**多机器人协同原理与技术作业报告**

**基于PSO和人工势场法的多机器人搜索Multi robot search based on PSO and artificial potential field method**

学 院（系）： 电子信息与电气工程

专 业： 人工智能

学 生 姓 名： 马艺鸣

学 号： 201981187

大连理工大学

Dalian University of Technology

# 摘 要

计算机自出现以来便开始快速发展，尤其是进入21世纪后，随着相关学科和技术的突飞猛进，计算机也更加迅速的发展了起来，而人类对计算机的利用也发展到了各个领域，多机器人系统便是其中的一个重要领域。多机器人系统通过交流协作和分享信息改进了单个机器人的性能，如任务执行效率、健壮性、灵活性和容错性，同时涵盖了[分布式](https://so.csdn.net/so/search?q=%E5%88%86%E5%B8%83%E5%BC%8F&spm=1001.2101.3001.7020)决策，编队控制、区域覆盖及其相关应用。本次实验通过利用粒子群算法、人工势场法、沿墙法等算法，模拟完成了一个多机器人协同探索的问题，并在最终取得了不错的效果。

**关键词：多机器人；粒子群算法；人工势场法；沿墙法**

**目录**

[摘 要 2](#_Toc99448598)

[1 任务描述与算法描述 4](#_Toc99448599)

[1.1 任务描述 4](#_Toc99448600)

[1.2 算法描述 4](#_Toc99448601)

[2 算法流程设计与关键代码解析 7](#_Toc99448602)

[2.1 算法流程设计 7](#_Toc99448603)

[2.2 关键代码解析 9](#_Toc99448604)

[3 算法评测 11](#_Toc99448605)

[4 不足与展望 13](#_Toc99448606)

[结 论 15](#_Toc99448607)

[附录A 部分重要代码 16](#_Toc99448608)

# 1 任务描述与算法描述

## 1.1 任务描述

假设存在一个150\*150的空间，存在一个目标点（130，130）。现有10个检测半径为50的机器人，分布在如图1.1所示的左下角扇形区域，扇形半径为65。当机器人间的距离大于50时，两个机器人不能直接通信。在空间中除了机器人和目标之外，还存在障碍物，障碍物中心在（110，110），半径为5。

任务要求为，设计一个算法，让机器人群在不碰到障碍物的前提下快速地搜索到目标点。

图表

低可信度描述已自动生成

图1.1 机器人任务图

## 1.2 算法描述

在本篇研究报告中，核心算法包括粒子群算法(Particle swarm optimizatio, PSO)、人工势场法(Artificial Potential Field Method).

粒子群算法是多智能体优化中的经典算法，由J. Kennedy和R. Eberhart在论文” Particle swarm optimization”中提出。该算法通过初始化一系列点，并不断迭代求解。对于个粒子的第次更新，其更新公式如下：



其中，为第个粒子第次迭代的位置，为速度，为第个粒子截止第次的最优历史位置，为截止为第次更新全部粒子的最优历史位置。为惯性因子，为一个0到1的随机数。

该算法的效果图如下：

图示

描述已自动生成

图1.2 粒子群算法效果图

（上图源于<https://blog.csdn.net/daaikuaichuan/article/details/81382794>）

但是在实际的搜索过程中，空间中往往会存在障碍物，这时就需要在搜索算法的基础上加入避障算法。

在本次实验中，我使用的是人工势场法，该算法原理如下：

我们利用势函数来建立人工势场，该势函数是一种可微函数，空间中某点的势函数值大小，代表了该店的势场强度。最简单的势函数是引力、斥力势函数。其作用是让目标对机器人产生吸引力，障碍物对机器人产生斥力。某一点的势函数可以表达为：



最常见的引力势函数为：



其中，为引力系数，表示当目标点的距离。

最常见的斥力势函数如下：



其中，表示点到最近障碍物的距离，表示斥力系数，表示斥力作用的最小距离。

如果把某点 处的势函数的取值 看作该点的能量大小，那么梯度 则可以看作该点的力向量，其定义为：



可以看出，某点处梯度的方向即为势函数增长最快的方向，更新速度时需用这个公式。

有了上述两个理论算法，便可以开始设计本次任务了。

# 2 算法流程设计与关键代码解析

## 2.1 算法流程设计

本次任务的两大关键点，一个是机器人，即粒子群中的粒子如何构建，另一个是粒子群算法和人工势场法如何对粒子群中的粒子位置进行更新。

由于机器人在一开始的时候不知道目标点在哪，所以会进行一些初始化的搜索，当机器人直接或者间接的得知目标点之后，会进行更精确地搜索。

在不考虑障碍的情况下，我们可以将算法流程概括为如下的流程图：

图示

描述已自动生成

图2.1 不考虑障碍物时的算法流程图

因为初始化时我们不知道目标点何在，所以需要对地图进行盲目式探索。在本次实验中，为了能够让机器人能够检测到最大的搜索范围，我的初始化搜索方法为让机器人的初始速度均匀分散在90°上，并且对PSO算法的适应度函数设置为距离远点越远越好。

通过上述设计，我们便可以让粒子尽可能的去搜索到更多的范围。但是在搜索的过程中，所有粒子不可能一直在同一个粒子群中，他们会分散为多个小的独立的粒子群。

通过建立一个10\*10关系表记录每个粒子之间的距离是否小于通信距离，便可以得到每个分散的小粒子群。

当某个粒子群搜索到目标点时，该粒子群中所有个体的适应度函数会进行更改，变化为距离目标点的距离值。每个粒子速度的更新也会结合人工势场法进行修正。因此，该粒子群会倾向于向目标点接近。也就是上图所示的精确搜索。当粒子群到达目标点后，搜索结束。

当然，搜索的过程中每时每刻都可能会会遇到障碍物，当遇到障碍物时，需要对粒子速度进行如下修改：

图示

描述已自动生成

图2.2 遇到障碍时的测量

其中，人工势场法即向粒子施加一个反方向的力，让粒子获得一个反方向的速度增量。沿墙法则是对粒子增加一个障碍物切线方向的速度增量，让粒子绕过障碍物。通过人工势场法和沿墙法相结合，可以做到在尽量远离障碍物的前提下绕过障碍这一目的。

将上述两个流程进行结合，便可以得到本次实验的整体流程。从代码的角度来说，上述流程的关键步骤是更新粒子速度、更新粒子位置、判断粒子是否离群、判断粒子与哪些粒子属于一个小粒子群。

## 2.2 关键代码解析

上述流程只是对整个流程的总结，在具体实现的过程中涉及到了很多细节。总的来说，本次实验中主要利用到了两个类，一个是粒子类，一个是算法类。

粒子类中包含了粒子速度、位置、历史轨迹、自身最优位置和最佳适应度，所在群里最优位置和最佳适应度、是否知道目标点、是否到达目标点等属性及对应方法。

算法类中包含了粒子关系表、对单个粒子速度更新方法、对单个粒子位置更新方法等。

下边将会对其中的重要部分进行解析。下方代码顺序与附录A中代码顺序相同。

（1）适应度函数设计

由于对于粒子而言，一开始并不知道具体的目标点位置，所以一开始要进行一些随机搜索。而当粒子得知目标点位置之后，会利用目标点位置进行速度位置更新。所以，对于一个粒子，他的适应度函数需要符合粒子对应的情况。

在本次实验中，我对这两个阶段的设计思想为：当粒子不知道目标点位置时，适应度函数需驱使粒子远离原点，并在此基础上增加一些随机性。当粒子知道目标点之后，粒子越接近目标点，适应度函数越小。

（2）速度初始化

为了更好的探索更大的范围，在初始化机器人粒子的速度时我将粒子的速度在角度上进行了平均分配。该过程结合了旋转矩阵rotate，若想对速度向量逆时针旋转，只需对向量右乘下述矩阵：



（3）粒子间关系表建立

因为机器人本身存在通信范围，所以在搜索的过程中难免会出现某些机器人通信不上的情况，这个时候粒子群就会分为多个小粒子群。互不相容的粒子群之间的群体最优点是不能共享的。

因为存在上述情况，所以在每次粒子更新完位置之后，对粒子间的关系进行判断。我通过计算两两粒子间的距离，若小于50（即通信范围），则将表中对应位置设置为1，若大于50，则设置为0.

因为算法类中更新速度和位置都需要利用上述表，所以该关系表十分重要。

（4）更新速度

最简单的更新速度就是直接利用PSO算法结合人工势场法进行更新，但是因为存在随即搜索和精确搜索两个阶段，以及遇到障碍物的情况，所以速度的更新需要在PSO的基础上加入多个约束。

两个阶段利用到的适应度函数与（1）中所述对应，当遇到障碍物时，我在人工势场法的基础上增加了沿墙法的思想，也就是绕着障碍物走。这个过程中同样利用到了旋转矩阵，只需对粒子施加一个切线方向的速度即可。

（5）更新位置

更新位置本身的操作非常简单，在之前的基础上加上更新速度即可。但是在更新完之后需对各种情况进行判断。

首先第一个是更新关系表，然后判断是否找到目标点，之后基于关系表对各粒子的属性进行更新。在最后判断如果达到了结束标准，则该粒子静止不动。

# 3 算法评测

本次评测首先对基本的任务进行评测，之后在此基础上增加多个障碍物进行难度更高的测试，测试效果如下：

（1）基本任务

图表, 散点图

描述已自动生成

图3.1 基本任务动态展示图

图表

描述已自动生成

图3.2 上述过程中最快的路径

（2）多个障碍物

图表, 散点图

描述已自动生成

图3.3 多障碍物的情况

图片包含 散点图

描述已自动生成

图3.4 最佳路径

# 4 不足与展望

在三中所示的两种任务下，算法都能达到不错的效果，但是在测试的过程中发现还会存在一些问题，包括粒子丢失、粒子卡死在某一个地方等情况，如下图所示：

图表, 散点图

描述已自动生成

图4.1 粒子丢失情况

可以发现，最后在右下角有一个粒子丢失了。

图表, 散点图

描述已自动生成

图4.2 粒子卡死的情况

由于沿墙法、人工势场法、PSO等多种算法都会对粒子的速度进行约束，所以在某些特殊的情况下，粒子会提前处于某个平衡下。也就是处于卡死的情况。

综上所述，算法本身还是存在缺陷，但是从大体结果上来看还是不错的，这为接下来进一步学习打下了良好的基础。

结 论

通过本次实验，我对多机器人协同有了进一步的理解。通过对多智能体协同方法进行编程实现，加强了对算法本身的理解和思考。在实现PSO、人工势场法、沿墙法的过程中，我对这些方法如何结合到一起进行了设计，并最终取得了不错的效果。

但是，本次实验并不是完美的，最终的算法在某些情况下会出现缺陷，所以仍有改进空间。

附录A 部分重要代码

（1）fitness设计

def fitness\_function\_unknown(x):

return -np.sum(5\* np.power(x, 2) + np.power(x - 10 \* np.random.rand(1, 2)[0], 2))

def fitness\_function\_known(x):

return np.sum(np.power(x-destination, 2))

（2）速度初始化

def init\_v(self, dim, num):

        #这里只假设了dim=2的情况，其他情况需将rotate扩展

        PI = 3.1415626535

        angle = PI/2/(num -1)

        rotate = np.array([[np.cos(angle), -np.sin(angle)],

                           [np.sin(angle), np.cos(angle)]])

        vs = [np.array([0, 1])]

        for i in range(num-1):

            vs.append(np.matmul(vs[-1], rotate))

        vs = np.array(vs)

        return vs

（3）关系表建立

self.team\_relation = np.ones((num, num))

 def calc\_relation(self):

        for i,p1 in enumerate(self.Particle\_team):

            for j,p2 in enumerate(self.Particle\_team):

                #距离小于一个值

                if self.calc\_distance(p1.get\_pos(), p2.get\_pos()) < 50:

                    self.team\_relation[i][j] = 1

                else:

                    self.team\_relation[i][j] = 0

（4）速度更新

#更新速度

def update\_v(self, part):

    if part.is\_single:

        v = (self.W \* part.get\_v() + (self.C1 \* np.random.rand() \* (part.get\_pbest() - part.get\_pos()))).squeeze() + np.random.rand(1, 2)[0]\*0.1

    else:

        v = (self.W \* part.get\_v() +  self.C1 \* np.random.rand() \* (part.get\_pbest() - part.get\_pos()) + self.C2 \* np.random.rand() \* (part.get\_gbest() - part.get\_pos())).squeeze() + np.random.rand(1, 2)[0]\*0.1

    if np.sqrt(np.sum(np.power(part.get\_pos()-destination, 2))) < 50:

        v += (destination - part.get\_pos())

        #如果在障碍附近，就绕着走

    for i in range(hurdle\_num):

        if np.sqrt(np.sum(np.power(part.get\_pos()-hurdle[i], 2))) < 15 and (part.get\_pos()[1] < hurdle[i][1]):

            v -= v #归零

            PI = 3.1415626535

            angle = PI/2

            rotate = np.array([[np.cos(angle), -np.sin(angle)],

                                [np.sin(angle), np.cos(angle)]])

        v += np.matmul((hurdle[i] - part.get\_pos()), rotate)\*0.1 + (part.get\_pos()-hurdle[i])\*0.0001

        #如果马上要碰到了

        if np.sqrt(np.sum(np.power(part.get\_pos()-hurdle[i], 2))) < 8:

            v += part.get\_pos() - hurdle[i]

        #如果已经距离目标点十分近了，就直接去就行

        if np.sqrt(np.sum(np.power(part.get\_pos()-hurdle[i], 2))) < 15 and (part.get\_pos()[1] < hurdle[i][1]) and np.sqrt(np.sum(np.power(part.get\_pos()-destination, 2))) < 10:

            v = (destination - part.get\_pos())

#不能太大

    max\_sub = self.v\_max - v

    for i, sub\_v in enumerate(max\_sub):

        if sub\_v < 0:

            v[i] = self.v\_max[i]

    #也不能太小

     min\_sub = self.v\_min - v

     for i, sub\_v in enumerate(min\_sub):

        if sub\_v > 0:

            v[i] = self.v\_min[i]

     if -0.1 < v[0] - v[1] < 0.1:

           v[0] = v[0] \* 1.1

    #更新该粒子速度

    part.set\_v(v)

（5）更新位置

    def update\_pos(self, part, number):

        #更新位置

        part.set\_pos(part.get\_pos() + part.get\_v())

        #更新位置表

        self.calc\_relation()

        #判断是否更新该粒子的历史最优点, 评测函数越小说明越接近目标，由unknow变为know

        if self.calc\_distance(part.get\_pos(), self.destination) < 50:

            #将这个信息传递给组中

            self.fitness\_value\_init(number)

            self.trans\_destination\_hurdle(number)

        val = part.calc\_fitness\_value()

        #如果粒子是单个粒子漫游,只更新自己的best即可

        if part.single(number) == 1:

            if val < part.get\_bestfitness\_value():

                #更新最佳评测函数

                part.set\_bestfitness\_value(val)

                #更新粒子最佳历史点

                part.set\_pbest(part.get\_pos())

        else:

            if val < part.get\_bestfitness\_value():

                #更新最佳评测函数

                part.set\_bestfitness\_value(val)

                #更新粒子最佳历史点

                part.set\_pbest(part.get\_pos())

            if val < self.get\_gbestfitness\_value\_in\_team(number):

                #更新群体最佳评测

                #更新种群历史最佳点

                self.update\_gbestfitness\_in\_team(number, val, part.get\_pos())

        #判断是否不需要进一步更新

        if val < part.over\_time and part.is\_know:

            part.is\_over()