# 多智能体系统分布式包围控制总结

**项目名称**：多智能体系统分布式包围控制

**团队成员**：孙玉斐、李浩杨、丁宁

**参考文献**：《多智能体系统分布式包围控制》，作者：段敏

**一阶多智能体协同分布式包围控制**

**问题描述：**

假设有n个智能体和n个目标的多智能体系统，每个智能体与一个目标相对应，每个智能体只能获取与之对应的目标状态信息。目标是可能移动的，任意时刻t第i个智能体可以检测到自身的位置yi(t)，也可知道第i个目标的位置ri(t), 但不能获取其他目标的位置信息。目标也可能是静止的，即实现静止目标的包围。所有目标的平均中心（目标横纵坐标的均值）为P

每个智能体看作是无向连通通信拓扑图G中的一个节点，每条边 (sj,si) 对应个体 j 到 i 可进行通信的信息链，具有边连通的两个节点称为邻接（如下图2中蓝色线段所示）。每个智能体通过自身状态和邻接智能体状态信息更新自身当前状态。

**实现目标**：

符合下述目标：

（1）每一智能体与目标中心P的距离是ρ的k倍（ρ定义为所有目标距离目标中心P的最大值）

（2）圆形编队：所有智能体按角度平均分布于一个圆周上（假如有四个智能体对应四个目标，那么最终智能体呈现圆形包围目标时，任意两个邻近的智能体之间的角度差正好为Π/2，即为圆周2Π的1/4）

**问题模型：**

智能体与智能体之间的通信拓扑结构是事先给的。例如图2中四个智能体的通信拓扑结构。有边的代表两个智能体之间可以进行通讯，无边的就不能够进行通讯。另外，公式（1.5）中的aij，拉普拉斯矩阵中的值，也体现了这点。

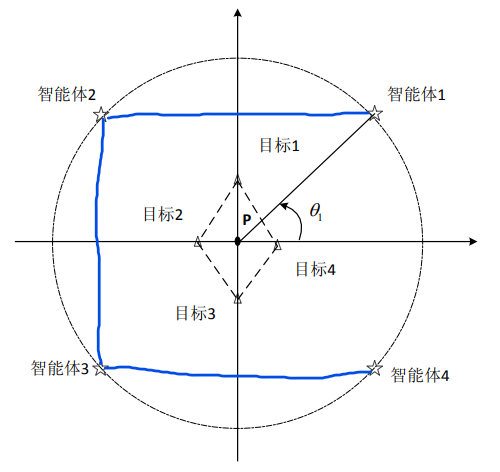


图1 基于圆形编队的分布式包围控制模型

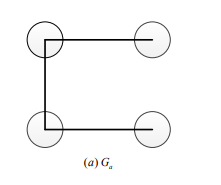


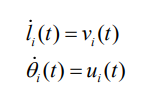
图2 智能体通讯拓扑结构

假设多智能体系统中每个智能体满足以下动态方程：

 （1.1）

其中yi(t)表示第i个智能体（i=1……n）的位置状态，（y对于t的函数上加一点）表示位置y对于时间t的导函数，ui(t) 表示第i个智能体的控制输入，按照物理移动模型，ui(t)作为控制输入实际上表示的是速度（智能体船向速度，这里不考虑水流速度）。

为了便于圆形编队的分析，将问题转为化为以目标中心P为原点的极坐标下的角度和半径的问题。极坐标可以转换回二维坐标。后面通过修改角度和半径的值来修改船的位置。具体式子如下图：

（1.2）

其中i ∈1,2,……,n，li（t）、θi（t）分别表示极坐标下第 i 个智能体以目标平均中心 P 为坐标原点的坐标半径和角度。式（1.2）中的表示半径l对于时间t的导数，表示角度θ对于时间t的导数

第i个智能体的位置信息可以计算为

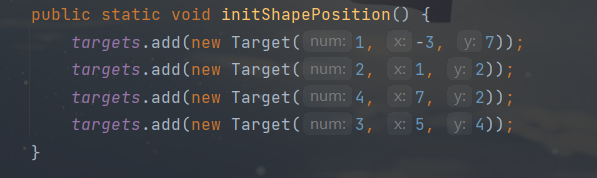
（1.3）

其中pi（t）表示第i个智能体对目标平均中心 P 的估计值。

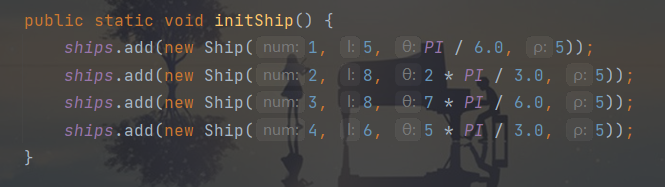
右边代表第i个智能体对于目标平均中心P的估计值与第i个智能体的半径在x轴方向的分量和y轴方向的分量的和。在进行代码实现的时候，pi（t）的值用包含坐标信息的纵向矩阵[x,y]T进行表示，这样有利于公式的计算。

**算法步骤：**

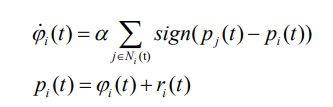
1. **根据目标来确定目标的中心位置：**

这里给出计算目标中心的方法。首先初始化赋值目标的编号和位置信息，还有船的信息：

船的信息包括船的编号（num），船距离目标中心的半径长度（l），船与目标中心的连线与x轴的夹角（θ），ρ是智能体对（目标距离目标平均中心的最大距离）的估计值，ρ通过初值赋值后续通过式子调整。



**计算目标中心位置的式子：**

（1.4）

其中 i ∈1,2,……,n，ϕi（t）表示目标平均中心动态估计器的内部状态，且 ϕi(0)= 0。

Ni(t)表示能够能与智能体i进行直接通信的智能体的集合（如上图2中智能体2可以同时与智能体1、智能体3进行通信，那么N2(t)就包括智能体1与智能体3）

Pi(t)表示的是智能体i对目标中心的位置的估计值，即每一艘船都有一个对目标中心的估计值。ri(t)表示与智能体i对应的目标的位置，按假设，智能体i可以获得ri(t)的实际值，ɑ是给定的常数。

对于（1.4）式，Pi(t)存在的意义在于假如目标是移动的，那么智能体需要实时获取目标中心的位置（目标位置变化导致目标中心位置改变）用来对半径以及角度等参数进行修正，因此每个智能体对于目标中心的位置需要有一个实时的估计，而估计值为目标此时的位置ri(t)与目标平均中心动态估计器的内部状态ϕi(t)之和。

ϕi(t)可以让目标通过加上这个值使结果落在目标中心的位置，在代码中将ϕi(t)也分为x方向和y方向，可以直接与目标的坐标ri(t)进行加减，它的初值为0。

下举一数值实例来展示ϕi（t）的作用：

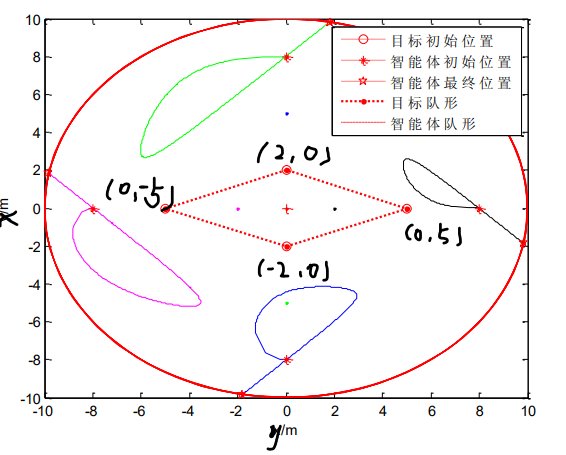
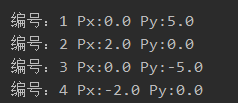


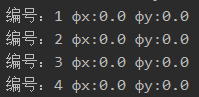
图3 静止目标包围

智能体与目标在0时刻与最后时刻的位置状态信息如图3所示，此时目标平均中心的位置为（0,0）。

按照问题描述，0时刻时每个智能体只有当前时刻对应目标的位置信息，因此对于每个智能体的Pi（t）来说都与目标的初始位置相同：



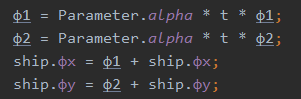
而0时刻每个智能体的内部状态器ϕi（0）= 0:



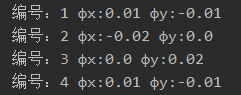
按照式（1.4）的处理，每个智能体通过与邻接智能体的信息交流并改变内部状态器ϕi（t）的值，这里拿智能体1（编号1）举例：按照图1的通信拓扑结构，智能体1只与智能体2进行直接通讯，因此按照式（1.4）中Ni(t)的定义，智能体1的ϕ(t)只与智能体2进行增量计算：



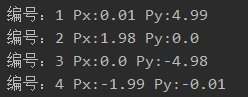
这里计算的只是ϕ(t)的导数，需要对该计算结果进行积分操作：



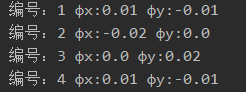
因为P1(0)对应的坐标为（0,5），P2(0)对应的坐标为（2,0），α取值为10，因此经过计算智能体1的x增量为1，y增量为-1，每一时刻取的值为0.001，再乘以增量与α得到智能体1的ϕx=0.01，ϕy=-0.01，其他智能体计算类似：

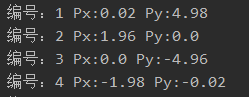


而修改的值通过与目标的位置信息进行相加便会逐渐向目标中心位置（坐标点为（0,0））靠近：

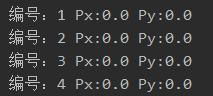


再进行一个时刻的变化后，因为P1(0.001)对应的坐标为（0.01,4.99），P2(0.001)对应的坐标为（1.98,0），α取值为10，因此经过计算智能体1的x增量为1，y增量为-1，每一时刻取的值为0.001，再乘以增量与α得到智能体1的ϕx=0.01，ϕy=-0.01，其他智能体计算类似：

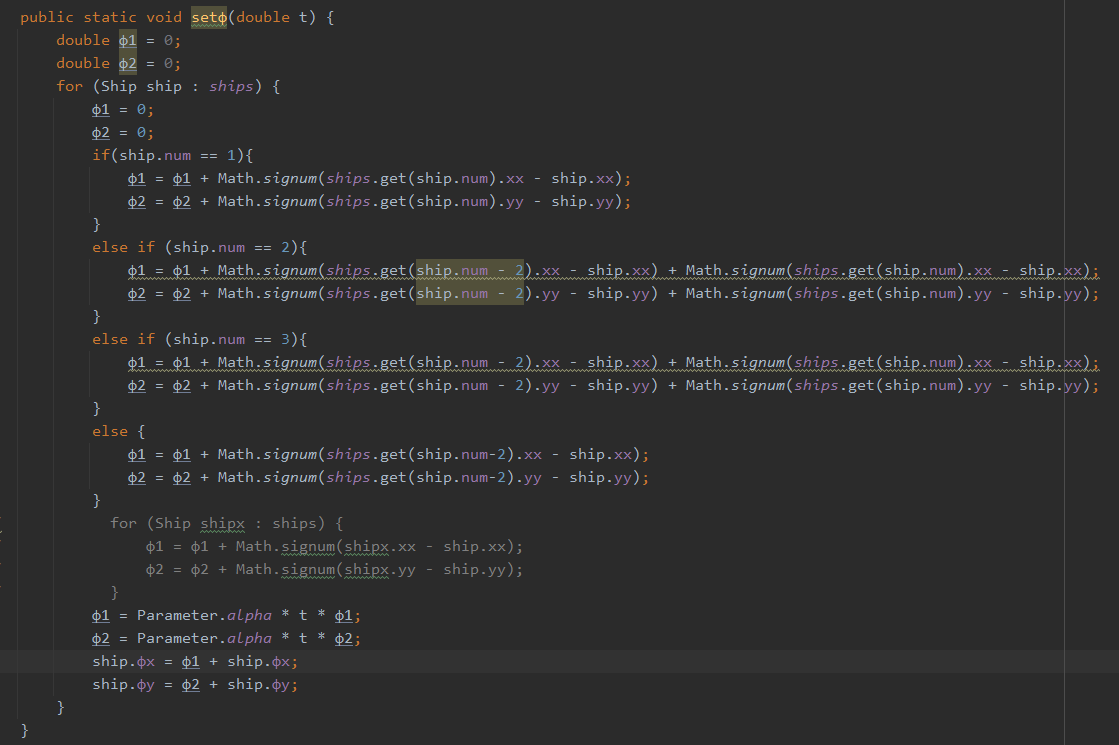


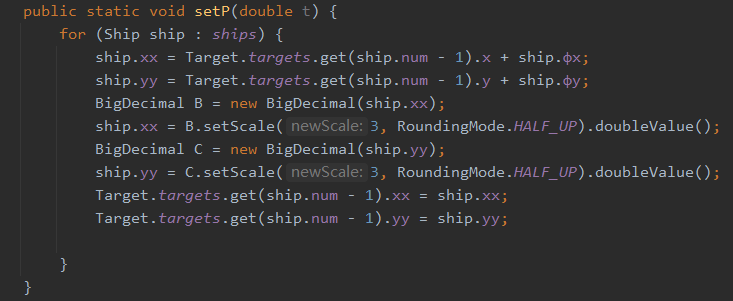


这样在每一时刻不停获取内部状态信息会使智能体对目标中心的估计值逐渐靠近，最终在某一时刻做到估计值与实际值相同：



下面的代码展示了如何对ϕi（t）及pi（t）进行计算：





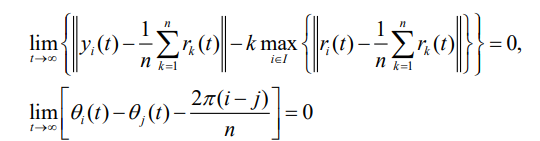
Setϕ中，前两行定义了两个值分为代表ϕ在x方向上的导数和在y方向上的导数，中间的循环过程是按照通讯图进行的计算，比如智能体1只能与智能体2进行计算，加入了时间t这个变量，计算的是导数的值乘以时间就是这个值的增量。最后两行就是得到新的ϕ值。

SetP中，ship.xx表示的是目标中心x方向的估计值。由于一个智能体对应一个目标，如图2所示，编号为1的目标与智能体1相对应，前两行是根据式子ship.xx等于船对应的目标的（Target.targets.get(ship.num - 1).x）x坐标加上刚刚计算出来的ϕi（t）的x方向上的值，ship.yy也同理得到，后面四行将这两个值取三位小数，最后两行把这两个值给Target的xx,yy。这样方便计算目标距离目标中心的距离。

首先利用当前时刻ϕi（t）导数的值会由上一时刻变换而来求出当前时刻的ϕi（t）再加上此时ri（t）便是pi(t)的值，代码中用ship的xx与yy属性表示每个智能体对于目标中心坐标的估计值。

在计算的时候，因为r(t)等式子的表示都是x,y这样来进行表示，所以将代码中将式子也按照x,y来进行单独的计算。

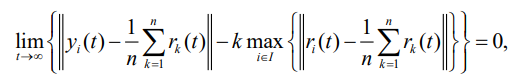
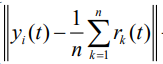
**2.停止条件：**

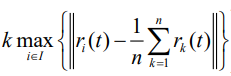


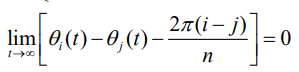
**式(1.7)**

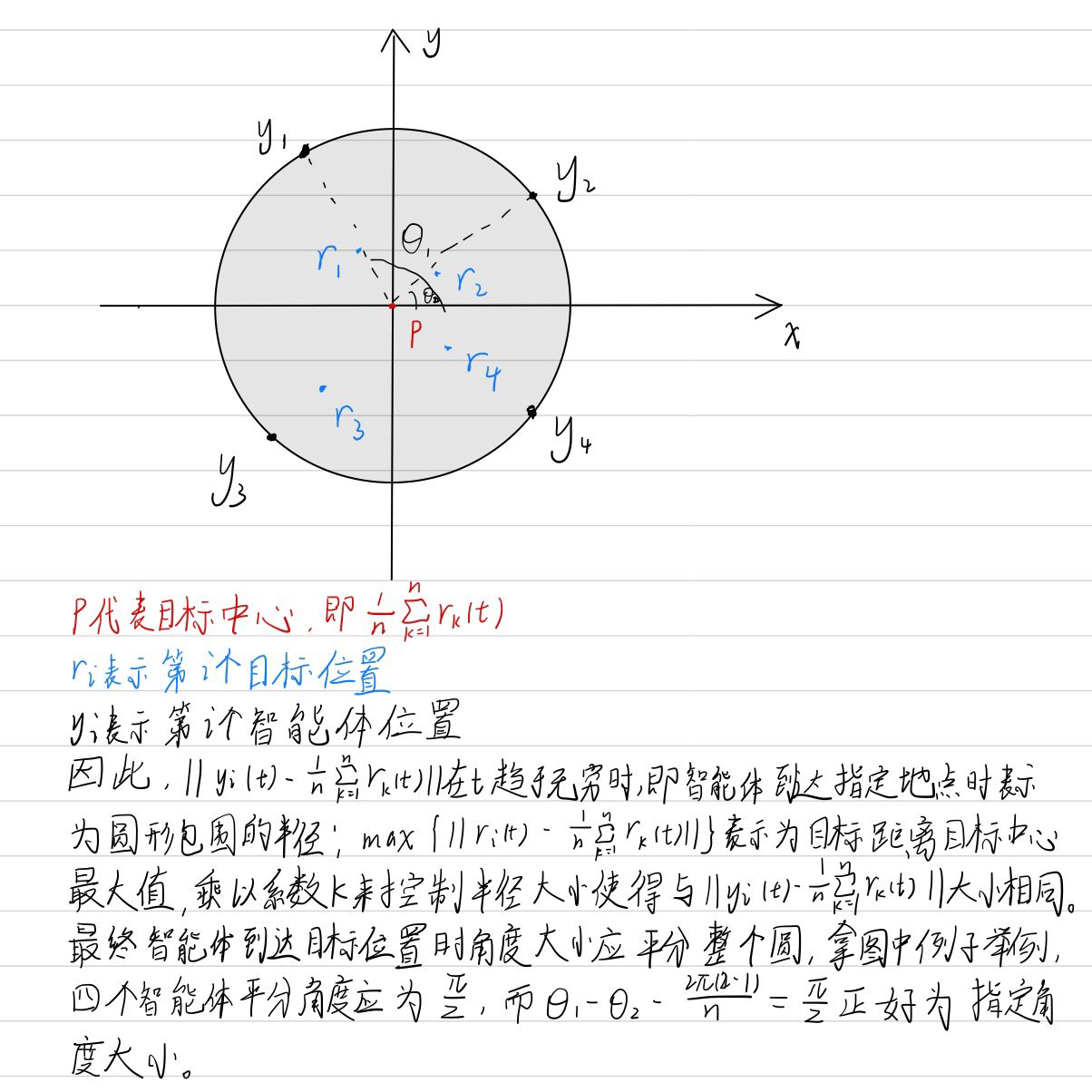
其中 ri(t)表示第 i 个目标的位置状态，ri(t)∈ R2，k表示多智能体系统编队半径增益，且 k>1。

上面的一条式子是对半径停止的判断，下面一条是对角度停止的判断。对于第一条式子前半部分表示智能体距离目标中心的距离，后半部分表示的是编队的半径。

对于式，式表示第i个智能体的位置与目标中心位置的距离，式

表示目标距离目标中心距离位置的最大值，乘以一个系数k来控制半径的大小，即最终停止时智能体的半径长为目标距离目标中心距离位置的最大值乘以一个系数k。

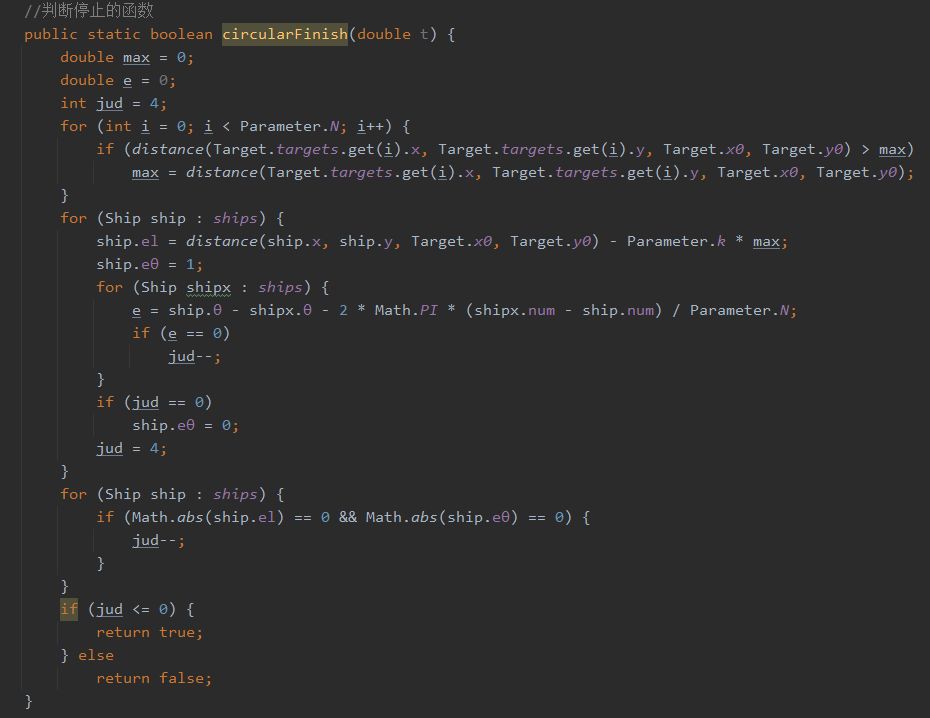
对于式，任意两个智能体i，j他们的角度差再减去智能体编号差除以n（总的智能体数量）乘以角度2Π，即表示任意两个智能体他们之间的角度差符合总的智能体平分一个圆的角度大小时说明智能体都到达指定位置。



总的计算思路是先是循环出目标的中心位置，然后在拿这个中心位置的信息去修改ρ(t)，然后在去修改l(t)和是θ(t)。

下用代码解释停止条件：

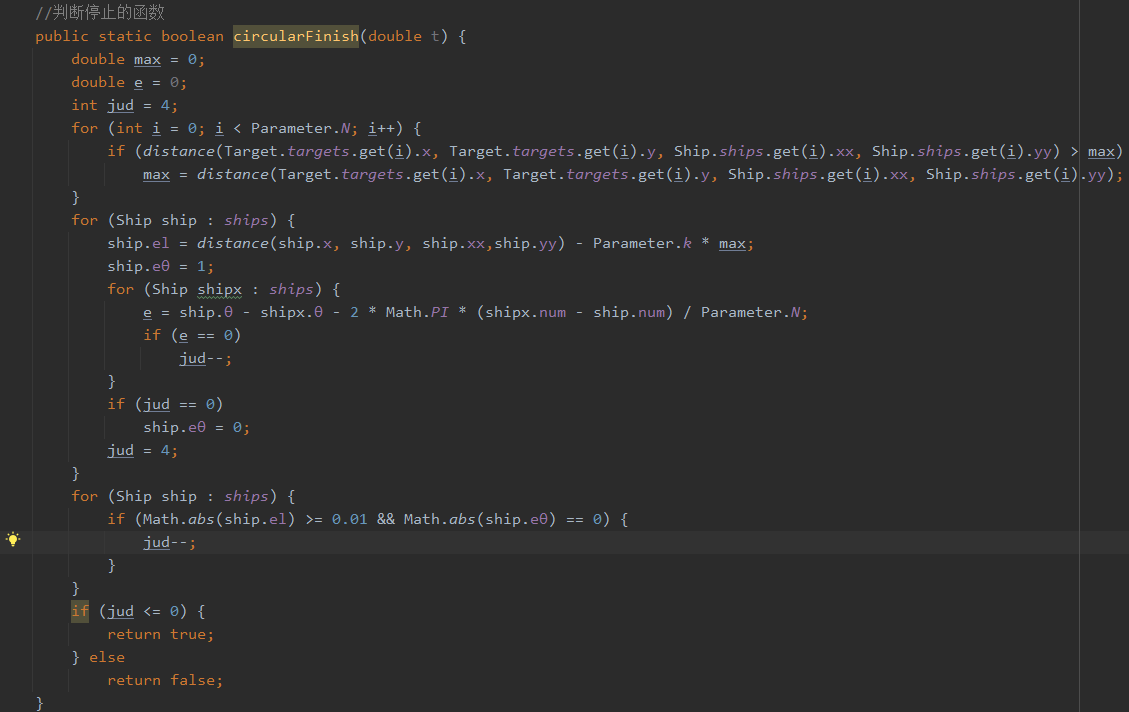
这里的代码使用集中式表现，即在目标中心位置实际值已知的情况下进行代入式（1.7）进行计算停止条件，在该停止条件下，智能体到达既定位置之后便停止。



circularFinish()函数是用来确定停止条件的。参数的定义分别是误差（距离停止条件的差值），用来方便计算的中间参数以及智能体的数量，用来判定是否每个智能体都达到停止条件。每个智能体的半径停止条件el计算首先通过循环计算目标到达目标中心距离的最大值，接着计算每个智能体到目标中心的距离与最大值的k倍差，角度计算同理。每有一个智能体同时符合半径和角度的停止条件，jud就-1，直到jud变成0时所有智能体均已符合停止条件，判定可以停止。最后如果判定都停止则返回true，否则就返回false。

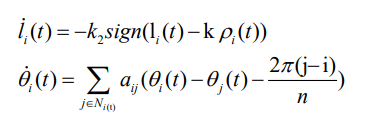
接下来使用分布式情况去改写停止条件，即式（1.7）中的

对于每个智能体来说并不知道，因此使用估计值Pi(t)去代替目标中心位置进行停止条件的计算：



其中将目标中心距离的实际值x0，y0替换成每个智能体的估计值xx，yy，并且通过测试给定了一定的误差，即el>=0.01即可停止，防止智能体调整时间过长而不停止，通过观察最后的结果也是可以正确实现包围。

**3.控制器设计：**

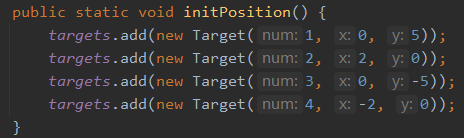
（1.5）

其中i ∈1,2,……,n，ρi(t)表示第i个智能体对目标距离目标平均中心的最大距离估计值。本文中 sign(.) 表示符号函数,k2 > 0 为控制器增益。

(1.6)

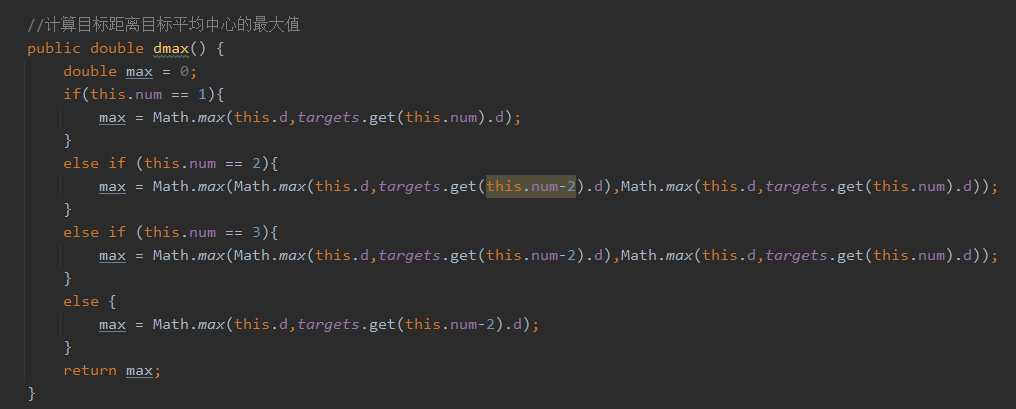
其中 i ∈1,2,……,n,k1 > 0 表示增益系数

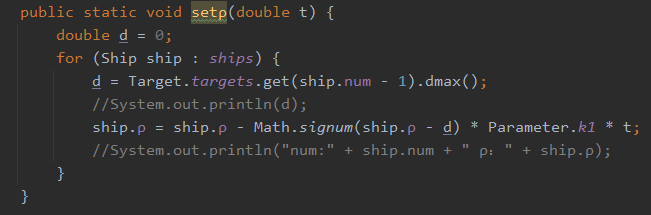
ρ的初值直接通过赋值，后续的修正采用上述的式子，式子表现为导数，在代码实现的时候，用导数的值乘上时间t来表示这个值的增量)。Max（d(t)）表示目标距离目标平均中心的最大距离，因为四个目标的位置是已知的，因此每一时刻目标平均中心的位置也是已知的，每一时刻通过比较每个目标到目标中心的距离，通过比较最大值即为Max（d(t)）。ρ的最终值表现为目标中距离目标中心最长的长度，比如这四个目标:



中心为（0，0），目标距离中心最长距离为5 所以最后ρ会稳定在5这个值。

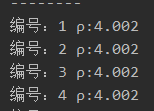
目标距离目标中心dmax的计算使用如下代码，这里没有按照论文中集中式的方式（即所有智能体都已知目标目标距离目标中心dmax的信息），而是按照邻域Ni(t)的方式，智能体只能获取对应目标对于dmax的信息。



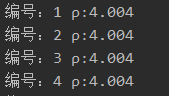


ρ的计算如上图所示，对于每个智能体来说，其只能获得自己对应的目标信息以及与之直接进行通信的目标信息，也就是说dmax的获取并不是严格按照论文中所说的直接计算出所有目标的dmax使得所有智能体都能获取这个值。这里dmax也是按照Ni(t)集合进行计算，每个智能体只能获取自己对应的以及能够直接通讯的智能体对应的目标提供的距离信息。

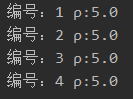
初始设定的ρ值均为4，实际最大值为5，第一次进行ρ计算后结果如下：



第二次进行ρ计算后结果如下：



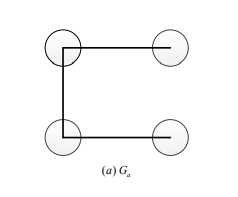
最终经过一段时间ρ值会与实际值相同：



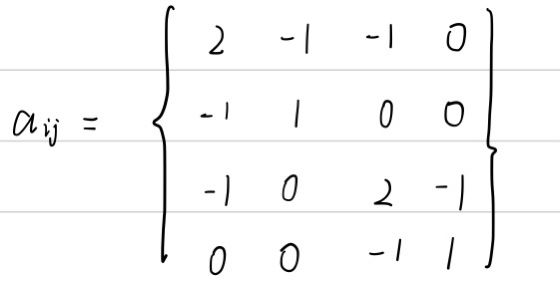
l(t)是极坐标下的半径长，初值由初始化的时候给出。最后应该实现船均匀的分配2Π角度，分布在半径为kρ(t)的圆上。式（1.5）中第一式左边为也即vi（t），右式为一个符号函数（sign）乘以一个系数-k2，符号函数中的式子通过比较当前智能体半径大小以及ρ的大小（这里乘以一个系数k来控制最终包围圆的大小）来控制第i个智能体的速度方向，其中大小是确定的参数k2，k2是给定的值。用比较通俗的话说，即如果当前智能体的到目标中心的距离大于kρi(t)，则对当前智能体到目标中心的距离进行缩小；如果当前智能体的到目标中心的距离小于kρi(t)，则对当前智能体到目标中心的距离进行增大，最终使智能体的到目标中心的距离等于kρi(t)。其中无论是缩小还是增大，变化的幅度都由给定的系数k2进行调整。

其中kρ(t)中的k是控制最后的船距离中心的半径长，比如示例中k为给定值2，ρ的最终稳定值是5，所以最后船与中心的距离应该稳定在10，组成一个半径为10的圆形。

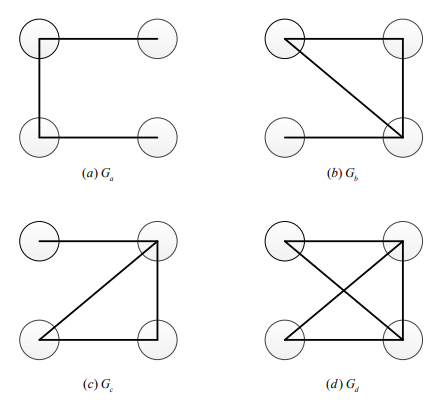
式（1.5）中第二式左边为，右边通过比较第i个智能体与其他所有智能体的角度大小以及总体智能体数量来不断控制智能体直接的角度差变成相同大小，即最终平分整个圆。

aij是拉普拉斯矩阵里的值，对于智能体小船，有着不同的通信拓扑结构，而根据联通情况可以得到这个连通图的拉普拉斯矩阵（对角线表示了节点的度，其他元素如果邻接为-1其他为0），比如下图的连通情况： 

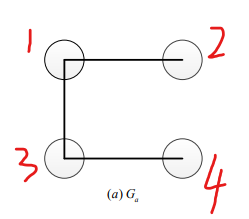
拉普拉斯矩阵为{{2,-1,-1,0},{-1,1,0,0},{-1,0,2,-1},{0,0,-1,1}}，而aij就是里面的值。



由于每个智能体只能获取与之对应的目标信息，因此想要实时获取目标中心的位置，就需要让智能体间进行信息交换，因此智能体所构成的图一定是连通的，这样才能保证每个智能体都能获取其他所有智能体的信息。这里拿我们选取的四个智能体作为例子，共有如下四种连通方式：



考虑到目前智能体数量较少，信息传递较快，因此我们选取连通方式最简单的一种：



剩下三种虽然能使每个智能体更准确的获取信息，但连通方式较复杂，期间进行信息交换时间相比于第一种有所提升。

之所以选择拉普拉斯矩阵，原理与半径的变化也是类似的。如上图所示，对于智能体1来说，智能体可以直接与智能体2、智能体3进行通讯，因此a11=2表示智能体1可以直接同时与其他两个智能体进行通讯。a12=-1、a13=-1在式（1.5）中的第二式乘以后面角度变化表示如果两者角度差比最终预期情况大，则往角度变小的方向变化；反之同理。智能体2、3、4在aij（拉普拉斯矩阵）里的表现与智能体1类似，这里就不再赘述。

**二阶多智能体协同分布式包围控制**

**问题描述：**

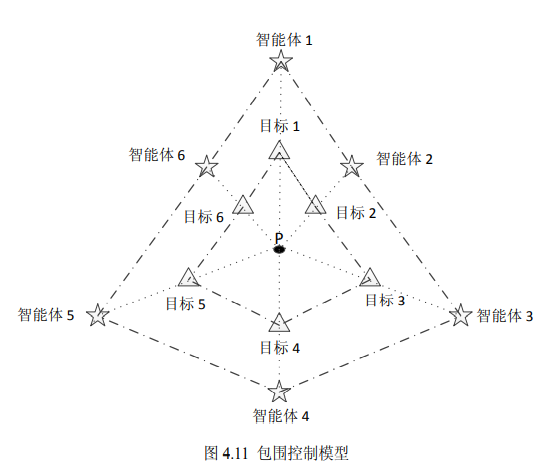
假设考虑有 n 个智能体的多智能体系统，每个智能体看作是连通无向通信拓扑图 G 中的一个节点，每条边(sj,si)对应个体 j 到 i 可进行通信的信息链。并且，每个智能体通过自身状态和邻接智能体状态更新当前状态。假设多智能体系统每个 智能体满足以下动态方程：



（参考资料[1]式（5.1））

其中,xi(t) 表示第 i 个智能体的位置状态信息， vi(t)表示第 i 个智能体的速度状态信息，ui(t) 表示第 i 个智能体的控制输入。

简单来说，要实现n个智能体对n个目标的包围，其中每个智能体对应一个目标，可以不断获取对应目标的状态信息。n个智能体是连通的，可以看成一个大小为n的连通图，因此每一个智能体获取对应目标的信息后可以通过连通将每一个目标的状态信息转递给其他所有智能体，因此在下一时刻，所有智能体都能获取当前时刻所有目标的信息。然后根据状态方程智能体不断修改自己的状态，最终达到一个包围的效果，即以一个更大的相同图案将目标包围。



**研究思路：**

根据上式（5.1）可以看出第i个智能体的控制输入为ui（t），而ui（t）是速度vi（t）对于时间t的导数，因此对于该二阶多智能体系统分布式包围控制模型的最关键初始输入为各个智能体的加速度（这里先不考虑智能体的质量，后续考虑智能体质量控制输入为各个智能体的推力）。

其他初始信息为：

目标数n（目标数与智能体数相当）

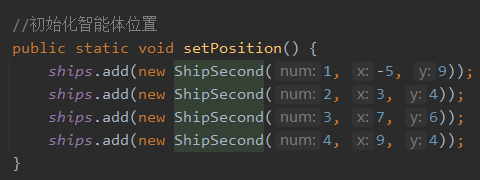
各个目标位置r（用坐标表示）

各个智能体的初始位置（用坐标表示）

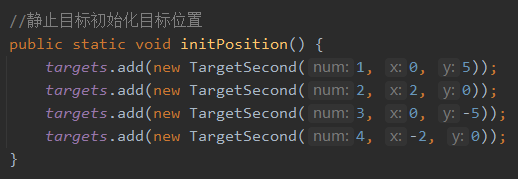
目标的移动轨迹（用方程表示）

对于该模型，目前只考虑给予的推力所带来的速度，而不考虑水流速度与阻力等。我们以目标中心（表示目标的平均中心）为原点建立直角坐标系，将各个智能体的速度与加速度按照坐标系x，y方向进行分解，按照两个方向的情况进行修改后最后再合并去让智能体移动，改变智能体的x，y坐标。而各个智能体每个时刻的加速度、速度以及位置信息的变化则在控制器中去考虑，因此需要设计一个控制器算法，进行智能体状态的实时修改。

下面代码定义了智能体的初始位置：

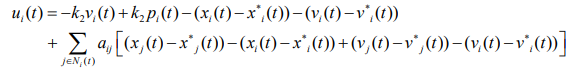


下面代码定义了目标的初始位置：



**接下来进行编程实现控制器，控制器设计原理略去：**

实际上，控制器就是一个根据不同时刻修改智能体状态信息的函数，按照二阶系统的思路，智能体每个时刻更改加速度大小与方向，然后按照数学中的积分公式去推出下一时刻的速度与位置，这样不断循环直至达到终止条件停止。第i个控制输入即第i个智能体的加速度信息可以用如下算法实现：

（参考资料[1]式（5.3））

其中，，k2>0为控制器增益。xi(t)为第i个智能体的位置信息，vi(t)为第i个智能体的速度信息，aij为智能体采用的连通图拉普拉斯矩阵，pi(t)为第i个智能体对于目标中心的估计值，ri(t)为第i个目标的状态信息。

为了简便分析，对模型进行了转换，令



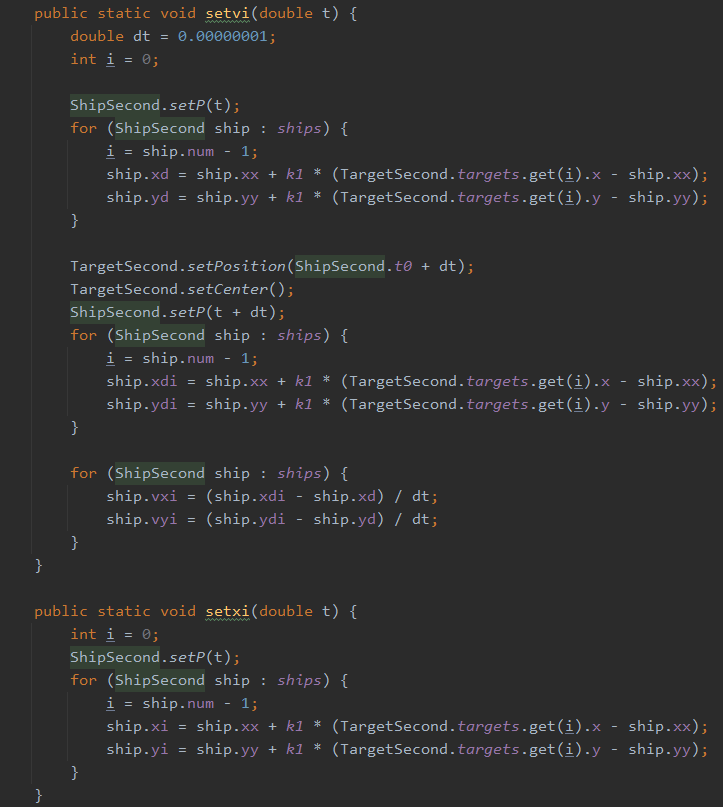
因此式（5.3）可转换为：

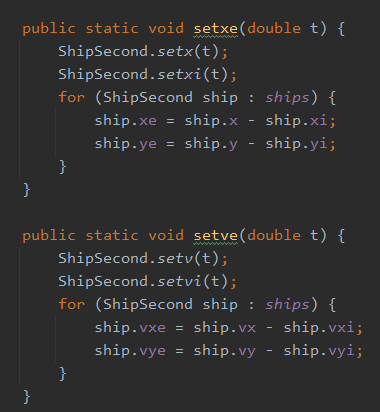


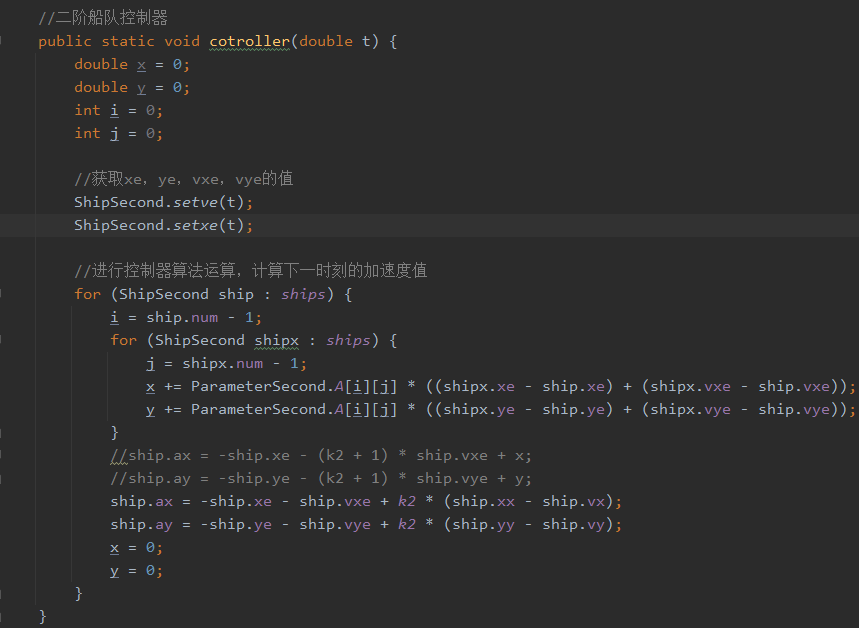
（参考资料[1]式（5.4））

但在实际实现的过程中，按照该式子作为控制器作为调整智能体的依据会导致智能体不会停止，而且位置坐标会一直变大。为了解决这个问题，我经过各种尝试发现，如果将智能体与目标之间的速度差信息删去，并增加控制器增益k2的值，经过一段时间后智能体的位置信息将会趋于稳定并且值会比较正常。

**控制器代码部分：**

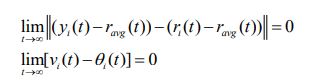






**关于智能体的运动终止：**

通过测试代码，我们发现经过一段时间后，智能体已经成功将目标包围，但是智能体会在一个坐标范围内不断变化，因此我们设法让智能体停下来。通过研究论文，我们发现以下两个式子可以作为智能体运动的终止条件：



其中，ravg(t)表示目标的平均中心，rk(t)表示第k个目标的状态（k∈I，k1>1表示编队队形系数增益），ri(t)表示第i个目标的位置状态信息，θi(t)表示第i个目标的速度状态信息。

和一阶系统中类似， (yi(t)- ravg(t))表示第i个智能体到目标几何平均中心的误差向量，(ri(t)- ravg(t))表示第 i 个目标到目标几何平均中心的误差向量，二阶系统比一阶系统多出速度误差向量 (vi(t)- θi(t))。当然在实际计算中一定有误差的存在，因此我们将0的值扩大，经过测试，这个值为1.72时就可以让智能体在成功包围目标的情况下停下。

**停止条件代码：**



、

**静态目标与动态目标：**

对于二阶系统，静态目标与动态目标本质上是一样的，通过控制器都能成功实现。仅有的区别为目标状态的设置，对于静态目标来说，通过初始输入目标的状态信息就能确定目标的位置，而智能体按照控制器经过一段时间后便可以成功包围目标。而对于动态目标来说，首先需要设定好目标的轨迹，然后考虑是否给目标一个停止条件，如果目标不停止的话，那么智能体只能做到在移动过程中不断跟随智能体，而在移动过程中无法达到对目标的准确包围，除非设定智能体的移动频率比目标快，但是这个频率对于不同情况也是不确定的，因此我们选择让目标移动一段时间后（即到达目的地）停止，这时候一直跟随的智能体可以在目标停止后不断调整位置最后达到包围的目的。

**代码说明**：

代码环境：Java 16

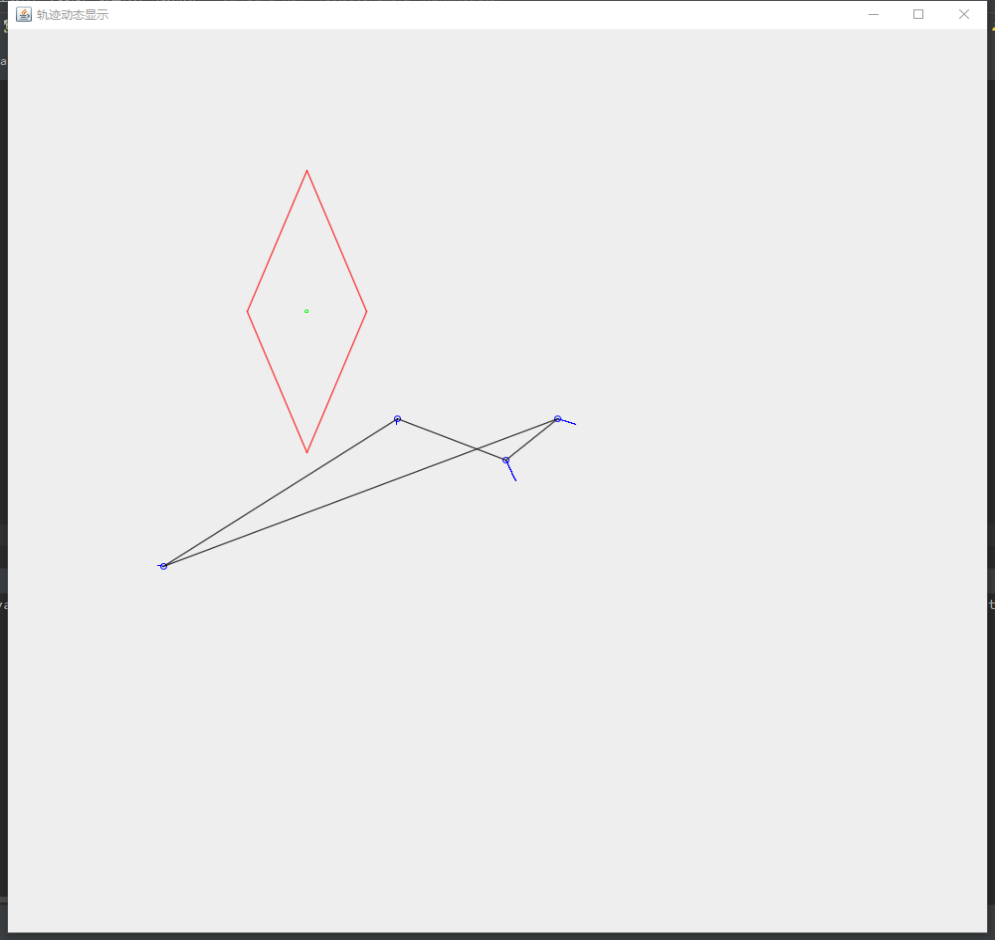
开发环境：IntelliJ IDEA

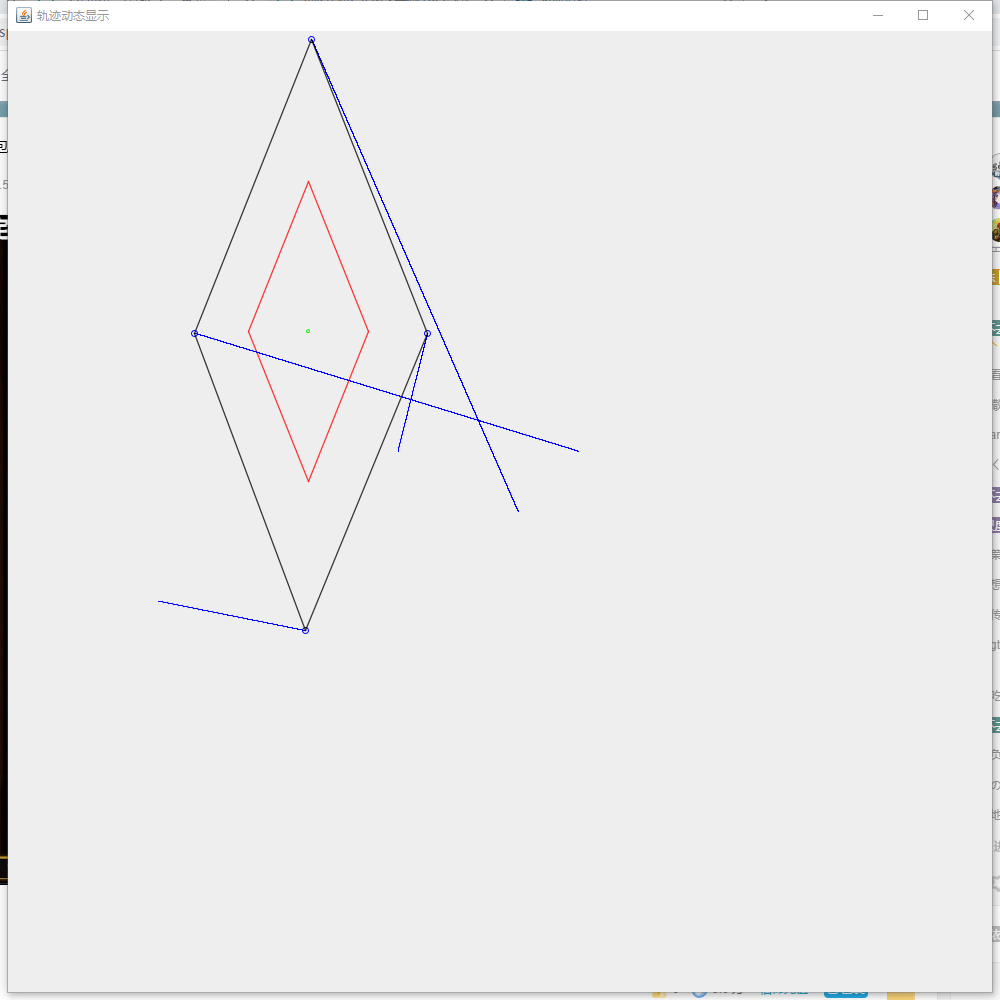
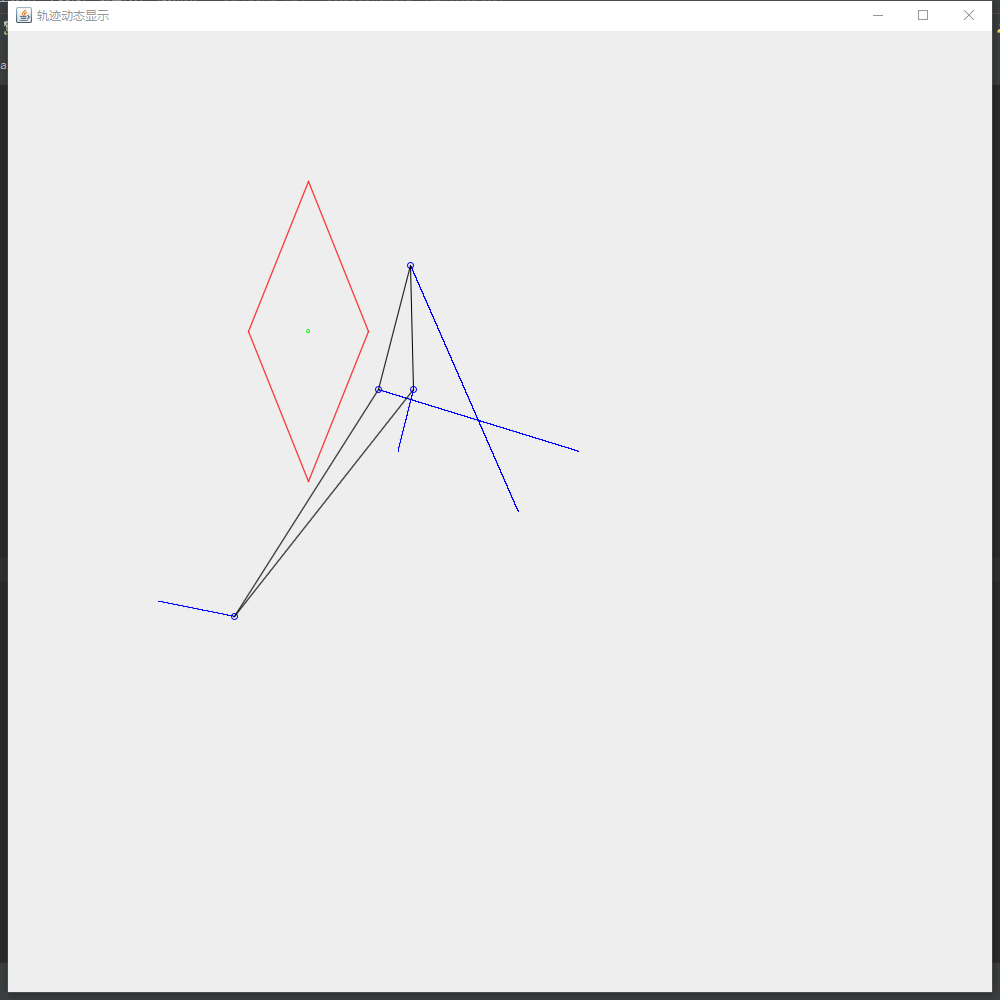
**二阶效果展示**：

1.静态目标：

其中红色的线表示目标所连成的线，其中四个红色顶点为静态目标的四个位置，绿色的点为目标中心位置，蓝色的点表示智能体，蓝色的线表示智能体所走过的路径，黑色的线表示智能体所连的线。

可以看到最终智能体对于静态目标实现了包围。





2.动态目标：

可以看到动态目标中目标与智能体的位置一起发生改变，直到到达目的地后目标停止，经过一段时间智能体实现对目标的包围。

