**点石·种子杯**

小组报告

**报告小组**：妈妈从今天开始我要自己上厕所

**小组成员**：李语心、张佳艺

**报告时间**：2024年11月24日

**目 录**

**1.设计思路 1**

**2.算法结构 3**

**3.核心流程 4**

**4.队员分工 6**

**参考文献 7**

**附录 源代码 8**

**1设计思路**

本次比赛，我们小组最终打采用了“先自己设计机器臂路径的基础算法，再采用强化学习在基础算法的基础上提高机器臂避障能力和路径合理性”的大体设计思路。

在了解和阅读比赛给的代码和要求后，我们首先想到的算法便是让机器臂上抬前进，从障碍物的上方去取目标物，但经过试验，在障碍物z轴较高时会与之碰撞，我们又考虑此情形下从下方绕过障碍物去取目标物，但实现困难，最终我们放弃初版方案，考虑其他情形。

从上方与下方绕过障碍物实现困难，所以我们打算从侧面绕过障碍物实现路径规划，六个关节的路径规划复杂，为了简便算法，我们将机器臂抽象成两条臂：将第三关节和第五关节固定展平，这样从x-o-y平面考虑最终姿态，两条手臂与目标物到原点的连线构成三角形，通过三角形余弦定理，臂长和目标物坐标，进而推得各个关节的最终角度。在此基础上我们根据障碍物位置，通过调整从左侧或右侧构建三角形以实现基础的避障。在大致完成x-o-y平面的定位后，我们又加上了z轴的影响，最终实现了基本算法。

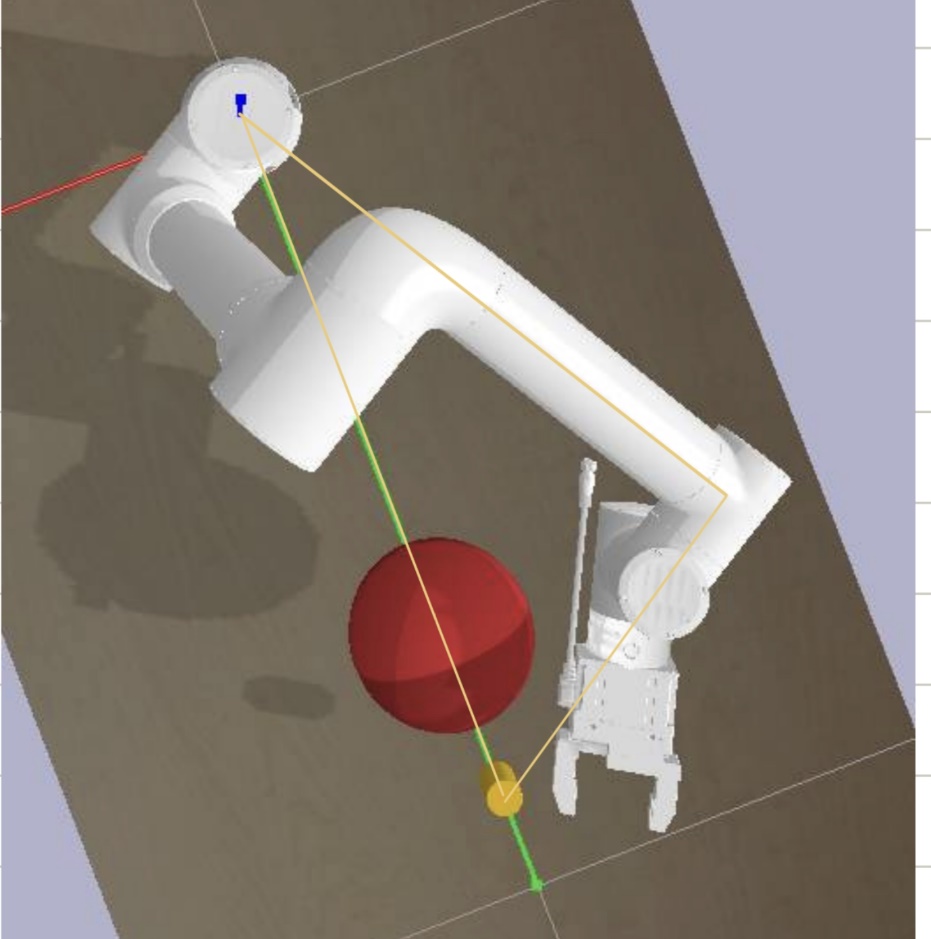
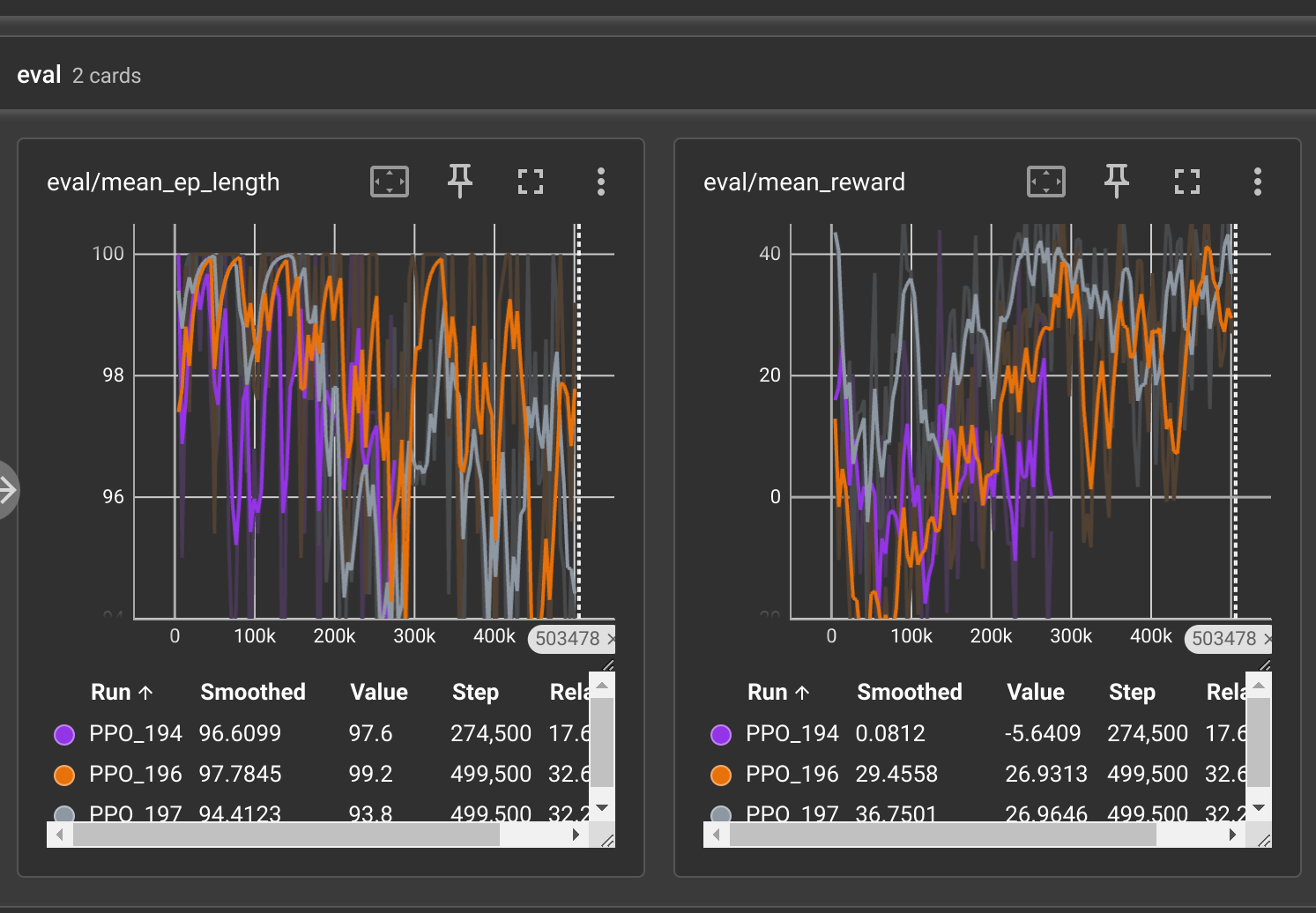
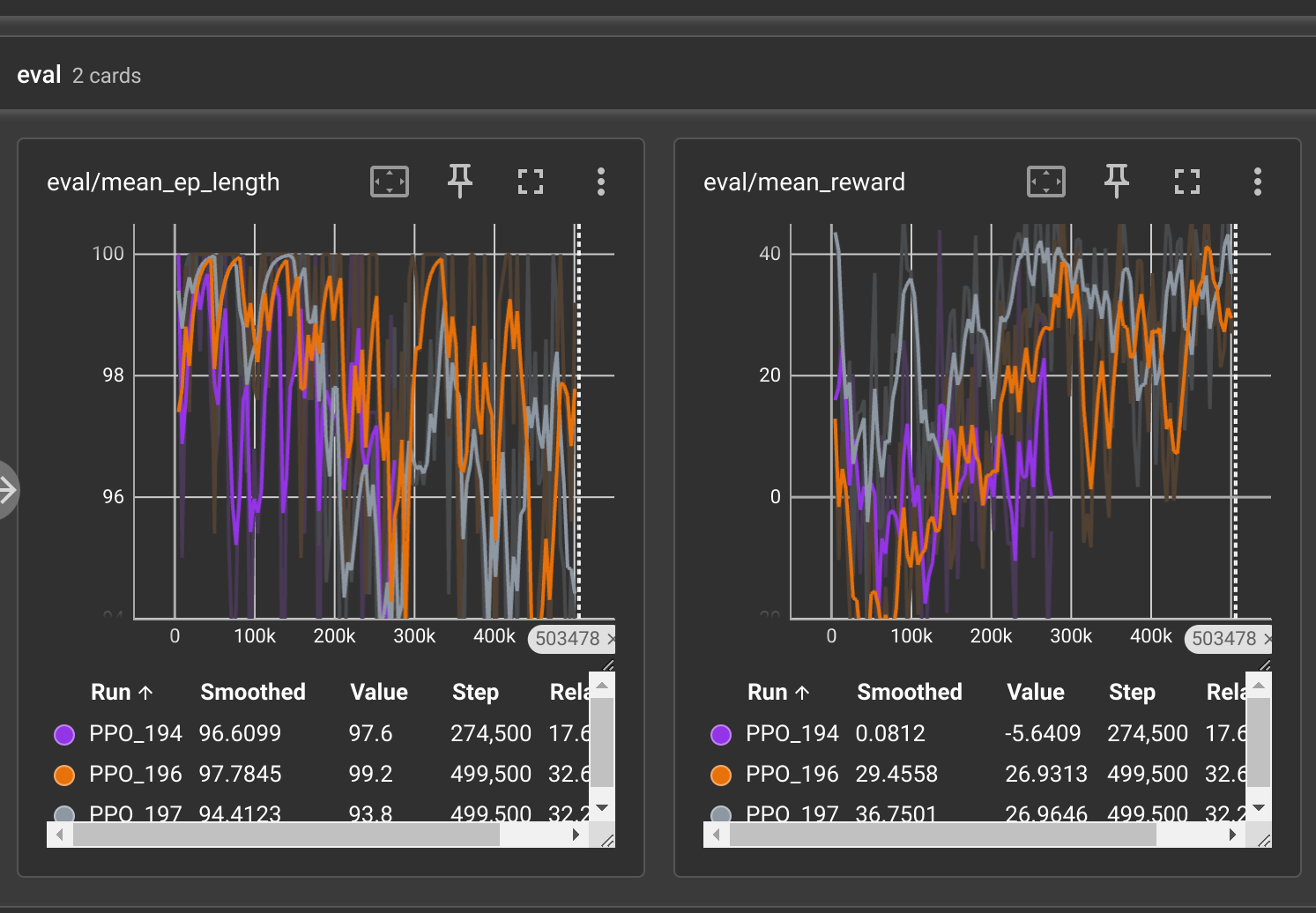


图1-1 最终姿态构建示意图

在基本算法不断的测试和调整后，我们发现在部分位置基本算法规划的路径会导致步数不够和碰障问题，最终决定采用强化学习进行进一步的完善。

我们使用 Gymnasium 和 Stable Baseline3 进行强化学习训练. 首先修改 env.py 使它符合 Gymnasium 的 env 接口. 然后用 Stable Baseline3 中的 PPO 以及 DQN 等 RL 算法进行训练. 我们修改了很多版 reward 函数和训练参数. 经过了几百次训练, 逐渐得到了较好的效果.



**2算法结构**

**1. 全局变量：**

**数组a, theta, d, alpha分别代表机器臂关节的参数，包括关节偏移量、旋转角度、平移距离和旋转轴的倾斜角度。**

**2. Calc 类：**

**\_\_init\_\_：构造函数，为空。**

**transferQue：计算齐次变换矩阵，用于描述机器臂关节的运动。**

**PositiveKine：正运动学求解，计算给定关节角度下机器臂末端爪子的位置和姿态。**

**LastPos：计算机器臂末端爪子的坐标位置。**

**WristPos：计算机器臂腕部关节的坐标位置。**

**jointPos：计算机器臂各个关节的位置。**

**collisionAngle：判断机器臂当前关节角度是否会导致碰撞。**

**near\_obs：判断机器臂各臂距离障碍物的距离。**

**idlePos：基于几何分析确定机器臂的最终姿态。**

**3. MyCustomAlgorithm 类：**

**在初始化时加载了一个预训练的模型（PPO，即Proximal Policy Optimization）。**

**get\_action：根据观测值（包括关节角度、目标位置和障碍物位置）计算并返回动作。基于观测值和Calc类计算得到的数据，通过预训练模型预测得到并返回动作。**

**3核心流程**

**在查阅相关资料并进行讨论分析后，我们确定了思路和步骤，最终实现算法的大体流程如下：**

**3.1基本避障算法的实现**

**首先，根据基坐标系的长度通过模拟图测量确定了关节2到关节4的臂长和末端臂长，后找到模型图纸修改了准确数据；**

**接着，我们在草稿纸上模拟最终机器臂抓取目标物的姿态和不同情形，推算各关节角度和目标物坐标的关系；**

**然后是算法的具体实现，我们在原test.py的基础上改造了自己的测试算法my\_test.py，放慢了机器臂行进速度，并增加了暂停和继续功能以便更好地观察并改进算法。我们根据observation传递的障碍物坐标和目标物坐标通过几何运算确定了最终关节角，并结合现关节角确定此时输出的action；**

**经过测试，基本算法可以通过40~50分，仍存在碰障和超步数等问题，需要继续改善。**

**3.2深度强化学习**

**首先，我们根据DH参数和observation传递的各关节现在角度，实现了齐次转换矩阵和坐标变换，进而通过正运动分析，推得了各关节的坐标位置，并在此基础上，通过逆运动分析求得相应姿态下各关节角度。从而用于奖励计算。**

我们 RL 模型的 obervation 如下:

self.observation = np.hstack((

            obs\_joint\_angles, # 当前关节角度

            target\_position, obstacle1\_position, # 目标位置和障碍物位置

            calc.LastPos(obs\_joint\_angles),  # 末端位置

            calc.WristPos(obs\_joint\_angles), # 手腕位置

            calc.jointPos(obs\_joint\_angles), # 其他重要关节位置

            calc.idlePos(target\_position, obstacle1\_position) # 机械臂理想位姿

        )).flatten().reshape(1, -1)

**我们的奖励大致分为以下几个部分：**

1. **与目标位姿距离**

* **机械臂当前姿态 与 通过前半部分所计算出的最终姿态 的欧几里得距离. 如果减小就奖励, 远离就惩罚.**

1. **与目标位置距离**

* **如果靠近目标就奖励, 远离就惩罚.**

1. **与障碍物距离**

* **通过 DH 参数计算出各连杆的位姿, 从而计算出机械臂和障碍物之间的最小距离. 与障碍物距离小于一定值时, 如果靠近障碍物就奖励, 远离就惩罚.**

1. **碰撞, 达到目标, 达到最大步数等**

**训练代码大致如下 ：**

class my\_log(BaseCallback):

  def \_\_init\_\_(self, verbose: int = 1):

    super().\_\_init\_\_(verbose)

  def \_on\_training\_end(self):

    return True

  def \_on\_step(self):

    my\_test.main(self.logger)

    return True

eval\_callback = EvalCallback(eval\_env,

                              best\_model\_save\_path=eval\_log\_dir,

                              log\_path=eval\_log\_dir,

                              n\_eval\_episodes=5, deterministic=True,

                              render=False, eval\_freq=1500,

                              callback\_on\_new\_best=my\_log())

if len(sys.argv) > 1:

  load\_path = sys.argv[1]

  model = PPO.load(load\_path, train\_env, verbose=1, tensorboard\_log='./tensorboard/')

else:

  model = PPO("MlpPolicy", train\_env, verbose=1, tensorboard\_log='./tensorboard/',

              batch\_size=256, n\_steps=2048, n\_epochs=10)

model.learn(500000, callback=eval\_callback)

**我们还发现左右的情况很不相同，所以我们对障碍物在目标点左右边的两种情况分别训练，并使用不同的模型。**

**4队员分工**

**李语心（组长）：** 避障算法的分析和思路确定

基本避障算法x-o-y平面的定位实现

深度强化学习

代码打包整理与提交

**张佳艺**： 基本避障算法补充z轴影响

机器臂正逆运动学分析的实现

碰撞判定的实现

报告撰写

**参考文献**

**[1] 菠萝爱吹雪.csdn,2022-08-02.**

**https://blog.csdn.net/weixin\_47849087/article/details/126121331**

**[2] liutangplease.csdn, 2023-11-22.**

**https://blog.csdn.net/m0\_53966219/article/details/125612152**

**[3] Ning.Dan .csdn, 2023-04-17.**

**https://blog.csdn.net/fqianqian96/article/details/126984441**

**[4]https://fr-documentation.readthedocs.io/zh-cn/3.6.8/CobotsManual/robot\_brief\_introduction.html#id5**

**[5] https://stable-baselines3.readthedocs.io/en/master/**

**[6] https://gymnasium.farama.org/**

**[7]https://github.com/llSourcell/Unity\_ML\_Agents/blob/master/docs/best-practices-ppo.md**

**[8]https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/**

**附录**

import numpy as np

from stable\_baselines3 import PPO

from abc import ABC, abstractmethod

import math

from time import sleep

import os

#DH参数

a=[0 , -0.425 , -0.395 ,0 , 0 , 0]

theta=[0 , 0 , 0 , 0 , 0 , 0]

d=[0.152 , 0 , 0 , 0.102 , 0.102 , 0.100]

alpha=[math.pi/2 , 0, 0, math.pi/2 , -math.pi/2 , 0]

class Calc:

    def \_\_init\_\_(self):

        pass

    def transferQue(self,i,j):      #齐次变换方程

        global a, theta, d, alpha

        c1 = math.cos(theta[j-1])

        s1 = math.sin(theta[j-1])

        c2 = math.cos(alpha[j-1])

        s2 = math.sin(alpha[j-1])

        dd = d[j-1]

        aa = a[j-1]

        A=np.array([[c1, -s1\*c2, s1\*s2, aa\*c1],

                    [s1, c1\*c2, -c1\*s2, aa\*s1],

                    [0, s2, c2, dd],

                    [0, 0, 0, 1]])

        return A

    def PositiveKine(self,n\_angle=[1, 1, 1, 1, 1, 1], i=6):   #正运动学求解

        global a, theta, d, alpha

        theta = math.pi\*(2\*n\_angle-1)

        # T1=transferQue(0,1)

        # T2=transferQue(1,2)

        # T3=transferQue(2,3)

        # T4=transferQue(3,4)

        # T5=transferQue(4,5)

        # T6=transferQue(5,6)

        # T7=T1.dot(T2).dot(T3).dot(T4).dot(T5).dot(T6)

        # print(T7)

        T = [0]\*7

        T[1] = self.transferQue(0,1)

        T[2] = T[1].dot(self.transferQue(1,2))

        T[3] = T[2].dot(self.transferQue(2,3))

        T[4] = T[3].dot(self.transferQue(3,4))

        T[5] = T[4].dot(self.transferQue(4,5))

        T[6] = T[5].dot(self.transferQue(5,6))

        return T[i]

    def LastPos(self,n\_angle):

        a = self.PositiveKine(n\_angle)

        return np.array([-a[0][3], -a[1][3], a[2][3]])

    def WristPos(self,n\_angle):

        global a, theta, d, alpha

        T=self.PositiveKine(n\_angle)

        # print(T[0][3],T[1][3],T[2][3])

        pw=np.array([-(T[0][3]-d[5]\*T[0][2]), -(T[1][3]-d[5]\*T[1][2]), T[2][3]-d[5]\*T[2][2]])

        return pw

    def jointPos(self,n\_angle): # 18

        a = []

        for i in range(1,7):

            t = self.PositiveKine(n\_angle,i)

            a = a + [-t[0][3], -t[1][3], t[2][3]]

        a = np.array(a).flatten()

        # b = (a[0:5] + a[1:6])/2

        # print("all joint pos", a, a.shape)

        return a

    def collisionAngle(self,now\_angle):         #判断是否碰撞

        theta2=now\_angle[1]

        theta3=now\_angle[2]

        l1=0.425

        l2=0.395

        R=0.1

        x=0.6

        w=0.08

        theta2L=-math.asin((2\*R+w)/2\*x)

        theta2H=math.asin((2\*R+w)/2\*x)

        m=math.sqrt((l1\*math.sin(theta2))\*\*2+(x-l1\*math.cos(theta2))\*\*2)

        theta3L=math.acos((l1\*\*2+m\*\*2-x\*\*2)/(2\*l1\*m))-math.asin((w+2\*R)/(2\*m))-math.pi

        theta3H=math.acos((l1\*\*2+m\*\*2-x\*\*2)/(2\*l1\*m))+math.asin((w+2\*R)/(2\*m))-math.pi

        if theta2<theta2H and theta2>theta2L:

            return True

        if theta3<theta3H and theta3>theta3L:

            return True

        return False

    def near\_obs(self,my\_obs, hand\_obs\_dis, if\_print=False) -> float:

        near\_r = 0.15 # param 球边离关节中心的距离

        obs = my\_obs[0]

        obs\_pos = obs[9:12]

        res = 0

        for i in range(12, len(obs)+3, 3):

            if i >= len(obs):

                dis = hand\_obs\_dis

            else:

                dis = np.linalg.norm(obs\_pos - obs[i:i+3])

            if dis < 0.1 + near\_r:

                if if\_print:

                    print(obs\_pos, obs[i:i+3])

                    print("touch obs")

                res = max(res, 0.5/(dis-near\_r))

        return res

    def idlePos(self,t, o):

        target = np.array([0.5, 0.07, 0.40097904, 0.04461199, 0.5, 0.5])

        try:

            r = 0.9

            l1 = 0.865 \* r

            l2 = 0.225 \* r

            l3 = 0.121  \* r

            l4 = 0.168 \* r

            a\_t\_n3 = -60\*(t[2]\*\*4)+51.3333\*(t[2]\*\*3)-15.65\*(t[2]\*\*2)+2.161\*t[2]-0.045

            l = np.sqrt(l1 \*\* 2 + l3 \*\* 2)

            s = np.sqrt(l \*\* 2 + l4 \*\* 2)

            l = s \* np.cos(a\_t\_n3 \* math.pi / 2)

            d = np.sqrt(t[0] \*\* 2 + t[1] \*\* 2)

            if l + l2 < d: d = l + l2 - 0.0000001

            a1 = math.acos((d\*\*2 + l\*\*2 - l2\*\*2) / (2 \* d \* l))

            a2 = np.arctan2(t[1], t[0])

            oa = np.arctan2(o[1], o[0])

            a3 = np.arctan2(l3, l1)

            a\_t\_n = 0

            a\_t\_n2 = 0

            if o[0] > t[0] - 0.1:

                if oa > a2 + 0.01:

                    a\_t = a2 - a1 + a3

                else:

                    a\_t = a1 + a2 + a3

                a\_t\_n = (a\_t / math.pi) / 2%1

                #print(a1, a2, a3)

                # action[4]

                a4 = math.acos((l\*\*2 + l2\*\*2 - d\*\*2) / (2 \* l \* l2))

                if oa > a2 + 0.01:

                    a\_t\_2 = math.pi/2+a4+a3

                else:

                    a\_t\_2 = math.pi/2-a4+a3

                a\_t\_n2 = (a\_t\_2/math.pi + 1)/2%1

            target = np.array([a\_t\_n, 0.07, 0.40097904, 0.04461199, a\_t\_n2, 0.5])

        except:

            print("error")

        return target

calc = Calc()

class BaseAlgorithm(ABC):

    @abstractmethod

    def get\_action(self, observation):

        """

        输入观测值，返回动作

        Args:

            observation: numpy array of shape (1, 12) 包含:

                - 6个关节角度 (归一化到[0,1])

                - 3个目标位置坐标

                - 3个障碍物位置坐标

        Returns:

            action: numpy array of shape (6,) 范围在[-1,1]之间

        """

        pass

class MyCustomAlgorithm(BaseAlgorithm):

    def \_\_init\_\_(self):

        path\_right = os.path.join(os.path.dirname(\_\_file\_\_), "right\_model")

        path\_left = os.path.join(os.path.dirname(\_\_file\_\_), "left\_model")

        print("ppo load path: ", path\_right, path\_left)

        sleep(1)

        self.model\_r = PPO.load(path\_right, device="cpu")

        self.model\_l = PPO.load(path\_left, device="cpu")

    def get\_action(self, observation):

        n\_angle = observation[0, :6]

        target\_position = observation[0][6:9]

        obstacle1\_position = observation[0][9:12]

        my\_obs = np.hstack((

            observation[0],

            calc.LastPos(n\_angle),

            calc.WristPos(n\_angle),

            calc.jointPos(n\_angle),

            calc.idlePos(target\_position, obstacle1\_position)

        )).reshape(1, -1)

        obs\_angle = np.arctan2(obstacle1\_position[1], obstacle1\_position[0])

        target\_angle = np.arctan2(target\_position[1], target\_position[0])

        if obs\_angle - target\_angle > 0.05:

            action, \_ = self.model\_r.predict(my\_obs)

        else:

            action, \_ = self.model\_l.predict(my\_obs)

        return np.reshape(action, (6, ))