组号：7

浙江大学

本科实验报告

课程名称：	双足移动机器人强化实践
姓　　名：	徐绍俊，陈佳禾，周晨旭，裘小钰
院　　系：	控制科学与工程学院
专　　业：	自动化（控制）
指导老师：	周春琳
选课时间：	周日 3-10 节

2023 年 1 月 1 日

浙江大学实验报告

专业：自动化（控制）
姓名：徐绍俊，陈佳禾，周晨旭，裘小钰
日期：2023 年 1 月 1 日
地点：教 4-304

课程名称：双足移动机器人强化实践 指导老师：周春琳 成绩：
实验名称：太空机械臂运动仿真 实验类型：实物实验 同组学生姓名：无

一 实验目的和要求

1. 根据网络公开资料，设计一款中国空间站的机械臂 3D 模型，机械臂的关节和连杆可以简化为圆柱体，自定义机械臂的尺寸，要求臂展不超过 80cm
2. 设计开发机械臂的运动学控制程序，在仿真环境中展示太空机械臂的行走运动，如视频所示，至少展示行走 2 步，即从初始位置移动到位置 1，再移动到位置 2，移动姿态和速度在合理范围内自定义
3. 根据机械臂的尺寸，自定义基座位置，但两个基座需要满足轴线正交关系，如图所示
4. 假设机械臂末端于基座之间接触后可以通过电磁铁紧密吸合

二 实验内容

1. 3D 建模
   (a) 使用 3D 建模软件 (如 SolidWorks, Inventor 等)，搭建仿中国空间站天河机械臂的 3D 模型
   (b) 将 3D 模型拖入 CoppeliaSim，完成装配与坐标系建立
2. 正运动学
   构建标准 D-H 参数表，计算关节变换矩阵，完成正运动学
3. 逆运动学
   使用逆运动学求解工具 IKSolver 完成逆运动学解算，根据约束选择合适的解
4. 轨迹规划
   使用轨迹规划曲线 (样条函数法、梯形曲线法)，完成机械臂运动的轨迹规划

三 实验原理

四 实验结果与分析

4.1 3D 建模

1. 模型搭建在 soulidworks 中完成模型设计，并检验装配，模型结构中，我们仿造中国空间站天河二号上的机械臂进行设计，为了满足设计要求，将机械臂等比例缩放 20 倍，此时臂展为 62cm。

2. 仿真装配在 CoppeliaSim 中完成对模型的 3D 装配，检验效果如下（机械臂与太空舱初始位置）

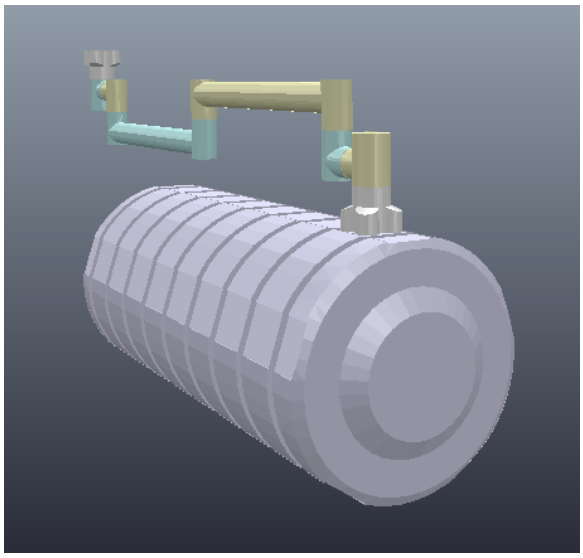


图 1: 装配结果（初始位置）

4.2　正运动学

由于机械臂是对称的，在这里我们考虑将机械臂的规划分为两个部分进行，以此来减少逆运动学关节位置求解的计算量（将七自由度转化为两个三自由度关节与一个一自由度关节的结合）。其前三个关节对应的 DH 参数表如下

1. DH 参数表

i	α_i	a_i	d_i	θ_i
1	$-\pi/2$	0	50	θ_1
2	$\pi/2$	0	50	θ_2
3	0	260	50	$\theta_3 - \pi/2$
4	0	0	50	θ_4

2. 变换矩阵

$$\begin{bmatrix} s_1c_{34} + s_{34}c_1c_2 & -s_1s_{34} + c_1c_2c_{34} & s_2c_1 & 260s_1c_3 - 50s_1 + 100s_2c_1 + 260s_3c_1c_2 \\ s_1s_{34}c_2 - c_1c_{34} & s_1c_2c_{34} + s_{34}c_1 & s_1s_2 & 100s_1s_2 + 260s_1s_3c_2 - 260c_1c_3 + 50c_1 \\ -s_2s_{34} & -s_2c_{34} & c_2 & -260s_2s_3 + 100c_2 + 50 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.3　逆运动学

由于采用先考虑机械臂前三个和后三个关节的方式来求解逆运动学，可以想到由于关节数量减少，我们无法控制第四个关键达到工作空间中的任意位姿，因而其能够达到的位置是受限的，在这里我们仅

求解在位置上对应的关节角（由于是退化情况，在给定位置上，四自由度关节可能拥有无穷组解）。通过设计让第四个关节处于两个落脚点的中垂面或角平分面上，同时控制第四个关节对两个落脚点有相同的视角与距离，可以通过联立两个三关节机械臂的正运动学，找到一个两个三关节机械臂均能达到的位姿，进而完成对七关节机械臂逆运动学的求解

在正运动学中，我们已经计算得到了 ${}^0T_4$ 的求解结果如下:

$$\begin{bmatrix} s_1c_{34}+s_{34}c_1c_2 & -s_1s_{34}+c_1c_2c_{34} & s_2c_1 & 260s_1c_3-50s_1+100s_2c_1+260s_3c_1c_2 \\ s_1s_{34}c_2-c_1c_{34} & s_1c_2c_{34}+s_{34}c_1 & s_1s_2 & 100s_1s_2+260s_1s_3c_2-260c_1c_3+50c_1 \\ -s_2s_{34} & -s_2c_{34} & c_2 & -260s_2s_3+100c_2+50 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

通过末端位姿参数 R，P 求出正运动学的变换矩阵 ${}^0T_4$:

$$\text{末端位姿} = (x, y, z, \alpha, \beta, \gamma)$$

$${}^0T_4 = \begin{bmatrix} c\alpha c\beta & -s\alpha c\beta & s\beta & x \\ s\alpha c\gamma + c\alpha s\beta s\gamma & c\alpha c\gamma - s\alpha s\beta s\gamma & -c\beta s\gamma & y \\ s\alpha s\gamma - c\alpha s\beta c\gamma & c\alpha s\gamma + s\alpha s\beta c\gamma & c\beta c\gamma & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

为了简化计算，将该矩阵做如下标记

$$T = \begin{bmatrix} r_1 & r_2 & r_3 & x \\ r_4 & r_5 & r_6 & y \\ r_7 & r_8 & r_9 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

逆变换矩阵 ${}^iT_{i-1}$ 如下:

$$\begin{bmatrix} c\theta_i & s\theta_i & 0 & -a_i \\ -c\alpha_i s\theta_i & c\alpha_i c\theta_i & s\alpha_i & -d_i s\alpha_i \\ s\alpha_i s\theta_i & -s\alpha_i c\theta_i & c\alpha_i & -d_i c\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

利用以下等式:

$${}^1T_0\,{}^0T_4 = {}^1T_2\,{}^2T_3\,{}^3T_4 = {}^1T_4$$

求解得到:

$${}^1T_4(1) = \begin{bmatrix} s_{34}c_2 & c_2c_{34} & s_2 & 100s_2+260s_3c_2 \\ s_2s_{34} & s_2c_{34} & -c_2 & 260s_2s_3-100c_2 \\ -c_{34} & s_{34} & 0 & 50-260c_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
{}^1T_4(2) = \begin{bmatrix} r_1c_1 + r_4s_1 & r_2c_1 + r_5s_1 & r_3c_1 + r_6s_1 & xc_1 + ys_1 \\ -r_7 & -r_8 & -r_9 & 50 - z \\ -r_1s_1 + r_4c_1 & -r_2s_1 + r_5c_1 & -r_3s_1 + r_6c_1 & -xs_1 + yc_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

根据 $(1,4)$、$(2,4)$、$(3,4)$ 结果相等，得到以下方程:

$$
100s_2 + 260s_3c_2 = xc_1 + ys_1 \tag{1}
$$

$$
260s_2s_3 - 100c_2 = 50 - z \tag{2}
$$

$$
50 - 260c_3 = -xs_1 + yc_1 \tag{3}
$$

利用以下等式:

$$
{}^2T_0{}^0T_4 = {}^2T_3{}^3T_4 = {}^2T_4
$$

求解得到:

$$
{}^2T_4(1) = \begin{bmatrix} s_{34} & c_{34} & 0 & 260s_3 \\ -c_{34} & s_{34} & 0 & -260c_3 \\ 0 & 0 & 1 & 100 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^2T_4(2) = \begin{bmatrix} r_1c_1c_2 + r_4s_1c_2 - r_7s_2 & r_2c_1c_2 + r_5s_1c_2 - r_8s_2 & r_3c_1c_2 + r_6s_1c_2 - r_9s_2 & xc_1c_2 + ys_1c_2 - zs_2 + 50s_2 \\ -r_1s_1 + r_4c_1 & -r_2s_1 + r_5c_1 & -r_3s_1 + r_6c_1 & -xs_1 + yc_1 - 50 \\ r_1s_2c_1 + r_4s_1s_2 + r_7c_2 & r_2s_2c_1 + r_5s_1s_2 + r_8c_2 & r_3s_2c_1 + r_6s_1s_2 + r_9c_2 & xs_2c_1 + ys_1s_2 + zc_2 - 50c_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

根据 $(1,4)$、$(3,4)$ 结果相等，得到以下方程:

$$
260s_3 = xc_1c_2 + ys_1c_2 - zs_2 + 50s_2 \tag{4}
$$

$$
100 = xs_2c_1 + ys_1s_2 + zc_2 - 50c_2 \tag{5}
$$

联立方程 (1)、(4)、(5)，可以解得如下结果：

$$
\begin{aligned}
\theta_1 &= -2 * atan2((x * y + ((x^2 + y^2 + z^2 - 100 * z - 7500) * (100 * z + 2 * x * (-z^2 + 100 * z + 7500)^{\frac{1}{2}} \\
&\quad + x^2 - z^2 + 7500))^{\frac{1}{2}} + \frac{y * (-z^2 + 100 * z + 7500)^{\frac{1}{2}}}{-x^2 - z^2 + 100 * z + 7500} \\
\theta_2 &= -2 * \frac{atan2(-z^2 + 100 * z + 7500)^{\frac{1}{2}}}{z + 50} \\
\theta_3 &= \pi
\end{aligned} \tag{6}
$$

4.4　轨迹规划

轨迹规划中我们采用的是梯形速度曲线方法，规划的结果与机器人在仿真过程中的姿态截图如下所示：

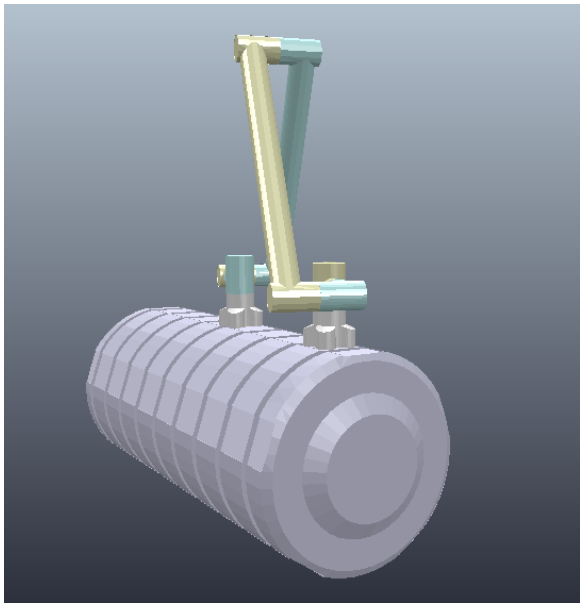


图 2: 第一步落脚后状态

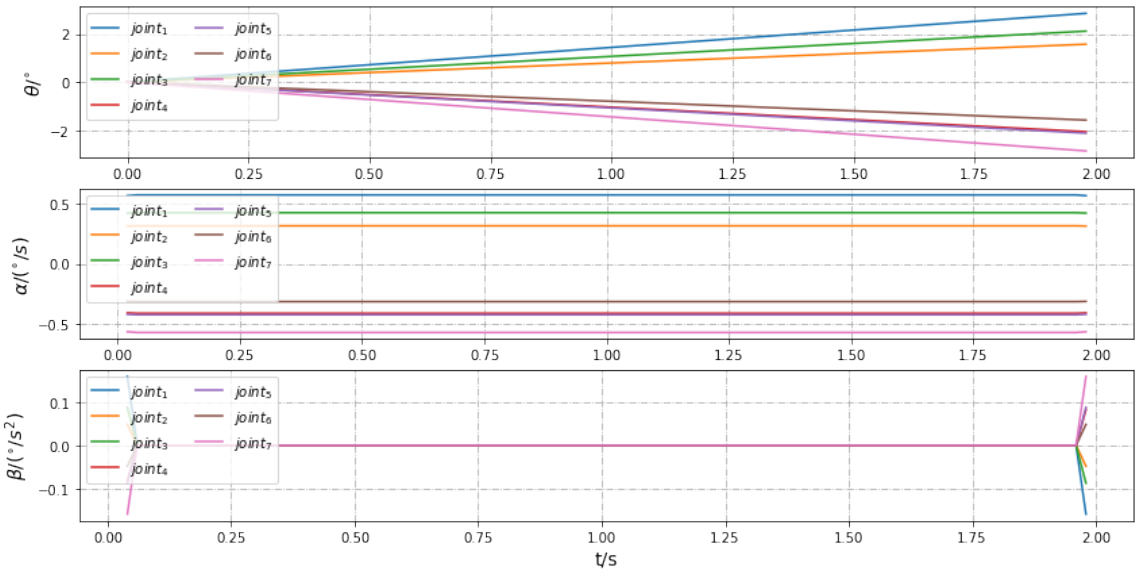


图 3: 从初始位置到达第一步

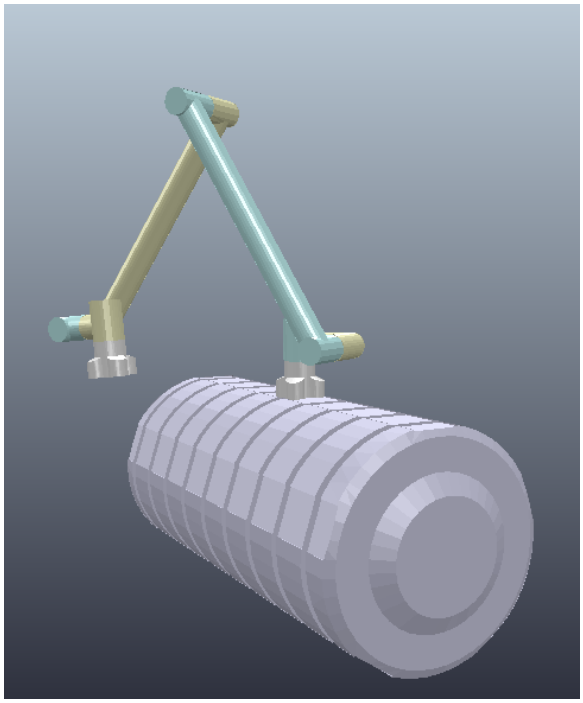


图 4: 换脚后进行第二步直行

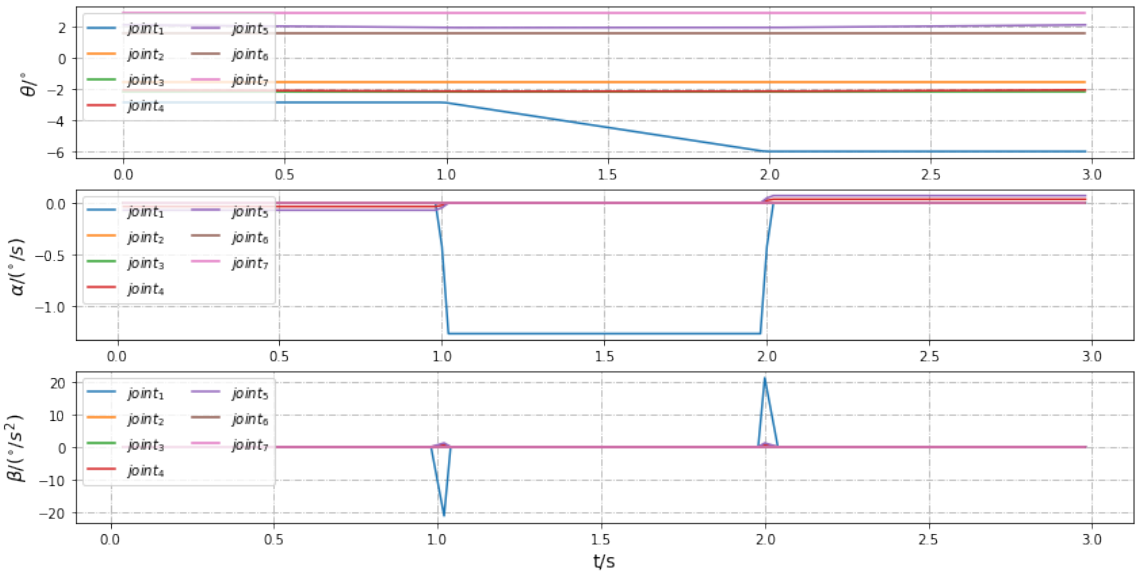


图 5: 第二步关节角变化状态

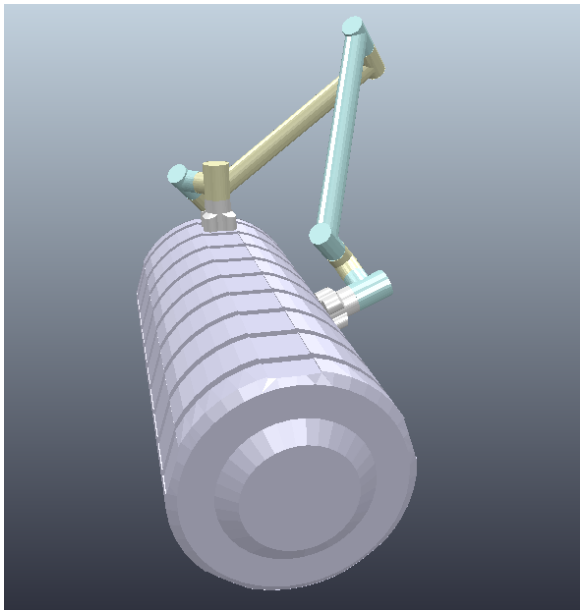


图 6: 第三步环绕舱体走斜线最终位置

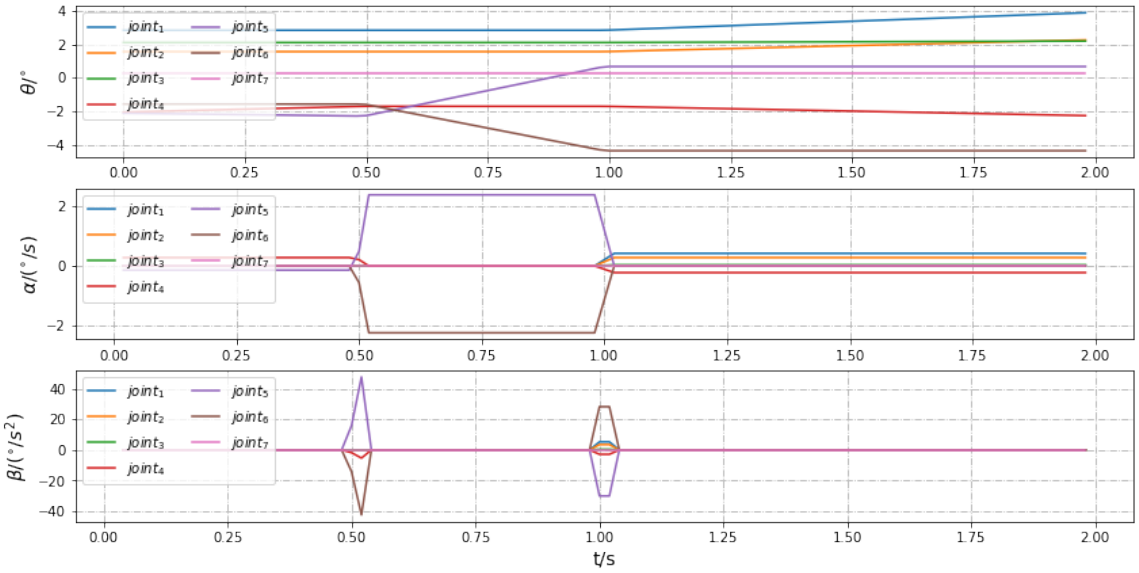
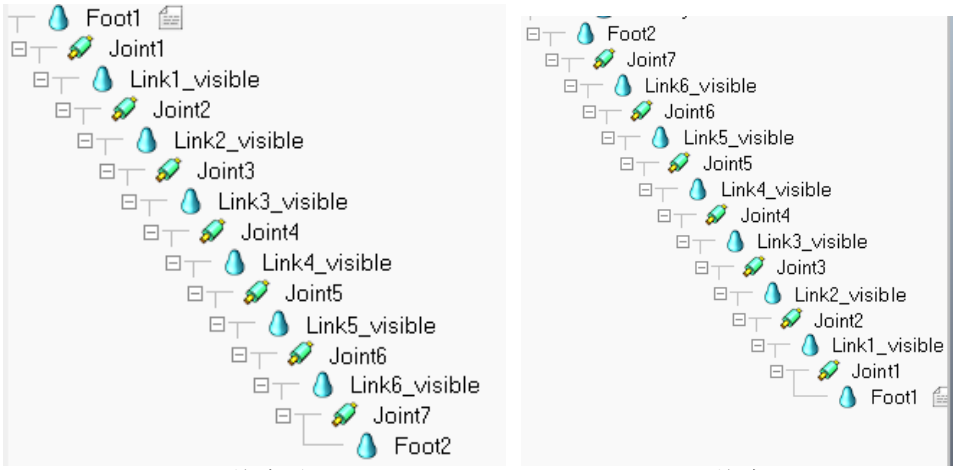


图 7: 第三步关节角变化状态

4.5　控制与换脚

在本次实验中，我们尝试采用 CoppeliaSim 的远程 ZMQRemote API 进行控制，并实现了该控制方案。

在控制层面，本次实验主要的难点在双足机器人的换脚上，由于是仿真环境，我们不太需要考虑重力等因素，而主要难点在于如何表述换脚后各个关节之间的关系。经过探索，我们通过将整个模型的树结构进行倒转（以落脚点为模型 Object Tree 的跟节点），并在每次落脚时进行根节点的转换，实现效果如下：



(a) 换脚前　　(b) 换脚后

图 8: 模型关系树

五　总结与讨论

本次实验中我们主要设计了一种能够在太空舱表面进行行走的双足机械臂，实现了走直线、走斜线、换脚等功能，同时探索了 CoppeliaSim 的远程 ZMQRemote API 的使用。另一方面，也对如何从不规则 STL 文件中重建得到三维 URDF 或 TTT 文件有了更多的了解。本次实验收获颇丰。

附录

关键代码如下所示

使用时需要将

```
C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu\programming\zmqRemoteApi\clients\python
```

中的内容复制到 control.ipynb 所在的文件夹下

```python
import time
from zmqRemoteApi import RemoteAPIClient
import numpy as np


class parabolicBlends():
    def __init__(self):
        pass
    def generateCurve(self,x,y,a):
        assert len(x)==len(y)
        # print(x)
        self.n=len(x)
```

```
13          self.x=x
14          self.y=y
15          self.abs_a=np.abs(a)
16          # number of via points
17          self.k=[0]
18          for i in range(self.n-1):
19              self.k+=[(self.y[i+1]-self.y[i])/(self.x[i+1]-self.x[i])]
20          self.k+=[0]
21          self.a=[]
22          self.b=[]
23          self.c=[]
24          self.l=[]
25          self.r=[]
26          self.intervals=[]
27          for i in range(self.n):
28
                ↪  l,r,a,b,c=parabolicBlends.getParabolic([self.x[i],self.y[i],self.k[i],self.k[i+1]],
29              self.a+=[a]
30              self.b+=[b]
31              self.c+=[c]
32              self.l+=[l]
33              self.r+=[r]
34              self.intervals+=[l,r]
35          self.intervals=np.array(self.intervals)
36          # print('x',self.x)
37          # print('y',self.y)
38          # print('k',self.k)
39          # print('intervals',self.intervals)
40      def getValue(self,x):
41          i=np.searchsorted(self.intervals,x)
42          if i==0 :return self.y[0]
43          # then the x was in the i-1~i+1
44          if i%2==0:
45              return self.y[int(i/2-1)]+(x-self.x[int(i/2-1)])*self.k[int(i/2)]
46          if i%2==1:
47              return self.a[int(i/2)]*x**2+self.b[int(i/2)]*x+self.c[int(i/2)]
48      def calculate(self,x):
49          res=[]
50          for i in range(len(x)):
51              res+=[self.getValue(x[i])]
52          return np.array(res)
```

```
53      @staticmethod
54      def getParabolic(para,a=30):
55          assert len(para)==4
56          # y=ax2+bx+c
57          # with the tangent
58          # y=k1(x-x0)+y0
59          # y=k2(x-x0)+y0
60          x0,y0,k1,k2=para
61          if (k1>k2):
62              a=-a
63          x1=(k1-k2)/4/a+x0
64          x2=(k2-k1)/4/a+x0
65          b=(k1+k2)/2-2*a*x0
66          c=y0+a*x0**2-(k1+k2)/2*x0+((k1-k2)**2/16/a)
67          # print((x1,x2,a,b,c))
68          return (x1,x2,a,b,c)
69  def LinearParabolicBlends(theta,timePoints,a=400,itersPerSec=50):
70      CureveGenerator=parabolicBlends()
71      # time[0]=0 time[-1]=2
72      theta=np.array(theta).T
73      # theta timepoints are 6*n n equals to the interval points
74      timePoints=np.array(timePoints)
75      # if timePoints.shape[0]<6:
76      #     timePoints=timePoints[0].repeat(6, axis=0)
77      # print(timePoints)
78      # time_interval=np.array(time_interval)
79      totalIters=int(itersPerSec*(timePoints[-1]-timePoints[0]))
80      dt=1/itersPerSec
81      x=np.linspace(timePoints[0],timePoints[-1],totalIters)
82      t=0
83      q=[]
84      for i in range(len(theta)):
85          # print((timePoints,theta[i],a))
86          CureveGenerator.generateCurve(timePoints,theta[i],a)
87          q+=[CureveGenerator.calculate(x)]
88      # self.control(q,totalIters,dt)
89      return q
90  class TianHe():
91      def __init__(self) -> None:
92          self.dt=0.02
93          self.client=RemoteAPIClient()
```

```
94          self.client.setStepping(True)
95          self.sim = self.client.getObject('sim')
96
97          self.joint_nums=7
98          print ('Connected to ZMQ API server')
99          # 初始化机械臂关节
100         self.retValue = []
101         self.jointHandle = []
102         # self.Foot = self.sim.getObject("/Foot1")
103         self.foot=1
104         self.root=self.sim.getObject("/Foot1")
105         while(self.sim.getObjectParent(self.root)!=-1):
106             self.root=self.sim.getObjectParent(self.root)
107         self.objectTree=[self.root]
108         while(self.objectTree[-1]!=-1):
109             self.objectTree.append(self.sim.getObjectChild(self.objectTree[-1],0))
110         self.objectTree.pop()
111         print(self.objectTree)
112         for i in range(7):
113             jointName = "/Joint" + str(i+1)
114             joint = self.sim.getObject( jointName)
115             # print(joint)
116             # self.retValue.append(ret)
117             self.jointHandle.append(joint)
118             self.shape= self.sim.getObject('/Shape')
119         self.generateCurve=LinearParabolicBlends
120         self.sim.startSimulation()
121         self.setJoint([0,0,0,0,0,0,0])
122
123         print(self.jointHandle)
124         pass
125     def setJoint(self,q):
126         for ith in range(self.joint_nums):
127             self.sim.setJointPosition(self.jointHandle[ith],q[ith])
128         pass
129     def getJoint(self):
130         q=[]
131         for ith in range(self.joint_nums):
132             q.append(self.sim.getJointPosition(self.jointHandle[ith]))
133         return np.array(q)
134         pass
```

```
135     def stop(self):
136         self.sim.stopSimulation()
137 
138     def changeFoot(self):
139         self.sim.setObjectParent(self.objectTree[-1],-1,True)
140         for i in range(len(self.objectTree)-1,0,-1):
141             self.sim.setObjectParent(self.objectTree[i-1],self.objectTree[i],True)
142             # print(self.objectTree[i-1],self.objectTree[i])
143         self.objectTree=self.objectTree[::-1]
144         self.jointHandle=self.jointHandle[::-1]
145         if(self.foot==1):
146             self.foot==2
147         else:
148             self.foot==1
149         pass
150     def control(self,q=[[0,0,0,0,0,0,0]],t=[0]):
151         qs=self.generateCurve(q,t)
152         qs=np.array(qs).T
153         print(len(qs))
154         for i in qs:
155             self.setJoint(i)
156             time.sleep(self.dt)
157         pass
158     def pace(self,d,r1,r2,theta)->list:
159         # return 7 angle
160         # inverse
161         pass
162 
163 
164 Robot=TianHe()
165 L=3
166 
167 # q1=0
168 d=np.sqrt(L**2+4*L+8)
169 # d=2+L*np.cos(q1)
170 q4=np.arcsin(d/10.4)
171 q1=np.arctan2(1,5.2*np.sin(q4/2)-1)
172 
173 q1=q1/np.pi*180+5
174 q4=q4/np.pi*180
175 print(d,q1,q4)
```

```
a1,a2,a3,a4=q1+90,90,q4+90,-180+q4*2
q1=np.array([
    [0,0,0,0,0,0,0],
    [a1,a2,a3,a4,-a3,-a2,-a1],
])
q1=q1/180*np.pi
t=[0,2]
Robot.control(q1,t)
Robot.changeFoot()
a1=a1-180
q2=np.array([
    Robot.getJoint()*(-180)/np.pi,
    Robot.getJoint()*(-180)/np.pi+np.array([0,0,0,-5,-10,0,0]),
    Robot.getJoint()*(-180)/np.pi+np.array([-180,0,0,-5,-10,0,0]),
    Robot.getJoint()*(-180)/np.pi+np.array([-180,0,0,0,0,0,0]),
    # [a1,a2,a3,a4,-a3,-a2,-a1][::-1],
])
t=[0,1,2,3]
q2=q2/180*np.pi+2*np.array(Robot.getJoint())
Robot.control(q2,t)

# time.sleep(5)
Robot.changeFoot()
joint_ori2=Robot.getJoint()
# q3=np.array([
#     # Robot.getJoint()*(-180)/np.pi,
#     [0,0,0,0,0,0,0],
#     # [60,30,10,-110,-45,-100,90],
#     joint_ori2*(-180)/np.pi+np.array([60,30,12,-110,-45,-100,-28][::-1]),
# ])
# t=[0,1]
# q3=q3/180*np.pi+joint_ori2
# Robot.control(q3,t)
# time.sleep(5)



# q3=np.array([
#     # Robot.getJoint()*(-180)/np.pi,
#     [0,0,0,0,0,0,0]+joint_ori2*(-180)/np.pi,
#     # [60,30,10,-110,-45,-100,90],
```

```
#     # joint_ori2*(-180)/np.pi+np.array([60,30,12,-110,-45,-100,360-28][::-1]),
#     # np.array([60,30,12,-110,-45,-100,-28][::-1]),
#     # [-20,-36,0,-20,0,0,0]+joint_ori2*(-180)/np.pi,
#     [0,0,0,-20,10,0,0]+joint_ori2*(-180)/np.pi,
#     [0,0,0,-20,-160,160,0]+joint_ori2*(-180)/np.pi,
#     [-60,-40,-5,10,-160,160,0]+joint_ori2*(-180)/np.pi,
#     # np.array([60,30,12,110,-45,-100,-28-90][::-1]),
# ])
# t=[0,0.5,1,2]
# q3=q3/180*np.pi
# Robot.control(q3,t)
# time.sleep(5)
# Robot.stop()

q3=np.array([
    # Robot.getJoint()*(-180)/np.pi,
    [0,0,0,0,0,0,0]+joint_ori2*(180)/np.pi,
    [0,0,0,20,-10,0,0]+joint_ori2*(180)/np.pi,
    [0,0,0,20,160,-160,0]+joint_ori2*(180)/np.pi,
    [60.542,40.878,5.210,-12.674,160.057,-160.012,-45]+joint_ori2*(180)/np.pi,
])
t=[0,0.5,1,2]
q3=q3/180*np.pi
Robot.control(q3,t)
time.sleep(5)
Robot.stop()
```