**点石·种子杯**

小组报告（复赛）

**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12月11日

**目 录**

**1.设计思路 1**

**2.算法结构 2**

**3.核心流程 5**

**4.队员分工 6**

**参考文献 7**

**附录 源代码 8**

**1设计思路**

在复赛中，我们基于初赛的训练模型，尝试采用测算目标物移动距离，预测其100步后位置的方式来拟合初赛模型，但在测试过程中，我们发现模型很难控制100步恰好抵达目标位置，过早过晚都会使其错开，从而抓取失败。

针对这一问题，我们首先尝试了再训练一个模型end\_model来处理初始抓取失败后回退抓取的情形。我们定义了第一次尝试抓取的目标步数——94，使用初赛的left\_model和right\_model来进行第一次抓取，若抓取失败，在原来目标步数的基础上进行循环扩展，预测下一位置，训练end\_model抓取尝试抓取，若抓取失败，继续预测下一位置，直到抓取成功或达到最终位置。但在实际训练过程中，end\_model训练效果并不理想，初次抓取失败后难以再次抓取成功，且碰障较为频繁。

考虑到机器学习训练end\_model模型成效低，我们进一步优化正运动学位姿推断和碰撞判定，在初次尝试抓取失败后，改用路径规划抓取目标物。我们首先的想法是 : 对正运动学计算的抓手位置进行类似深度学习的梯度下降算法, 找到能使抓手中心位置到目标点位置的距离下降最快的角度变化方案. 但我们发现这样不能进行避障.

最终, 我们使用了A\*算法进行路径探索和规划。我们把机械臂的某一角度作为搜索的状态 S, g(S) 为从初始状态到S走过的步数 ( 其实最终与实际步数有一定差异, 因为A\*算法中相邻状态的距离是随当前状态离障碍物和目标点的距离而变化的 ), h(S) 为当前状态的抓手位置与目标点的距离除以一个参数 dis\_per\_step, 因为我们假设每步最多能将此距离缩短 dis\_per\_step.

在初步实现A\*算法后，我们发现其路径规划耗时较长，且起始模型抓取碰撞严重，针对问题，我们又逐一优化：

我们发现适当减少A\*规划搜索关节数，会大大减少搜索空间，同时对路径规划影响较小，所有我们在原有A\*算法的基础上，减少了搜索关节角的个数，同时限制了搜索步数.

我们还通过计算机器臂距离障碍物和目标物的最短距离来调整搜索半径长度，在保证避障的同时加快搜索。

又由于有时搜索半径长度超过一步所能到达的范围, 所以我们在运行过程中动态运行A\*算法. 保证在大体A\*的相邻节点之间转移时不会碰撞.

最后我们发现固定A\*的搜索目标会带来一些问题, 比如机械臂可能会错过目标. 所以我们动态变化A\*的目标点进行搜索.

在改进后梯度运算和A\*交替进行，抓取速度明显提高，我们采用A\*和梯度运算交替靠近目标物，经过不断调试预测碰撞参数，碰撞明显减少，算法得到优化。

最后我们进行大量随机测试，对某些特殊情形，作针对性调整优化。

**2算法结构**

**本组算法主要由Calc类，MyCustomAlgorithm类，a\_star方法，traditional\_get\_action方法及其所需函数组成。具体如下：**

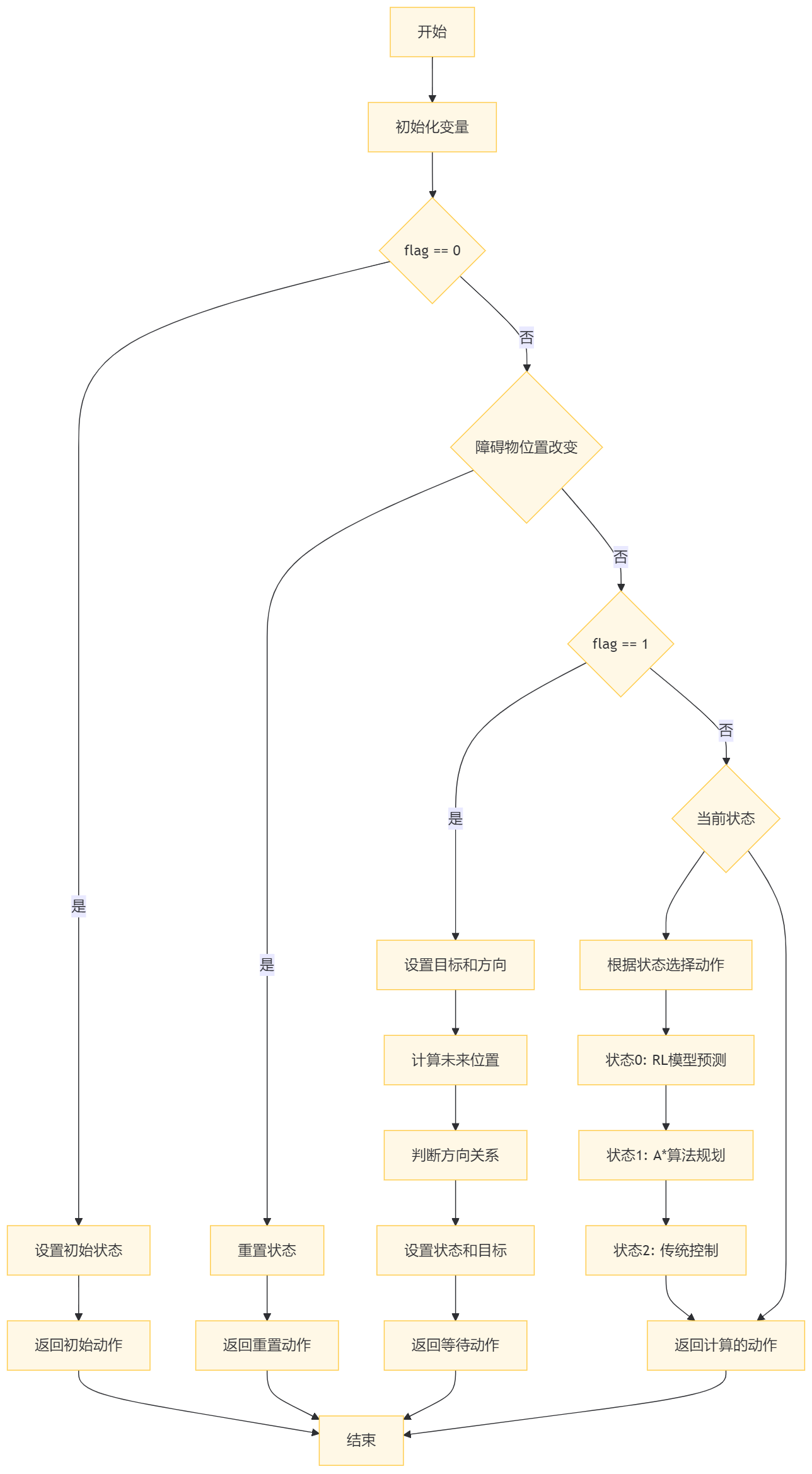
**（1）Calc类：**

**主要实现了正运动学求解末端爪子姿态位置和各关节位置，还包括了一些辅助函数，如碰撞检测和预测位置，用于A\*算法路径探索和启发函数计算。**

**（2）BaseAlgorithm和MyCustomAlgorithm类**：

**MyCustomAlgorithm**类定义了期望抓取步数——120和一些参数用于计算当前环境目标物速度以用于位置预测。

在核心算法**get\_action**中，主要由一个状态机选择当前策略并返回动作，其流程图如下：

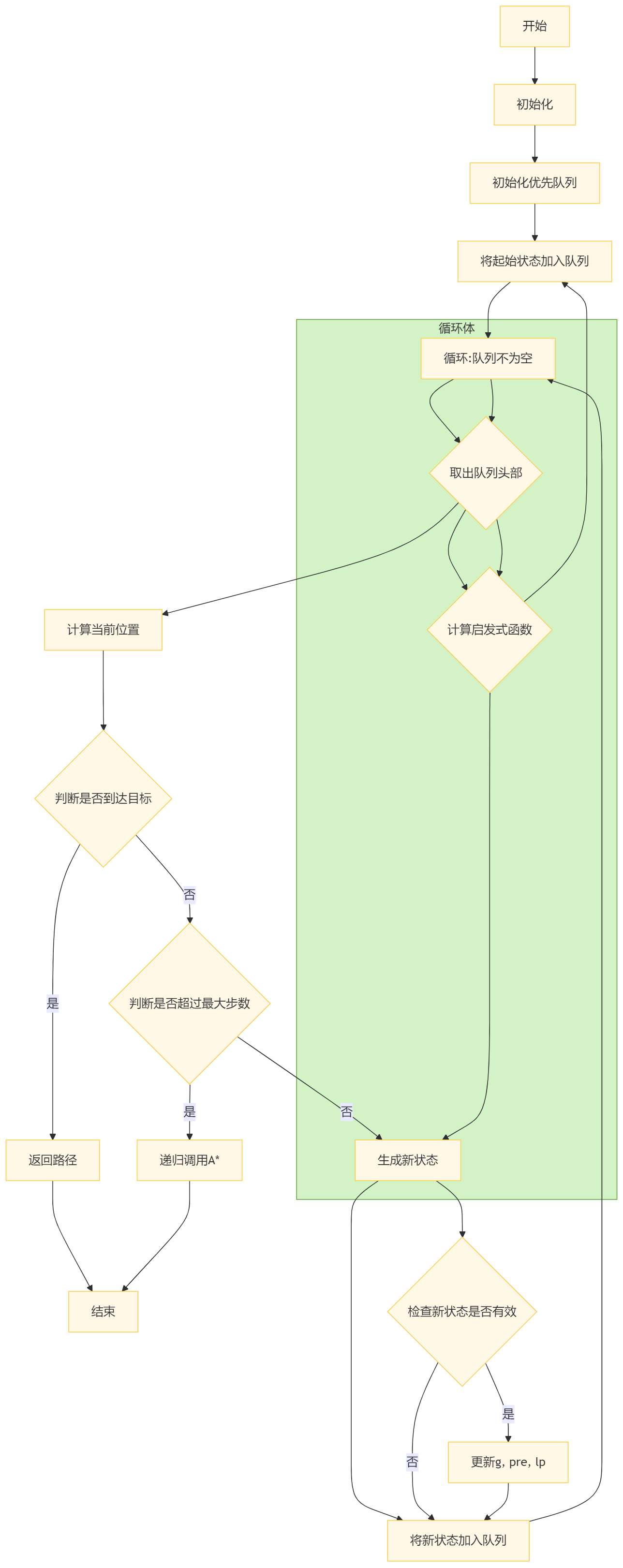


图**1-1 get-action**策略状态转换图

具体来说，采用A\*算法进行路径规划，当A\*算法将会碰撞时, 再次进行小范围A\*计算到达阶段目标点. 当A\*算法未找到路径，可能是已经碰撞时, 采用梯度算法靠近目标物，当梯度算法靠近预测会导致碰障时，再转为A\*算法进行路径规划，二者交替进行，在避障条件下向目标物逼近。

**（3）a\_star**算法:

主要由**a\_star**主体，**a\_star\_step，h，not\_valid，find\_path**等辅助函数组成，其简易流程图如下：



图**1-2 a\_star**算法流程图

首先，初始化必要的数据结构，包括优先队列q（记录待搜状态）、g（记录已找过的状态）、pre（记录每一状态的前驱节点）、lp（记忆化末端位置, 防止重复）和dis\_ob（记录当前角度下机器臂离障碍物距离），并将起始状态加入优先队列。

当待搜索状态队列不为空时，循坏进行如下操作：取出队列中具有最小启发式值的状态，计算当前状态的末端执行器位置，若当前位置与目标位置的距离小于阈值，则认为已经到达目标，用**find\_path**函数回溯路径并返回；如果步数超过最大限制，同样用**find\_path**函数回溯路径并返回；否则根据当前状态距离障碍物距离选择扩展搜索步数长度，然后广度搜索各个关节移动距离的状态，当新状态未搜索过且不会导致碰障时或搜索过但路径更短时，更新字典，并将其和g值和启发式函数值相加作为优先级，加入优先队列q中，继续循环。

当搜索半径长度过长时，路径中可能导致碰撞，则调用**a\_star\_step**进行更详细的路径规划。

当搜索路径过长时, 机械臂会错过目标点, 则改变目标点位置.

**（4）traditional\_get\_action**算法:

从观测值中提取关节角度、目标位置和障碍物位置，将关节角度转换为PyTorch张量，并设置需要梯度计算，计算末端执行器的当前位置与目标位置之间的距离，并取负值（因为我们想要最小化这个距离），如果当前距离与上一次的距离相比变化很小（可能意味着碰撞），则返回零动作。否则对距离进行反向传播，计算梯度，如果梯度有效，计算动作，即梯度的符号方向，并将梯度标准化，清除张量的梯度，为下一次计算准备，返回计算得到的动作，如果检测到可能的碰撞或梯度无效，则返回零动作。通过最小化末端执行器与目标之间的距离，迅速精确地达到目标位置。

**3核心流程**

先用动态目标的A\*算法总体规划路径. 再控制机械臂运动, 在将要碰撞时再进行小范围A\*搜索以到达阶段目标状态, 在最后使用梯度下降算法加快接近障碍物速度.

**4队员分工**

**李语心（组长）：** 强化学习环境调试及训练

A\*算法实现和优化

梯度运算优化

**张佳艺**： 强化学习训练

优化各关节坐标位置求解和碰撞预测

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

**[8]A\* : https://oi-wiki.org/search/astar/**

**[9]Adaptive A\*: https://idm-lab.org/bib/abstracts/papers/aamas05b.pdf**

**[10]torch 自动微分: https://zh.d2l.ai/chapter\_preliminaries/autograd.html**

**附录**

import numpy as np

from stable\_baselines3 import PPO

from abc import ABC, abstractmethod

import math

from time import sleep

import os

import torch

from queue import PriorityQueue

    #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)

        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):

        if not isinstance(n\_angle, np.ndarray):

            n\_angle = np.array(n\_angle)

        a = self.PositiveKine(n\_angle)

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

        return 2.5 \* pp - 1.5 \* self.WristPos(n\_angle)

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

        global a, d

        alpha = torch.tensor([torch.pi/2 , 0, 0, torch.pi/2 , -torch.pi/2 , 0])

        c1 = torch.cos(theta[j-1])

        s1 = torch.sin(theta[j-1])

        c2 = torch.cos(alpha[j-1])

        s2 = torch.sin(alpha[j-1])

        dd = torch.tensor(d[j-1])

        aa = torch.tensor(a[j-1])

        A2 = torch.tensor([[c1, -s1\*c2, s1\*s2, aa\*c1],

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

                    [0, s2, c2, dd],

                    [0, 0, 0, 1]], dtype=torch.float32)

        A = torch.mul(c1, torch.tensor([[1, 0, 0, aa],[0, c2, -s2, 0], [0, 0, 0, 0], [0, 0, 0, 0]])) + torch.mul(s1, torch.tensor([[0, -c2, s2, 0], [1, 0, 0, aa], [0, 0, 0, 0], [0, 0, 0, 0]])) + torch.tensor([[0, 0, 0, 0], [0, 0, 0, 0], [0, s2, c2, dd], [0, 0, 0, 1]])

        return A

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

        global a, d, alpha

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

        T = [0]\*7

        T[1] = self.transferQue\_torch(0, 1, theta)

        T[2] = torch.mm(T[1], self.transferQue\_torch(1,2, theta))

        T[3] = torch.mm(T[2], self.transferQue\_torch(2,3, theta))

        T[4] = torch.mm(T[3], self.transferQue\_torch(3,4, theta))

        T[5] = torch.mm(T[4], self.transferQue\_torch(4,5, theta))

        T[6] = torch.mm(T[5], self.transferQue\_torch(5,6, theta))

        return T[i]

    def LastPos\_torch(self, n\_angle):

        global d

        T = self.PositiveKine\_torch(n\_angle)

        T = T.t()

        pp = torch.mul(T[3][0:3], torch.tensor([-1, -1, 1]))

        return pp + 1.5 \* torch.mul(torch.mul(T[2][0:3], d[5]),torch.tensor([-1, -1, 1]))

    def WristPos(self,n\_angle):

        global a, theta, d, alpha

        T=self.PositiveKine(n\_angle)

        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 WristPos\_torch(self,n\_angle):

        global a, theta, d, alpha

        T=self.PositiveKine\_torch(n\_angle)

        T = T.t()

        pw = torch.mul(T[3][0:3], torch.tensor([-1, -1, 1])) - torch.mul(T[2][0:3], d[5])

        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()

        return a

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

        near\_r = 0.1 # param 球边离关节中心的距离

        near\_r2 = 0.15

        obs = my\_obs[0]

        obs\_pos = obs[9:12]

        res = 0

        for i in range(12, 39, 3):

            if i >= 36:

                dis = hand\_obs\_dis

            else:

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

            if dis < 0.1 + near\_r2:

                if if\_print:

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

                    print("touch obs")

                res = min(res, dis)

        return res

    def idlePos(self,t, o):

        # action[0]

        r = 0.9

        l1 = 0.865 \* r

        l2 = 0.225 \* r

        l3 = 0.121  \* r

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

        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

        # 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

        # action[5] && z

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

        return np.array([a\_t\_n, a\_t\_n3, 0.40097904, 0.04461199 - a\_t\_n3 + 0.07, a\_t\_n2, 0.5])

    def disjo(self,j1,j2,ob):

        AP = ob - j1

        AB = j2 - j1

        t = np.dot(AP,AB)/np.dot(AB,AB)

        t = max(0, min(1,t))

        Q = j1 + t\*AB

        d=np.linalg.norm(ob - Q)

        return d

    def collisionAngle(self,n\_angle,ob, dis\_ob=None):         #判断是否碰撞

        global theta

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

        T1= self.transferQue(0,1)

        T2 = T1.dot(self.transferQue(1,2))

        T3 = T2.dot(self.transferQue(2,3))

        T4 = T3.dot(self.transferQue(3,4))

        T5 = T4.dot(self.transferQue(4,5))

        T6 = T5.dot(self.transferQue(5,6))

        p1=np.array([-T1[0][3], -T1[1][3], T1[2][3]])

        p2=np.array([-T2[0][3], -T2[1][3], T2[2][3]])

        p3=np.array([-T3[0][3], -T3[1][3], T3[2][3]])

        p4=np.array([-T4[0][3], -T4[1][3], T4[2][3]])

        p5=np.array([-T5[0][3], -T5[1][3], T5[2][3]])

        p6=np.array([-T6[0][3], -T6[1][3], T6[2][3]])

        p8 = self.LastPos(n\_angle)

        p9 = np.array([-0.138\*math.sin(theta[0]), 0.138\*math.cos(theta[0]), p1[2]])

        p7=np.array([p2[0]+p9[0]-p1[0], p2[1]+p9[1]-p1[1], p2[2]+p9[2]-p1[2]])

        d1=self.disjo(p9,p7,ob)

        d2=self.disjo(p7,p2,ob)

        d3=self.disjo(p2,p3,ob)

        d4=self.disjo(p3,p4,ob)

        d5=self.disjo(p4,p5,ob)

        d6=self.disjo(p5,p6,ob)

        d7=self.disjo(p6,p8,ob)

        d8=np.linalg.norm(p3 - ob)

        d9=np.linalg.norm(p4 - ob)

        d10=np.linalg.norm(p5 - ob)

        d11=np.linalg.norm(p8 - ob)

        d12=np.linalg.norm(p7 - ob)

        if min(d1,d2) < 0.15 or d3 < 0.165:

            return True

        if min(d4, d5, d6) < 0.155:

            return True

        if min(d9, d12) < 0.169 or d8 < 0.18:

            return True

        if d7 < 0.18:

            # print("&",d7)

            return True

        if min(d10, d11) < 0.178:

            return True

        if d6 < 0.165:

            return True

        if min(p1[2], p2[2], p3[2], p4[2], p5[2], p6[2], p7[2], p8[2], p9[2]) < 0.025:

            return True

        if dis\_ob != None:

            dis\_ob[tuple(n\_angle)] = min(min(d1,d2,d3) - 0.15, min(d4, d5, d6) - 0.155, min(d8, d9, d12) - 0.169, d7 - 0.18, min(d10, d11) - 0.178, d6 - 0.165, min(p1[2], p2[2], p3[2], p4[2], p5[2], p6[2], p7[2], p8[2], p9[2]) - 0.015)

        return False

calc = Calc()

def predict\_pos(now, v, step):

  v = np.array(v)

  if v.shape[0] != 3:

    v = np.array([v[0], 0, v[1]])

  t = np.array(now) + v \* step / 12

  if t[0] > 0.5:

    t[0] = 1 - t[0]

  if t[0] < -0.5:

    t[0] = -1 - t[0]

  if t[2] > 0.5:

    t[2] = 1 - t[2]

  if t[2] < 0.1:

    t[2] = 0.2 - t[2]

  return t

# pos1 在 pos2 的哪个方向, 只看x和y坐标

#  o

# / \

# 2  \

#    1

# 1 在 2 的右边(right)

# right = 1, left = -1, center = 0

def relative\_dir(pos1, pos2, use\_int=False):

    angle1 = np.arctan2(pos1['y'], pos1['x'])

    angle2 = np.arctan2(pos2['y'], pos2['x'])

    if abs(angle1 - angle2) < 0.1:

        return 'center' if not use\_int else 0

    elif angle1 - angle2 > 0:

        return 'right' if not use\_int else 1

    elif angle1 - angle2 < 0:

        return 'left' if not use\_int else -1

    return 'center'

def next\_tar\_step(now, tar1, max):

  if now <= tar1:

    return tar1 - now

  return min(max, now + 1) - now

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.zip")

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

        sleep(1)

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

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

        self.flag = 0

        self.vx = 0

        self.vz =0

        self.target = [0, 0, 0]

        self.num = 0

        self.n\_obs = [0, 0, 0]

        self.max\_steps = 200

        self.target\_step = 120 # 未来目标位置的步数

    # v : [vx, vz]

    def get\_future\_pos(self, now\_tar\_pos, v, step):

        step = max(0, step - self.num)

        # print('step:', step, self.num)

        now\_tar\_pos = now\_tar\_pos.copy()

        v = v.copy()

        for \_ in range(step):

            now\_tar\_pos[0] += v[0]

            now\_tar\_pos[1] += v[1]

            if now\_tar\_pos[0] > 0.5 or now\_tar\_pos[0] < -0.5:

                v[0] = -v[0]

            if now\_tar\_pos[2] > 0.5 or now\_tar\_pos[2] < 0.1:

                v[1] = -v[1]

        # print(now\_tar\_pos)

        return now\_tar\_pos

    def get\_action(self, observation):

        self.set\_ball\_pos = observation[1]

        observation = observation[0]

        n\_angle = observation[0, :6]

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

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

        # print(self.num,target\_position)

        self.num += 1

        if self.flag == 0:

            self.n\_obs = obstacle1\_position

            self.st = target\_position

            self.target = target\_position

            self.flag = 1

            self.num = 0

            self.last\_dis = -1

            return np.array([0, 0, 0, 0, 0, 0])

        if self.n\_obs[0] != obstacle1\_position[0]:

            self.n\_obs = obstacle1\_position

            self.st = target\_position

            self.target = target\_position

            self.flag = 0

            self.num = 0

            self.last\_dis = -1

            return np.array([0, 0, 0, 0, 0, 0])

        if self.flag == 1:

            # init

            self.vx = target\_position[0]-self.st[0]

            self.vz = target\_position[2]-self.st[2]

            self.end\_tar = predict\_pos(self.st, [self.vx \* 12, self.vz \* 12], self.max\_steps)

            self.target = predict\_pos(self.st, [self.vx \* 12, self.vz \* 12], self.target\_step)

            self.path = []

            self.dir\_future = relative\_dir(

                {'x': self.target[0], 'y': self.target[2]},

                {'x': obstacle1\_position[0], 'y': obstacle1\_position[1]},

                True

            )

            self.dir\_end = relative\_dir(

                {'x': self.end\_tar[0], 'y': self.end\_tar[2]},

                {'x': obstacle1\_position[0], 'y': obstacle1\_position[1]},

                True

            )

            if self.dir\_end \* self.dir\_future < -0.5:

                self.target\_step = 170

            print("dir", self.dir\_future, self.dir\_end)

            self.now\_state = 1 # 0: RL, 1: A\*, 2: torch

            self.init\_state = 1

            self.flag = -1

            return np.array([0, 0, 0, 0, 0, 0])

        if self.flag == -1:

            real\_target = self.target

            real\_dir = self.dir\_future

            my\_obs = np.hstack((

                n\_angle, real\_target, obstacle1\_position,

                calc.LastPos(n\_angle),

                calc.WristPos(n\_angle),

                calc.jointPos(n\_angle),

                calc.idlePos(real\_target, obstacle1\_position),

                [self.vx \* 12, 0, self.vz \* 12],

                [self.dir\_future, self.dir\_end],

            )).reshape(1, -1)

            my\_obs2 = np.hstack((

                n\_angle, 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])

            action = [0] \* 6

            change\_target\_step = self.target\_step

            a\_star\_target\_step = self.target\_step + 25

if self.now\_state == 1:

                action\_tri = np.array(self.traditional\_get\_action(my\_obs2))

                if np.linalg.norm(calc.LastPos(n\_angle) - predict\_pos(target\_position, [self.vx \* 12, self.vz \* 12], 1)) <= 0.05 and not calc.collisionAngle(n\_angle + action\_tri / 360, obstacle1\_position):

                    print('use last triditional')

                    return action\_tri

                if not self.init\_state and np.linalg.norm(self.path[self.now\_tar\_idx] - n\_angle) < 0.003:

                    self.now\_tar\_idx = self.now\_tar\_idx + 1

                    if self.now\_tar\_idx >= len(self.path):

                        self.now\_state = 1

                        self.init\_state = 1

                if self.init\_state:

                    self.init\_state = 0

                    print('start A\*', self.num)

                    real\_target = predict\_pos(target\_position, [self.vx \* 12, self.vz \* 12], max(1, self.target\_step - self.num))

                    next\_target = predict\_pos(target\_position, [self.vx \* 12, self.vz \* 12], max(6, self.target\_step - self.num + 20))

                    self.path = self.a\_star(n\_angle, real\_target, obstacle1\_position, nxt=next\_target, mxstep=max(1, self.target\_step - self.num))

                    self.now\_tar\_idx = 0

                if self.path == None: # go to traditional

                    print('no path, go to traditional', self.num)

                    self.now\_state = 2

                    self.init\_state = 1

                    return np.zeros(6)

                action = np.array(self.path[self.now\_tar\_idx] - n\_angle) \* 360 / max(np.max(np.abs(self.path[self.now\_tar\_idx] - n\_angle)) \* 360, 1)

                if calc.collisionAngle(n\_angle + action / 360, obstacle1\_position):

                    now\_target = self.path[self.now\_tar\_idx]

                    self.path = self.a\_star\_step(n\_angle, now\_target, obstacle1\_position, 1/360) + self.path[self.now\_tar\_idx:]

                    self.now\_tar\_idx = 1

                    if self.path == None:

                        print('no path, go to traditional', self.num)

                        self.now\_state = 2

                        self.init\_state = 1

                        return np.zeros(6)

                    action = np.array(self.path[self.now\_tar\_idx] - n\_angle) \* 360 / max(np.max(np.abs(self.path[self.now\_tar\_idx] - n\_angle)) \* 360, 1)

            elif self.now\_state == 2:

                if self.init\_state:

                    self.init\_state = 0

                    print('traditional')

                action = self.traditional\_get\_action(my\_obs2)

                if calc.collisionAngle(n\_angle + np.array(action) / 360, obstacle1\_position):

                    print('traditional collision')

                    self.now\_state = 1

                    self.init\_state = 1

                    return np.zeros(6)

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

    def traditional\_get\_action(self, observation):

        n\_angle = observation[0, :6]

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

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

        a = [0] \* 6

        torch.enable\_grad()

        n\_angle\_tensor = torch.tensor(n\_angle, dtype=torch.float32, requires\_grad=True)

        dis = - torch.sum( (calc.LastPos\_torch(n\_angle\_tensor) - torch.tensor(target\_position)) \*\* 2 )

        if self.last\_dis > 0 and dis - self.last\_dis >= 1e-4: # maybe collision

            return torch.zeros(6)

        dis.backward()

        if n\_angle\_tensor.grad == None or torch.any(torch.isnan(n\_angle\_tensor.grad)):

            return torch.zeros(6)

        with torch.no\_grad():

            action = n\_angle\_tensor.grad / torch.max(torch.abs(n\_angle\_tensor.grad))

        n\_angle\_tensor.grad.zero\_()

        return action

    def a\_star(self, start, tarpos, obspos, nxt=None, mxstep=1000):

        """

        Args:

            start: list of 6 : angles

            tarpos: list of 3 : position

        Space:

            360 ^ 6

        """

        start = np.array(start)

        q = PriorityQueue()

        g = {} # g[ang] = 已走过的步数

        pre = {} # pre[ang] = 上一个状态

        lp = {} # lp[ang] = lastpos(ang)

        dis\_ob = {}

        g[tuple(start)] = 0

        pre[tuple(start)] = None

        lp[tuple(start)] = calc.LastPos(start)

        calc.collisionAngle(start, obspos, dis\_ob)

        print('start', start, dis\_ob.get(tuple(start), -1))

        total\_cnt = 0

        q.put((self.h(lp[tuple(start)], tarpos),) + tuple(start))

        while not q.empty():

            cur = q.get()[1:7]

            if tuple(cur) not in lp:

                lp[tuple(cur)] = calc.LastPos(cur)

            curpos = lp[tuple(cur)]

            dis = np.linalg.norm(curpos - tarpos)

            now\_dis\_ob = dis\_ob.get(tuple(cur), -1)

            total\_cnt += 1

            if dis < 0.05 or (total\_cnt > 30 and dis < 0.12):

                return self.find\_path(pre, cur)

            if total\_cnt > 50:

                return self.find\_path(pre, cur)

            if min(dis, now\_dis\_ob) > 0.3:

                step\_len = 0.02

            elif now\_dis\_ob < 0.05 or dis < 0.15:

                step\_len = 0.01

            elif dis < 0.07:

                step\_len = 1/360

            elif now\_dis\_ob < 0.01:

                step\_len = 1/360/5

            else:

                step\_len = 0.01

            new\_g = g[tuple(cur)] + 1

            if new\_g > mxstep:

                print('next')

                return self.a\_star(start, nxt, obspos)

            i = [0] \* 6

            for i[0] in range(-1, 2):

                for i[1] in range(-1, 2):

                    for i[2] in range(-1, 2):

                        for i[4] in range(-1, 2):

                            new\_ang = np.array([cur[j] + i[j] \* step\_len for j in range(6)])

                            if tuple(new\_ang) == tuple(cur):

                                continue

                            if tuple(new\_ang) not in g and self.not\_valid(new\_ang, obspos, dis\_ob):

                                continue

                            if tuple(new\_ang) not in g or new\_g < g[tuple(new\_ang)]:

                                g[tuple(new\_ang)] = new\_g

                                pre[tuple(new\_ang)] = cur

                                if tuple(new\_ang) not in lp:

                                    lp[tuple(new\_ang)] = calc.LastPos(new\_ang)

                                q.put((new\_g + self.h(lp[tuple(new\_ang)], tarpos),) + tuple(new\_ang))

        return None

    def a\_star\_step(self, start, tarang, obspos, minsteplen = 100):

        """

        Args:

            start: list of 6 : angles

            tarpos: list of 6 : target\_angle

        Space:

            360 ^ 6

        """

        def h(a, b):

            a = np.array(a)

            b = np.array(b)

            return np.max(np.abs(a - b))

        start = np.array(start)

        q = PriorityQueue()

        g = {} # g[ang] = 已走过的步数

        pre = {} # pre[ang] = 上一个状态

        dis\_ob = {}

        g[tuple(start)] = 0

        pre[tuple(start)] = None

        calc.collisionAngle(start, obspos, dis\_ob)

        print('start', start, dis\_ob.get(tuple(start), -1))

        total\_cnt = 0

        q.put((h(start, tarang),) + tuple(start))

        while not q.empty():

            cur = q.get()[1:7]

            now\_dis\_ob = dis\_ob.get(tuple(cur), -1)

            total\_cnt += 1

            if h(start, tarang) <= 1 or total\_cnt > 50:

                return self.find\_path(pre, cur)

            step\_len = minsteplen

            if now\_dis\_ob < 0.01:

                step\_len = 1/360/3

            new\_g = g[tuple(cur)] + 1

            i = [0] \* 6

            for i[0] in range(-1, 2):

                for i[1] in range(-1, 2):

                    for i[2] in range(-1, 2):

                        for i[4] in range(-1, 2):

                            new\_ang = np.array([cur[j] + i[j] \* step\_len for j in range(6)])

                            if tuple(new\_ang) == tuple(cur):

                                continue

                            if tuple(new\_ang) not in g and self.not\_valid(new\_ang, obspos, dis\_ob):

                                continue

                            if tuple(new\_ang) not in g or new\_g < g[tuple(new\_ang)]:

                                g[tuple(new\_ang)] = new\_g

                                pre[tuple(new\_ang)] = cur

                                q.put((new\_g + h(new\_ang, tarang),) + tuple(new\_ang))

        return None

    def h(self, lastpos, tarpos):

        dis\_per\_step = 0.007

        return np.linalg.norm(lastpos - tarpos) / dis\_per\_step

    def not\_valid(self, ang, obspos, dis\_ob):

        if not isinstance(ang, np.ndarray):

            ang = np.array(ang)

        return calc.collisionAngle(ang, obspos, dis\_ob)

    def find\_path(self, premap, end):

        path = []

        while end != None:

            path.append(end)

            end = premap[end]

        return path[::-1]