len = np.sqrt(v.x*v.x+v.y*v.y)
    return vec(v.x /  len, v.y /  len)

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid,s):
    a = s / 2 / pi
    return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len,s):
    theta = otheta
    l = 0
    r = theta
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta,s) - arc_len(mid,s) <  len):
            r = mid
        else:
            l = mid
```

```
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn,s):
    theta = otheta
    l = theta
    r = theta + 2 * pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid,s) - arc_len(theta,s) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len,s):
    theta = otheta
    l = theta
    r = theta + pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15
```

```
111
112     point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
113     point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15
114
115     point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
116     point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15
117
118     return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)
119
120 def quick_judge(a,b,c,d):
121     if ( max(a.x,b.x) <  min(c.x,d.x) or
122          max(c.x,d.x) <  min(a.x,b.x) or
123          max(a.y,b.y) <  min(c.y,d.y) or
124          max(c.y,d.y) <  min(a.y,b.y)) :
125         return False
126     else:
127         return True
128
129 def xmult(a,b,c,d):
130     vectorAx = b.x - a.x
131     vectorAy = b.y - a.y
132     vectorBx = d.x - c.x
133     vectorBy = d.y - c.y
134     return (vectorAx * vectorBy - vectorAy * vectorBx)
135 def cross(a,b,c,d):
136     if not quick_judge(a,b,c,d):
137         return False
138     xmult1 = xmult(c,d,c,a)
139     xmult2 = xmult(c,d,c,b)
140     xmult3 = xmult(a,b,a,c)
141     xmult4 = xmult(a,b,a,d)
142     if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
143         return True
144     else:
145         return False
146
147 def collide_detect(s,t):
148     for i in  range(4):
149         for j in  range(4):
150             if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
151                 return True
152     return False
153
154 def check(s):
155     t_prev = 2000
156     data = np.zeros((448))
157     x = s * 16
```

```
    y = 0
    theta = 32 * pi
    org_x = x
    org_y = y
    org_theta = theta
    rect_array = []
    for t in  range(223 * 10):
        if(t != 0):
            x = org_x
            y = org_y
            theta = org_theta
            if(point_distance(x,y,0,0)<4.5):
                print(f"{s} OK, True, t = {t}")
                return True
            x,y,theta = get_nxt_point(x,y,theta,0.1,s)
            org_x = x
            org_y = y
            org_theta = theta
        if(t > t_prev):
            data[0] = x
            data[1] = y

            x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55,s)
            data[2] = x
            data[3] = y
            rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
            for i in  range(222):
                x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55,s)
                data[(i + 2) * 2] = x
                data[(i + 2) * 2 + 1] = y
                rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
            for i in  range( len(rect_array)):
                for j in  range(i+2,  len(rect_array)):
                    if(collide_detect(rect_array[i],rect_array[j])):
                        print(f"{s} collided, False, t = {t}")
                        rect_array[i].print_point()
                        rect_array[j].print_point()

                        return False
            rect_array.clear()
    return False
L = 0.4402
R = 0.4500

# L = 0.4502
# R = 0.4503 -> Now binary searching: L = 0.45029990234374995, R = 0.4503

```

```
for _ in  range(30):
    mid = (L + R) / 2
    print(f"Now binary searching: L = {L}, R = {R}")
    if(check(mid)):
        R = mid
    else:
        L = mid

print(L)


# import pandas as pd
# df = pd.DataFrame(data_xlsx)
# df.to_excel("output2.xlsx",index=False)
# print("Done.")
```

Listing 6: **33.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
from tqdm import trange
pi = 3.1415926535

class point:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

class rect:
    p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]

    def __init__(self,a,b,c,d,e,f,g,h):
        self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]

    def print_point(self):
        print("Rect{")
        for i in  range(4):
            print(f"({self.p[i].x},{self.p[i].y})")
        print("}")
class vec:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
```

```
        self.y = y

def normalize(v):
     len = np.sqrt(v.x*v.x+v.y*v.y)
    return vec(v.x /  len, v.y /  len)

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid,s):
    a = s / 2 / pi
    return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len,s):
    theta = otheta
    l = 0
    r = theta
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta,s) - arc_len(mid,s) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn,s):
    theta = otheta
    l = theta
    r = theta + 2 * pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid,s) - arc_len(theta,s) < lenn):
```

```
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len,s):
    theta = otheta
    l = theta
    r = theta + pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15

    point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
    point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15

    point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
    point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15

    return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)

def quick_judge(a,b,c,d):
    if ( max(a.x,b.x) <  min(c.x,d.x) or
         max(c.x,d.x) <  min(a.x,b.x) or
         max(a.y,b.y) <  min(c.y,d.y) or
```

```
124          max(c.y,d.y) <  min(a.y,b.y)) :
125          return False
126      else:
127          return True
128
129  def xmult(a,b,c,d):
130      vectorAx = b.x - a.x
131      vectorAy = b.y - a.y
132      vectorBx = d.x - c.x
133      vectorBy = d.y - c.y
134      return (vectorAx * vectorBy - vectorAy * vectorBx)
135  def cross(a,b,c,d):
136      if not quick_judge(a,b,c,d):
137          return False
138      xmult1 = xmult(c,d,c,a)
139      xmult2 = xmult(c,d,c,b)
140      xmult3 = xmult(a,b,a,c)
141      xmult4 = xmult(a,b,a,d)
142      if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
143          return True
144      else:
145          return False
146
147  def collide_detect(s,t):
148      for i in  range(4):
149          for j in  range(4):
150              if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
151                  return True
152      return False
153
154  def check(s):
155      t_prev = 215 * 100
156      data = np.zeros((448))
157      x = s * 16
158      y = 0
159      theta = 32 * pi
160      org_x = x
161      org_y = y
162      org_theta = theta
163      rect_array = []
164      for t in  range(223 * 100):
165          if(t != 0):
166              x = org_x
167              y = org_y
168              theta = org_theta
169              if(point_distance(x,y,0,0)<4.5):
170                  print(f"{s} OK, True, t = {t}")
```

```
                return True
            x,y,theta = get_nxt_point(x,y,theta,0.01,s)
            org_x = x
            org_y = y
            org_theta = theta
        if(t > t_prev):
            data[0] = x
            data[1] = y

            x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55,s)
            data[2] = x
            data[3] = y
            rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
            for i in  range(222):
                x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55,s)
                data[(i + 2) * 2] = x
                data[(i + 2) * 2 + 1] = y
                rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
            for i in  range( len(rect_array)):
                for j in  range(i+2,  len(rect_array)):
                    if(collide_detect(rect_array[i],rect_array[j])):
                        print(f"{s} collided, False, t = {t}")
                        rect_array[i].print_point()
                        rect_array[j].print_point()

                        return False
            rect_array.clear()
    return False
L = 0.4503
R = 0.4505

# L = 0.4502
# R = 0.4503 -> Now binary searching: L = 0.45029990234374995, R = 0.4503

for _ in  range(30):
    mid = (L + R) / 2
    print(f"Now binary searching: L = {L}, R = {R}")
    if(check(mid)):
        R = mid
    else:
        L = mid

print(L)


# import pandas as pd
# df = pd.DataFrame(data_xlsx)
```

```
# df.to_excel("output2.xlsx",index=False)
# print("Done.")
```

Listing 7: **34.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
from tqdm import trange
pi = 3.1415926535

class point:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

class rect:
    p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]

    def __init__(self,a,b,c,d,e,f,g,h):
        self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]

    def print_point(self):
        print("Rect{")
        for i in  range(4):
            print(f"({self.p[i].x},{self.p[i].y})")
        print("}")
class vec:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

def normalize(v):
     len = np.sqrt(v.x*v.x+v.y*v.y)
    return vec(v.x /  len, v.y /  len)

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
```

```
43      y = r * np.sin(theta)
44      return x, y
45
46  def arc_len(mid,s):
47      a = s / 2 / pi
48      return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
49
50  def point_distance(x,y,a,b):
51      return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))
52
53  def get_nxt_point(x,y,otheta, len,s):
54      theta = otheta
55      l = 0
56      r = theta
57      a = s / 2 / pi
58      for _ in  range(50):
59          mid = (l + r) / 2
60          if(arc_len(theta,s) - arc_len(mid,s) <  len):
61              r = mid
62          else:
63              l = mid
64      ntheta = l
65      nrho = a * ntheta
66      nx, ny = Polar_to_Rectangular(nrho, ntheta)
67      return nx, ny, ntheta
68
69  def get_prev_point(x,y,otheta,lenn,s):
70      theta = otheta
71      l = theta
72      r = theta + 2 * pi
73      a = s / 2 / pi
74      for _ in  range(50):
75          mid = (l + r) / 2
76          if(arc_len(mid,s) - arc_len(theta,s) < lenn):
77              l = mid
78          else:
79              r = mid
80      ntheta = l
81      nrho = a * ntheta
82      nx, ny = Polar_to_Rectangular(nrho, ntheta)
83      return nx, ny, ntheta
84
85  def get_prev_board(x,y,otheta, len,s):
86      theta = otheta
87      l = theta
88      r = theta + pi
89      a = s / 2 / pi
```

```
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15

    point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
    point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15

    point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
    point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15

    return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)

def quick_judge(a,b,c,d):
    if ( max(a.x,b.x) <  min(c.x,d.x) or
         max(c.x,d.x) <  min(a.x,b.x) or
         max(a.y,b.y) <  min(c.y,d.y) or
         max(c.y,d.y) <  min(a.y,b.y)) :
        return False
    else:
        return True

def xmult(a,b,c,d):
    vectorAx = b.x - a.x
    vectorAy = b.y - a.y
    vectorBx = d.x - c.x
    vectorBy = d.y - c.y
    return (vectorAx * vectorBy - vectorAy * vectorBx)
def cross(a,b,c,d):
    if not quick_judge(a,b,c,d):
```

```
137         return False
138     xmult1 = xmult(c,d,c,a)
139     xmult2 = xmult(c,d,c,b)
140     xmult3 = xmult(a,b,a,c)
141     xmult4 = xmult(a,b,a,d)
142     if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
143         return True
144     else:
145         return False
146 
147 def collide_detect(s,t):
148     for i in  range(4):
149         for j in  range(4):
150             if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
151                 return True
152     return False
153 
154 def check(s):
155     t_prev = 215 * 1000
156     data = np.zeros((448))
157     x = s * 16
158     y = 0
159     theta = 32 * pi
160     org_x = x
161     org_y = y
162     org_theta = theta
163     rect_array = []
164     for t in  range(217 * 1000):
165         if(t != 0):
166             x = org_x
167             y = org_y
168             theta = org_theta
169             if(point_distance(x,y,0,0)<4.5):
170                 print(f"{s} OK, True, t = {t}")
171                 return True
172             x,y,theta = get_nxt_point(x,y,theta,0.001,s)
173             org_x = x
174             org_y = y
175             org_theta = theta
176         if(t > t_prev):
177             data[0] = x
178             data[1] = y
179 
180             x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55,s)
181             data[2] = x
182             data[3] = y
183             rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
```

```
            for i in  range(222):
                x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55,s)
                data[(i + 2) * 2] = x
                data[(i + 2) * 2 + 1] = y
                rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
            for i in  range( len(rect_array)):
                for j in  range(i+2,  len(rect_array)):
                    if(collide_detect(rect_array[i],rect_array[j])):
                        print(f"{s} collided, False, t = {t}")
                        rect_array[i].print_point()
                        rect_array[j].print_point()

                        return False
            rect_array.clear()
    print(f"{s} t == 217.000s, guaranteed, True, t = {t}")
    return True
L = 0.4503
R = 0.4505

# L = 0.4502
# R = 0.4503 -> Now binary searching: L = 0.45029990234374995, R = 0.4503

for _ in  range(30):
    mid = (L + R) / 2
    print(f"Now binary searching: L = {L}, R = {R}")
    if(check(mid)):
        R = mid
    else:
        L = mid

print(L)


# import pandas as pd
# df = pd.DataFrame(data_xlsx)
# df.to_excel("output2.xlsx",index=False)
# print("Done.")
```

Listing 8: **3check.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
from tqdm import trange
pi = 3.1415926535

class point:
```

```
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

class rect:
    p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]

    def __init__(self,a,b,c,d,e,f,g,h):
        self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]

    def print_point(self):
        print("Rect{")
        for i in  range(4):
            print(f"({self.p[i].x},{self.p[i].y})")
        print("}")
class vec:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

def normalize(v):
     len = np.sqrt(v.x*v.x+v.y*v.y)
    return vec(v.x /  len, v.y /  len)

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def arc_len(mid,s):
    a = s / 2 / pi
    return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len,s):
    theta = otheta
```

```
    l = 0
    r = theta
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta,s) - arc_len(mid,s) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn,s):
    theta = otheta
    l = theta
    r = theta + 2 * pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid,s) - arc_len(theta,s) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len,s):
    theta = otheta
    l = theta
    r = theta + pi
    a = s / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

```

```
102 def get_rectangle(x,y,a,b):
103     line_vec = normalize(vec(x-a,y-b))
104     up_vec = normalize(vec(line_vec.y, -line_vec.x))
105 
106     point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
107     point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15
108 
109     point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
110     point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15
111 
112     point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
113     point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15
114 
115     point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
116     point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15
117 
118     return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)
119 
120 def quick_judge(a,b,c,d):
121     if ( max(a.x,b.x) <  min(c.x,d.x) or
122          max(c.x,d.x) <  min(a.x,b.x) or
123          max(a.y,b.y) <  min(c.y,d.y) or
124          max(c.y,d.y) <  min(a.y,b.y)) :
125         return False
126     else:
127         return True
128 
129 def xmult(a,b,c,d):
130     vectorAx = b.x - a.x
131     vectorAy = b.y - a.y
132     vectorBx = d.x - c.x
133     vectorBy = d.y - c.y
134     return (vectorAx * vectorBy - vectorAy * vectorBx)
135 def cross(a,b,c,d):
136     if not quick_judge(a,b,c,d):
137         return False
138     xmult1 = xmult(c,d,c,a)
139     xmult2 = xmult(c,d,c,b)
140     xmult3 = xmult(a,b,a,c)
141     xmult4 = xmult(a,b,a,d)
142     if xmult1 * xmult2 < 0 and xmult3 * xmult4 < 0:
143         return True
144     else:
145         return False
146 
147 def collide_detect(s,t):
148     for i in  range(4):
```

```
        for j in  range(4):
            if(cross(s.p[i], s.p[i + 1], t.p[j], t.p[j + 1])):
                return True
    return False

def check(s):
    t_prev = 200 * 10
    data = np.zeros((448))
    x = s * 16
    y = 0
    theta = 32 * pi
    org_x = x
    org_y = y
    org_theta = theta
    rect_array = []
    for t in trange(223 * 10):
        if(t != 0):
            x = org_x
            y = org_y
            theta = org_theta
            if(point_distance(x,y,0,0)<4.5):
                print(f"{s} OK, True, t = {t}")
                return True
            x,y,theta = get_nxt_point(x,y,theta,0.1,s)
            org_x = x
            org_y = y
            org_theta = theta
        if(t > t_prev):
            data[0] = x
            data[1] = y

            x,y,theta = get_prev_board(x,y,theta,3.41 - 0.55,s)
            data[2] = x
            data[3] = y
            rect_array.append(get_rectangle(data[0],data[1],data[2],data[3]))
            for i in  range(222):
                x,y,theta = get_prev_board(x,y,theta, 2.2 - 0.55,s)
                data[(i + 2) * 2] = x
                data[(i + 2) * 2 + 1] = y
                rect_array.append(get_rectangle(data[(i+2)*2-2],data[(i+2)*2-1],data[(i+2)*2],data[(i+2)*2+1]))
            for i in  range( len(rect_array)):
                for j in  range(i+2,  len(rect_array)):
                    if(collide_detect(rect_array[i],rect_array[j])):
                        print(f"{s} collided, False, t = {t}")
                        rect_array[i].print_point()
                        rect_array[j].print_point()

```

```
                        return False
            rect_array.clear()
    return False

import random
ans = 0.450337
for _ in  range(1,101):
    test_mid = ans - 0.01 + random.random() * 0.01
    if check(test_mid):
        print(f"#{_} test = {test_mid}, passed, model failed.")
        exit(0)
    else:
        print(f"#{_} test = {test_mid}, collided")

# import pandas as pd
# df = pd.DataFrame(data_xlsx)
# df.to_excel("output2.xlsx",index=False)
# print("Done.")
```

Listing 9: **4.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import trange,tqdm
pi = 3.1415926535
A = 1.7
arc_circle_1 = 9.080829937621441
arc_circle_2 = 9.080829937621441 / 2
radius_circle_1 = 3.005417667789
radius_circle_2 = 3.005417667789 / 2

class point:
    x = 0
    y = 0
    def __init__(self,x):
        self.x = x[0]
        self.y = x[1]

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    # theta = theta * (np.pi / 180)
```

```python
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y
# 绕 pointx,pointy 逆时针旋转
def Nrotate(angle,valuex,valuey,pointx,pointy):
    valuex = np.array(valuex)
    valuey = np.array(valuey)
    nRotatex = (valuex-pointx)*math.cos(angle) - (valuey-pointy)*math.sin(angle) + pointx
    nRotatey = (valuex-pointx)*math.sin(angle) + (valuey-pointy)*math.cos(angle) + pointy
    return nRotatex, nRotatey
# 绕 pointx,pointy 顺时针旋转
def Srotate(angle,valuex,valuey,pointx,pointy):
    valuex = np.array(valuex)
    valuey = np.array(valuey)
    sRotatex = (valuex-pointx)*math.cos(angle) + (valuey-pointy)*math.sin(angle) + pointx
    sRotatey = (valuey-pointy)*math.cos(angle) - (valuex-pointx)*math.sin(angle) + pointy
    return sRotatex,sRotatey

def arc_len(mid):
    a = A / 2 / pi
    arc_in = a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
    return arc_in

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len):
    # rho, theta = Rectangular_to_Polar(x, y)
    # print(rho, theta)
    theta = otheta
    l = 0
    r = theta
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta) - arc_len(mid) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn):
```

```
    # rho, theta = Rectangular_to_Polar(x, y)
    # print(rho, theta)
    theta = otheta
    l = theta
    r = theta + 200 * pi
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid) - arc_len(theta) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

A_point = point((-2.7118558637066594, -3.591077522761074))
H = point((-0.7600091166555,-1.3057264263462)) # 大圆圆心
G = point((1.7359324901811, 2.4484019745536)) # 小圆圆心
C = point((0.9039519545689, 1.1970258409204)) # 两圆相切处
A_r = np.sqrt(A_point.x * A_point.x + A_point.y * A_point.y)
A_theta = 2 * pi * A_r / A
# print(arc_len(A_theta))

ax, ay, start_theta= get_prev_point(A_point.x,A_point.y,A_theta,100)
```

```
# print(arc_len(start_theta))

def position(x):
    if x < 0:
        return get_prev_point(ax, ay, start_theta, -x)[0:2]
    elif x == 0:
        return ax, ay
    elif x <= 100:
        return get_nxt_point(ax, ay, start_theta, x)[0:2]
    elif x > 100 and x <= 100 + arc_circle_1:
         len = x - 100
        return Srotate( len / radius_circle_1, A_point.x, A_point.y, H.x, H.y)
    elif x > 100 + arc_circle_1 and x <= 100 + arc_circle_1 + arc_circle_2:
         len = x - (100 + arc_circle_1)
        return Nrotate( len / radius_circle_2, C.x, C.y, G.x, G.y)
    else:
        t_x, t_y, t_theta = get_prev_point(A_point.x, A_point.y, A_theta, x - (100 + arc_circle_1 +
            arc_circle_2))
        return -t_x, -t_y#, t_theta

def get_prev_board_position(pos, len):
    U = 0.1
    t_pos = pos - U
    pos_x, pos_y = position(pos)
    while True:
        t_x1, t_y1 = position(t_pos)
        t_x2, t_y2 = position(t_pos - U)

        if (point_distance(pos_x, pos_y, t_x1, t_y1) -
            len) * (point_distance(pos_x, pos_y, t_x2, t_y2) -  len) <= 0:
            L = t_pos - U
            R = t_pos
            for _ in  range(25):
                mid = (L + R) / 2
                mid_x, mid_y = position(mid)
                if(point_distance(pos_x, pos_y, mid_x, mid_y) <  len):
                    R = mid
                else:
                    L = mid
            # print(f"binary search result: L = {t_pos-U}, R = {t_pos}, ANS = {L}")
            return L
        t_pos -= U

class data:
    t = -999
    pos = point((0,0))
    nxt = 0
```

```
166     prev = 0
167     speed = 0
168     def __init__(self,t,x,a,b):
169         self.t = t
170         self.pos = x
171         self.prev = a
172         self.nxt = b
173 
174 T = 0.01
175 
176 # x = ax
177 # y = ay
178 # theta = start_theta
179 
180 # org_x = x
181 # org_y = y
182 # org_theta = theta
183 data_array = []
184 
185 for t in trange(0, 201):
186     Time = t
187     x, y = position(t)
188     data_array.append(data(Time, point((x, y)), t + T, t - T))
189 
190     t = get_prev_board_position(t, 3.41 - 0.55)
191     x, y = position(t)
192     prev_data = data_array[ len(data_array) - 1]
193 
194     data_array.append(data(Time, point((x, y)),
195                            get_prev_board_position(prev_data.prev, 3.41 - 0.55),
196                            get_prev_board_position(prev_data.nxt, 3.41 - 0.55)))
197 
198     for i in  range(222):
199         t = get_prev_board_position(t, 2.2 - 0.55)
200         x, y = position(t)
201         prev_data = data_array[ len(data_array) - 1]
202         data_array.append(data(Time, point((x, y)),
203                         get_prev_board_position(prev_data.prev, 2.2 - 0.55),
204                         get_prev_board_position(prev_data.nxt, 2.2 - 0.55)))
205 
206 for i in trange( len(data_array)):
207     # print(data_array[i].t, data_array[i].prev,data_array[i].nxt,
        (data_array[i].prev-data_array[i].nxt)/T/2)
208     data_array[i].speed = (data_array[i].prev - data_array[i].nxt) / T / 2
209 
210 result_xlsx = np.zeros((448,201))
211 speed_xlsx = np.zeros((224,201))
```

```

for i in  range( len(data_array)):
    ans = data_array[i]
    result_xlsx[i * 2 - ans.t * 448][ans.t] = ans.pos.x
    result_xlsx[i * 2 + 1 - ans.t * 448][ans.t] = ans.pos.y
    speed_xlsx[i - ans.t * 224][ans.t] = ans.speed

import pandas as pd
df = pd.DataFrame(result_xlsx)
df.to_excel("output_4.xlsx",index=False)
print("Done. (1 / 2)")
df = pd.DataFrame(speed_xlsx)
df.to_excel("output_4_speed.xlsx",index=False)
print("Done. (2 / 2)")
```

Listing 10: **5.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import trange,tqdm
pi = 3.1415926535
A = 1.7
arc_circle_1 = 9.080829937621441
arc_circle_2 = 9.080829937621441 / 2
radius_circle_1 = 3.005417667789
radius_circle_2 = 3.005417667789 / 2

class point:
    x = 0
    y = 0
    def __init__(self,x):
        self.x = x[0]
        self.y = x[1]

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    # theta = theta * (np.pi / 180)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y
# 绕 pointx,pointy 逆时针旋转
```

```python
def Nrotate(angle,valuex,valuey,pointx,pointy):
    valuex = np.array(valuex)
    valuey = np.array(valuey)
    nRotatex = (valuex-pointx)*math.cos(angle) - (valuey-pointy)*math.sin(angle) + pointx
    nRotatey = (valuex-pointx)*math.sin(angle) + (valuey-pointy)*math.cos(angle) + pointy
    return nRotatex, nRotatey
# 绕 pointx,pointy 顺时针旋转
def Srotate(angle,valuex,valuey,pointx,pointy):
    valuex = np.array(valuex)
    valuey = np.array(valuey)
    sRotatex = (valuex-pointx)*math.cos(angle) + (valuey-pointy)*math.sin(angle) + pointx
    sRotatey = (valuey-pointy)*math.cos(angle) - (valuex-pointx)*math.sin(angle) + pointy
    return sRotatex,sRotatey

def arc_len(mid):
    a = A / 2 / pi
    arc_in = a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
    return arc_in

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_nxt_point(x,y,otheta, len):
    # rho, theta = Rectangular_to_Polar(x, y)
    # print(rho, theta)
    theta = otheta
    l = 0
    r = theta
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta) - arc_len(mid) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn):
    # rho, theta = Rectangular_to_Polar(x, y)
    # print(rho, theta)
    theta = otheta
    l = theta
```

```
    r = theta + 200 * pi
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid) - arc_len(theta) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

A_point = point((-2.7118558637066594, -3.591077522761074))
H = point((-0.7600091166555,-1.3057264263462)) # 大圆圆心
G = point((1.7359324901811, 2.4484019745536)) # 小圆圆心
C = point((0.9039519545689, 1.1970258409204)) # 两圆相切处
A_r = np.sqrt(A_point.x * A_point.x + A_point.y * A_point.y)
A_theta = 2 * pi * A_r / A
# print(arc_len(A_theta))

ax, ay, start_theta= get_prev_point(A_point.x,A_point.y,A_theta,100)
# print(arc_len(start_theta))

def position(x):
    if x < 0:
```

```
125         return get_prev_point(ax, ay, start_theta, -x)[0:2]
126     elif x == 0:
127         return ax, ay
128     elif x <= 100:
129         return get_nxt_point(ax, ay, start_theta, x)[0:2]
130     elif x > 100 and x <= 100 + arc_circle_1:
131          len = x - 100
132         return Srotate( len / radius_circle_1, A_point.x, A_point.y, H.x, H.y)
133     elif x > 100 + arc_circle_1 and x <= 100 + arc_circle_1 + arc_circle_2:
134          len = x - (100 + arc_circle_1)
135         return Nrotate( len / radius_circle_2, C.x, C.y, G.x, G.y)
136     else:
137         t_x, t_y, t_theta = get_prev_point(A_point.x, A_point.y, A_theta, x - (100 + arc_circle_1 +
                arc_circle_2))
138         return -t_x, -t_y#, t_theta
139
140 def get_prev_board_position(pos, len):
141     U = 0.1
142     t_pos = pos - U
143     pos_x, pos_y = position(pos)
144     while True:
145         t_x1, t_y1 = position(t_pos)
146         t_x2, t_y2 = position(t_pos - U)
147
148         if (point_distance(pos_x, pos_y, t_x1, t_y1) -
                len) * (point_distance(pos_x, pos_y, t_x2, t_y2) -  len) <= 0:
149             L = t_pos - U
150             R = t_pos
151             for _ in  range(25):
152                 mid = (L + R) / 2
153                 mid_x, mid_y = position(mid)
154                 if(point_distance(pos_x, pos_y, mid_x, mid_y) <  len):
155                     R = mid
156                 else:
157                     L = mid
158             # print(f"binary search result: L = {t_pos-U}, R = {t_pos}, ANS = {L}")
159             return L
160         t_pos -= U
161
162 class data:
163     t = -999
164     pos = point((0,0))
165     nxt = 0
166     prev = 0
167     speed = 0
168     def __init__(self,t,x,a,b):
169         self.t = t
```

```
        self.pos = x
        self.prev = a
        self.nxt = b

T = 0.0001

# x = ax
# y = ay
# theta = start_theta

# org_x = x
# org_y = y
# org_theta = theta
# data_array = []

def check(head_speed):
    t_init = 113.62124490643217
    delta_t = 0
    max_speed = 0
    while delta_t <= 3:
        data_array = []
        t = t_init + delta_t

        x, y = position(t)
        data_array.append(data(0, point((x, y)), t + head_speed * T, t - head_speed * T))
        t = get_prev_board_position(t, 3.41 - 0.55)
        x, y = position(t)
        prev_data = data_array[ len(data_array) - 1]

        speed = (get_prev_board_position(prev_data.prev, 3.41 - 0.55) -
            get_prev_board_position(prev_data.nxt, 3.41 - 0.55)) / T / 2
        # print(speed)
        if(speed > 2):
            print(f"head speed = {head_speed}, t = {t} exceeded, false")
            return False
        max_speed =  max(max_speed, speed)
        # data_array.append(data(Time, point((x, y)),
        #                 get_prev_board_position(prev_data.prev, 2.2 - 0.55),
        #                 get_prev_board_position(prev_data.nxt, 2.2 - 0.55)))
        delta_t += 0.001
    print(f"head speed = {head_speed}, OK, true, max speed = {max_speed} < 2")
    return True

# L = 1
# R = 1.5 # 第一次二分
L = 1.15
R = 1.35
```

```
216
217 # check(1)
218 # exit(0)
219 for _ in  range(30):
220     mid = (L + R) / 2
221     print(f"binary search L = {L}, R = {R}")
222     if check(mid):
223         L = mid
224     else:
225         R = mid
226
227 print(L) # 1.2838430792093278
228
229 # for t in trange(0, 3):
230 #     Time = t
231 #     t *= 1.2
232 #     x, y = position(t)
233 #     data_array.append(data(Time, point((x, y)), t + 1.2*T, t - 1.2*T))
234
235 #     t = get_prev_board_position(t, 3.41 - 0.55)
236 #     x, y = position(t)
237 #     prev_data = data_array[len(data_array) - 1]
238
239 #     data_array.append(data(Time, point((x, y)),
240 #                            get_prev_board_position(prev_data.prev, 3.41 - 0.55),
241 #                            get_prev_board_position(prev_data.nxt, 3.41 - 0.55)))
242
243 #     for i in range(222):
244 #         t = get_prev_board_position(t, 2.2 - 0.55)
245 #         x, y = position(t)
246 #         prev_data = data_array[len(data_array) - 1]
247 #         data_array.append(data(Time, point((x, y)),
248 #                           get_prev_board_position(prev_data.prev, 2.2 - 0.55),
249 #                           get_prev_board_position(prev_data.nxt, 2.2 - 0.55)))
250
251 # for i in trange(len(data_array)):
252 #     # print(data_array[i].t, data_array[i].prev,data_array[i].nxt,
    (data_array[i].prev-data_array[i].nxt)/T/2)
253 #     data_array[i].speed = (data_array[i].prev - data_array[i].nxt) / T / 2
254
255 # result_xlsx = np.zeros((448,201))
256 # speed_xlsx = np.zeros((224,201))
257
258 # for i in range(len(data_array)):
259 #     ans = data_array[i]
260 #     result_xlsx[i * 2 - ans.t * 448][ans.t] = ans.pos.x
261 #     result_xlsx[i * 2 + 1 - ans.t * 448][ans.t] = ans.pos.y
```

```
#     speed_xlsx[i - ans.t * 224][ans.t] = ans.speed

# import pandas as pd
# df = pd.DataFrame(result_xlsx)
# df.to_excel("output_4_1_2.xlsx",index=False)
# print("Done. (1 / 2)")
# df = pd.DataFrame(speed_xlsx)
# df.to_excel("output_4_speed_1_2.xlsx",index=False)
# print("Done. (2 / 2)")
```

Listing 11: **5check.py**

```
import time
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import trange,tqdm
pi = 3.1415926535
A = 1.7
arc_circle_1 = 9.080829937621441
arc_circle_2 = 9.080829937621441 / 2
radius_circle_1 = 3.005417667789
radius_circle_2 = 3.005417667789 / 2

class point:
    x = 0
    y = 0
    def __init__(self,x):
        self.x = x[0]
        self.y = x[1]

def Rectangular_to_Polar(x, y):
    r = np.sqrt(np.square(x) + np.square(y))
    theta = np.degrees(np.arctan(y / x))
    return r, theta

def Polar_to_Rectangular(r, theta):
    # theta = theta * (np.pi / 180)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y
# 绕 pointx,pointy 逆时针旋转
def Nrotate(angle,valuex,valuey,pointx,pointy):
    valuex = np.array(valuex)
    valuey = np.array(valuey)
    nRotatex = (valuex-pointx)*math.cos(angle) - (valuey-pointy)*math.sin(angle) + pointx
    nRotatey = (valuex-pointx)*math.sin(angle) + (valuey-pointy)*math.cos(angle) + pointy
```

```
36      return nRotatex, nRotatey
37  # 绕 pointx,pointy 顺时针旋转
38  def Srotate(angle,valuex,valuey,pointx,pointy):
39      valuex = np.array(valuex)
40      valuey = np.array(valuey)
41      sRotatex = (valuex-pointx)*math.cos(angle) + (valuey-pointy)*math.sin(angle) + pointx
42      sRotatey = (valuey-pointy)*math.cos(angle) - (valuex-pointx)*math.sin(angle) + pointy
43      return sRotatex,sRotatey
44
45  def arc_len(mid):
46      a = A / 2 / pi
47      arc_in = a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
48      return arc_in
49
50  def point_distance(x,y,a,b):
51      return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))
52
53  def get_nxt_point(x,y,otheta, len):
54      # rho, theta = Rectangular_to_Polar(x, y)
55      # print(rho, theta)
56      theta = otheta
57      l = 0
58      r = theta
59      a = A / 2 / pi
60      for _ in  range(50):
61          mid = (l + r) / 2
62          if(arc_len(theta) - arc_len(mid) <  len):
63              r = mid
64          else:
65              l = mid
66      ntheta = l
67      nrho = a * ntheta
68      # print(nrho, ntheta)
69      nx, ny = Polar_to_Rectangular(nrho, ntheta)
70      # print(nx,ny)
71      return nx, ny, ntheta
72
73  def get_prev_point(x,y,otheta,lenn):
74      # rho, theta = Rectangular_to_Polar(x, y)
75      # print(rho, theta)
76      theta = otheta
77      l = theta
78      r = theta + 200 * pi
79      a = A / 2 / pi
80      for _ in  range(50):
81          mid = (l + r) / 2
82          if(arc_len(mid) - arc_len(theta) < lenn):
```

```
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = A / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    # print(nrho, ntheta)
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    # print(nx,ny)
    return nx, ny, ntheta

A_point = point((-2.7118558637066594, -3.591077522761074))
H = point((-0.7600091166555,-1.3057264263462)) # 大圆圆心
G = point((1.7359324901811, 2.4484019745536)) # 小圆圆心
C = point((0.9039519545689, 1.1970258409204)) # 两圆相切处
A_r = np.sqrt(A_point.x * A_point.x + A_point.y * A_point.y)
A_theta = 2 * pi * A_r / A
# print(arc_len(A_theta))

ax, ay, start_theta= get_prev_point(A_point.x,A_point.y,A_theta,100)
# print(arc_len(start_theta))

def position(x):
    if x < 0:
        return get_prev_point(ax, ay, start_theta, -x)[0:2]
    elif x == 0:
        return ax, ay
    elif x <= 100:
        return get_nxt_point(ax, ay, start_theta, x)[0:2]
```

```
    elif x > 100 and x <= 100 + arc_circle_1:
         len = x - 100
        return Srotate( len / radius_circle_1, A_point.x, A_point.y, H.x, H.y)
    elif x > 100 + arc_circle_1 and x <= 100 + arc_circle_1 + arc_circle_2:
         len = x - (100 + arc_circle_1)
        return Nrotate( len / radius_circle_2, C.x, C.y, G.x, G.y)
    else:
        t_x, t_y, t_theta = get_prev_point(A_point.x, A_point.y, A_theta, x - (100 + arc_circle_1 +
            arc_circle_2))
        return -t_x, -t_y#, t_theta

def get_prev_board_position(pos, len):
    U = 0.1
    t_pos = pos - U
    pos_x, pos_y = position(pos)
    while True:
        t_x1, t_y1 = position(t_pos)
        t_x2, t_y2 = position(t_pos - U)

        if (point_distance(pos_x, pos_y, t_x1, t_y1) -
            len) * (point_distance(pos_x, pos_y, t_x2, t_y2) -  len) <= 0:
            L = t_pos - U
            R = t_pos
            for _ in  range(25):
                mid = (L + R) / 2
                mid_x, mid_y = position(mid)
                if(point_distance(pos_x, pos_y, mid_x, mid_y) <  len):
                    R = mid
                else:
                    L = mid
            # print(f"binary search result: L = {t_pos-U}, R = {t_pos}, ANS = {L}")
            return L
        t_pos -= U

class data:
    t = -999
    pos = point((0,0))
    nxt = 0
    prev = 0
    speed = 0
    def __init__(self,t,x,a,b):
        self.t = t
        self.pos = x
        self.prev = a
        self.nxt = b

T = 0.0001
```

```
175
176 # x = ax
177 # y = ay
178 # theta = start_theta
179
180 # org_x = x
181 # org_y = y
182 # org_theta = theta
183 # data_array = []
184
185 def check(head_speed):
186     t_init = 113.62124490643217
187     delta_t = 0
188     max_speed = 0
189     while delta_t <= 3:
190         data_array = []
191         t = t_init + delta_t
192
193         x, y = position(t)
194         data_array.append(data(0, point((x, y)), t + head_speed * T, t - head_speed * T))
195         t = get_prev_board_position(t, 3.41 - 0.55)
196         x, y = position(t)
197         prev_data = data_array[ len(data_array) - 1]
198
199         speed = (get_prev_board_position(prev_data.prev, 3.41 - 0.55) -
                 get_prev_board_position(prev_data.nxt, 3.41 - 0.55)) / T / 2
200         # print(speed)
201         if(speed > 2):
202             print(f"head speed = {head_speed}, t = {t} exceeded, false")
203             return False
204         max_speed =  max(max_speed, speed)
205         # data_array.append(data(Time, point((x, y)),
206         #                 get_prev_board_position(prev_data.prev, 2.2 - 0.55),
207         #                 get_prev_board_position(prev_data.nxt, 2.2 - 0.55)))
208         delta_t += 0.001
209     print(f"head speed = {head_speed}, OK, true, max speed = {max_speed} < 2")
210     return True
211
212 ans = 1.2838430792093278
213 import random
214 for _ in  range(1, 101):
215     test_speed = ans + random.random() * 0.01
216     if check(test_speed):
217         print(f"#{_} test = {test_speed}, speed < 2m/s, model failed.")
218         exit(0)
219     else:
220         print(f"#{_} test = {test_speed}, exceeded, pass")
```

Listing 12: **plot.py**

```python
#问题1 初始时刻板凳龙示意图

import matplotlib.pyplot as plt
import numpy as np
pi = 3.1415926535
head_len = 3.41 - 0.55
body_len = 2.2 - 0.55

class point:
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x = x
        self.y = y

def Polar_to_Rectangular(r, theta):  # 极坐标转直角坐标，输入的theta需为角度值
    # theta = theta * (np.pi / 180)
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

def point_distance(x,y,a,b):
    return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = 0.55 / 2 / pi
    for _ in  range(30):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta
plt.figure(figsize=(6, 6))
line_x = []
line_y = []
```

```
for i in  range(5000):
    atheta = 45 * pi / 5000 * i
    arho = atheta * 0.55 / 2 / pi
    ax, ay = Polar_to_Rectangular(arho, atheta)
    line_x.append(ax)
    line_y.append(ay)
plt.plot(line_x,line_y)

start_point = point(8.8, 0)
start_theta = 32 * pi
plt.scatter(start_point.x, start_point.y, color = 'green')

tx = start_point.x
ty = start_point.y
ttheta = start_theta

x,y,theta = get_prev_board(tx,ty,ttheta,head_len)
plt.scatter(x,y,color = 'green')
plt.plot([tx,x],[ty,y], linewidth = 3, color = 'red')

tx = x
ty = y
ttheta = theta
for i in  range(222):
    x,y,theta = get_prev_board(tx,ty,ttheta,body_len)
    plt.scatter(x,y,color = 'green')
    plt.plot([tx,x],[ty,y], linewidth = 3, color = 'red')
    tx = x
    ty = y
    ttheta = theta
plt.savefig("plot_1.eps", dpi=300)
plt.show()
```

Listing 13: **plot2.py**

```
#问题2 碰撞时刻板凳龙示意图

import matplotlib.pyplot as plt
import numpy as np
pi = 3.1415926535
head_len = 3.41 - 0.55
body_len = 2.2 - 0.55
crash_time = 412.47383767112257

class point:
    x = 0
    y = 0
```

```
13      def __init__(self,x,y):
14          self.x = x
15          self.y = y
16
17  class rect:
18      p = [point(0,0),point(0,0),point(0,0),point(0,0),point(0,0)]
19
20      def __init__(self,a,b,c,d,e,f,g,h):
21          self.p = [point(a,b),point(c,d),point(e,f),point(g,h),point(a,b)]
22
23      def print_point(self):
24          print("Rect{")
25          for i in  range(4):
26              print(f"({self.p[i].x},{self.p[i].y})")
27          print("}")
28
29  class vec:
30      x = 0
31      y = 0
32      def __init__(self,x,y):
33          self.x = x
34          self.y = y
35
36  def normalize(v):
37       len = np.sqrt(v.x*v.x+v.y*v.y)
38      return vec(v.x /  len, v.y /  len)
39
40  def Rectangular_to_Polar(x, y):
41      r = np.sqrt(np.square(x) + np.square(y))
42      theta = np.degrees(np.arctan(y / x))
43      return r, theta
44
45  def Polar_to_Rectangular(r, theta):
46      x = r * np.cos(theta)
47      y = r * np.sin(theta)
48      return x, y
49
50  def arc_len(mid):
51      a = 0.55 / 2 / pi
52      return a / 2 * (mid * np.sqrt(mid * mid + 1) + np.log(mid + np.sqrt(mid * mid + 1)))
53
54  def point_distance(x,y,a,b):
55      return np.sqrt((x-a)*(x-a)+(y-b)*(y-b))
56
57  def get_nxt_point(x,y,otheta, len):
58      theta = otheta
59      l = 0
```

```
    r = theta
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(theta) - arc_len(mid) <  len):
            r = mid
        else:
            l = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_point(x,y,otheta,lenn):
    theta = otheta
    l = theta
    r = theta + 2 * pi
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        if(arc_len(mid) - arc_len(theta) < lenn):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_prev_board(x,y,otheta, len):
    theta = otheta
    l = theta
    r = theta + pi
    a = 0.55 / 2 / pi
    for _ in  range(50):
        mid = (l + r) / 2
        tx, ty = Polar_to_Rectangular(a * mid, mid)
        if(point_distance(x, y, tx, ty) <  len):
            l = mid
        else:
            r = mid
    ntheta = l
    nrho = a * ntheta
    nx, ny = Polar_to_Rectangular(nrho, ntheta)
    return nx, ny, ntheta

def get_rectangle(x,y,a,b):
```

```
    line_vec = normalize(vec(x-a,y-b))
    up_vec = normalize(vec(line_vec.y, -line_vec.x))

    point1_x = x + line_vec.x * 0.275 + up_vec.x * 0.15
    point1_y = y + line_vec.y * 0.275 + up_vec.y * 0.15

    point2_x = x + line_vec.x * 0.275 + up_vec.x * -0.15
    point2_y = y + line_vec.y * 0.275 + up_vec.y * -0.15

    point3_x = a + line_vec.x * -0.275 + up_vec.x * -0.15
    point3_y = b + line_vec.y * -0.275 + up_vec.y * -0.15

    point4_x = a + line_vec.x * -0.275 + up_vec.x * 0.15
    point4_y = b + line_vec.y * -0.275 + up_vec.y * 0.15

    return rect(point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y)

def draw_rectangle(recta,color1):
    x = [recta.p[0].x, recta.p[1].x, recta.p[2].x, recta.p[3].x,recta.p[0].x]
    y = [recta.p[0].y, recta.p[1].y, recta.p[2].y, recta.p[3].y,recta.p[0].y]
    plt.plot(x,y,linewidth = 3, color = color1)
plt.figure(figsize=(6, 6))
line_x = []
line_y = []
for i in  range(5000):
    atheta = 12.5 * pi / 5000 * i
    arho = atheta * 0.55 / 2 / pi
    ax, ay = Polar_to_Rectangular(arho, atheta)
    line_x.append(ax)
    line_y.append(ay)
plt.plot(line_x,line_y)

start_point = point(8.8,0)
start_theta = 32 * pi

x,y,theta = get_nxt_point(start_point.x, start_point.y, start_theta, crash_time)
tx,ty,ttheta = get_prev_board(x,y,theta,head_len)
plt.plot(x,y,'o')
plt.plot(tx,ty,'o')
draw_rectangle(get_rectangle(x,y,tx,ty),'blue')

for i in  range(18):
    x,y,theta = get_prev_board(tx,ty,ttheta,body_len)
    plt.plot(x,y,'o')
    draw_rectangle(get_rectangle(x,y,tx,ty),'red')
    tx = x
    ty = y
```

```
154     ttheta = theta
155 plt.savefig("plot_2.eps", dpi=300)
156 plt.show()
```