# Decision Making in Robots and Autonomous Agents
# Assignment 2

s0831408

## Assumptions:

During this assignment, a few assumptions were made:

1. All robots are identical, and have the same uncertainty kinematic equation
2. All robots know their own location (exact location, not expected)
3. They do not know where the vents are
4. They know the locations of all other robots
5. There is a centralised auctioneer for the grouping and movement decider

## Key:

The following is a list of all the features visible in the plots of the environment:

- The black crosses represent the three vent locations
- The coloured triangles represent the calculated centre point of each group
- The coloured stars represent the optimal formation points of the robots, given the calculated centre of the group and the given radius
- The coloured dots represent each robot, coloured by team
- The coloured circle represents the formation circle given by the set radius

## Classes:

The problem was programmed in Python, with the following classes and methods:

1. Robot
    - Able to get the location of others and send own current location
    - Able to get and send allocated formation positions
    - Able to send current sensor reading to auctioneer
2. Group
    - Holds the IDs of the robots in the group
    - Holds the centroid location of the group
3. Vents
    - Has standard deviation and location
    - Calculates the reading at a given point

## Formation Control:

The formation of the group is controlled in the following steps:

1. When the grouping of robots is decided, we compute the centre point of all the member's positions (this is the triangle in the figures).
2. We use a set radius (0.25), and trigonometry to calculate four points, evenly spread around the circumference from the group's centre point, using the set radius. We use these four points as the positions in which a member of the group should position. This forms a square like pattern around the centre point of the group.
3. We use the robot's initial locations, in respect of the centre, to allocate each member one of the quadrants. For example, if member one's position is higher than the centre's $x$-value and higher than the centre's $y$-value, it receives the top right quadrant. This is the quadrant it will stay in for the rest of the problem. As it checks which quadrant it should be in, it requests all other member's assignment to ensure that the quadrant is free. If it is not, it accepts the next free quadrant.
4. Once the robot knows its desired location with regards to all other robots, it moves towards the location. This is done in small incremental steps. For each robot, we calculate the distance from current location to desired location (allocated quadrant position), and move towards it in steps of maximum size 0.001. We increment both $x$ and $y$ values separately to ensure that if a robot is in the way, in one of the directions, it does not move in that direction. This incremental update process continues until the robot is within the known error of the desired location.

An example of this algorithm in action can be seen in the following figures. We use a uniform distribution over the square unit to get the initial positions of the robots. We then assign each robot to a group, first 4 in group one, next 4 in group two, and final 4 in group three. Figure 1 shows this basic grouping and initial centre points. Figure 2 shows the robots after they have updated their locations. On average it took 56 moves to position the robot on it's formation position, given maximum movements of 0.001.

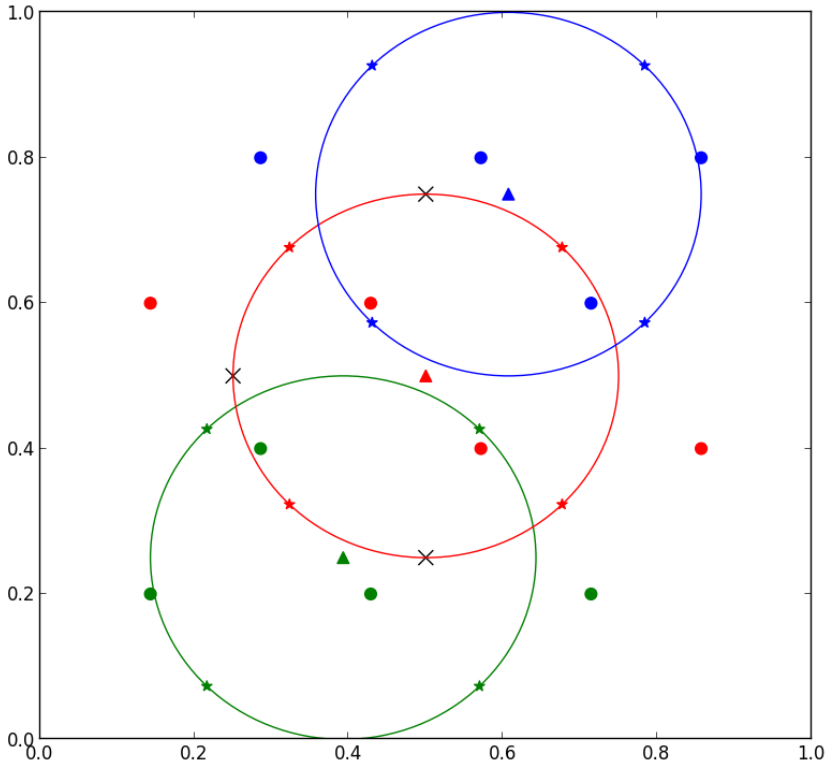


*Figure 1: Shows the initial positions of the robots and centroids of groups*

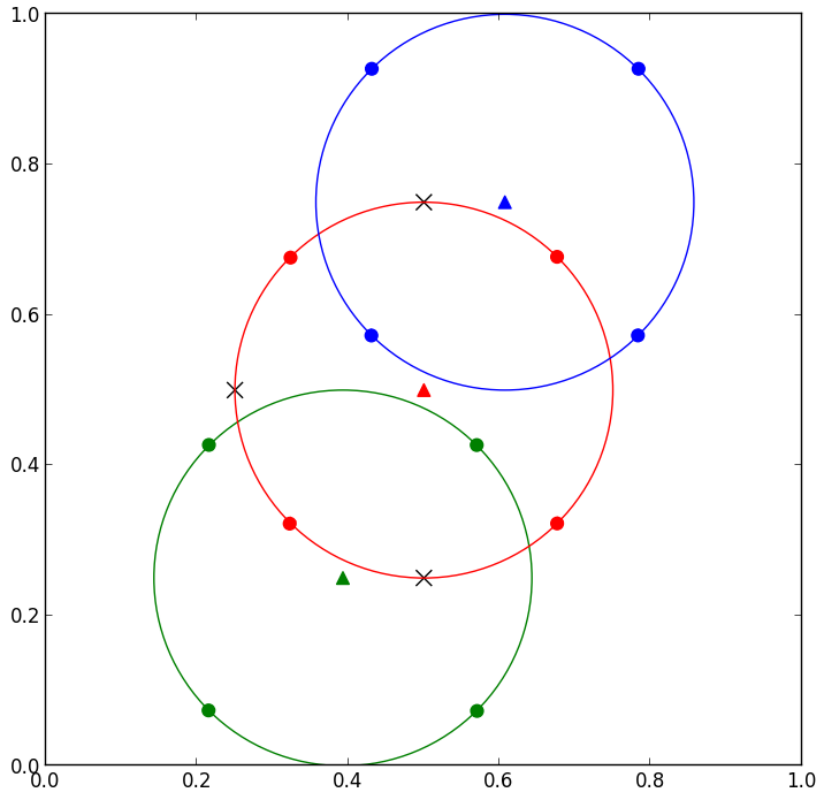


*Figure 2: After the robots have moved to the group formations*

Once in formation, to move the robots, we first determine the required movement, then apply it to the centre location (known locally by all robots in the group) and current locations of the robots in the group. Figure 3 shows a transformation of +0.3 in the *x*-value of the green from from Figure 2. Even though only the *x*-value has been changed, the movement includes error in both *x* and *y* directions during a move. As seen in Figure 3, the transformation has moved the robots away from their desired formation. Thus, we again apply the iterative formation movement function, moveToPost( ), to reposition the robots as desired. The final positions of the robots can be seen in Figure 4.

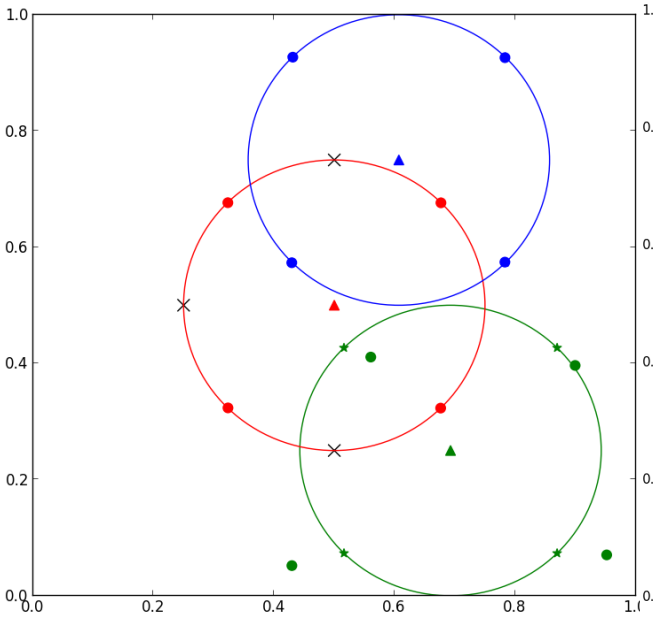


*Figure 3: Movement with error*

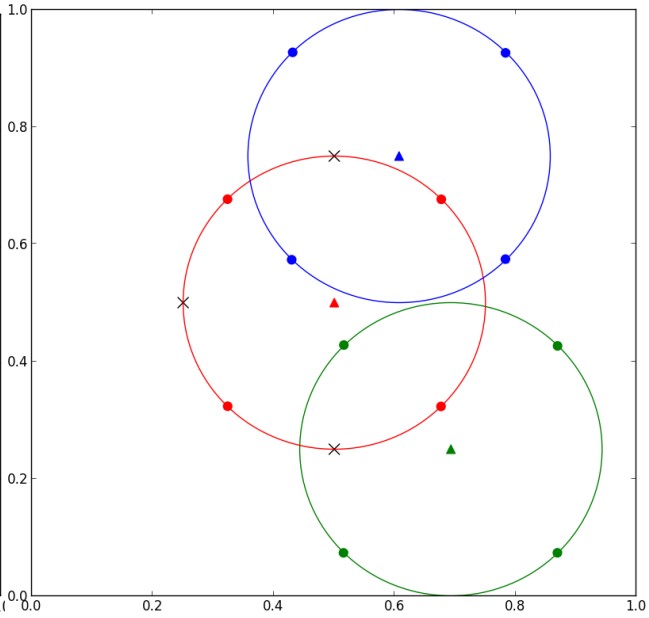


*Figure 4: Errors corrected to formation*

## Grouping Auction:

The grouping of the robot's is decided through a single first-price auction as all robots bid truly. When each robot is created, it is assigned a unique ID. We select one of these IDs at random and declare the corresponding robot the 'seller'. All other robots in the auction bid to become a member of it's team. The value they bid corresponds to the distance between their initial location and the 'seller's' initial position. From the bid, the three lowests bidding robots are assigned to the seller's group. The seller and the assigned robots are removed from the auction. A second auction, for the remaining robots is conducted in the same manner. After the second auction, the remaining robots are assigned to the third group. Figure 5 illustrates the auction grouping, and Figure 6 illustrates the group's initial position after they move into formation. We use the distances to the seller as the bids as this gives an accurate representation of how desirable it is to be apart of that group. This ensures that robots move the minimum required in order to get into formation quickly.

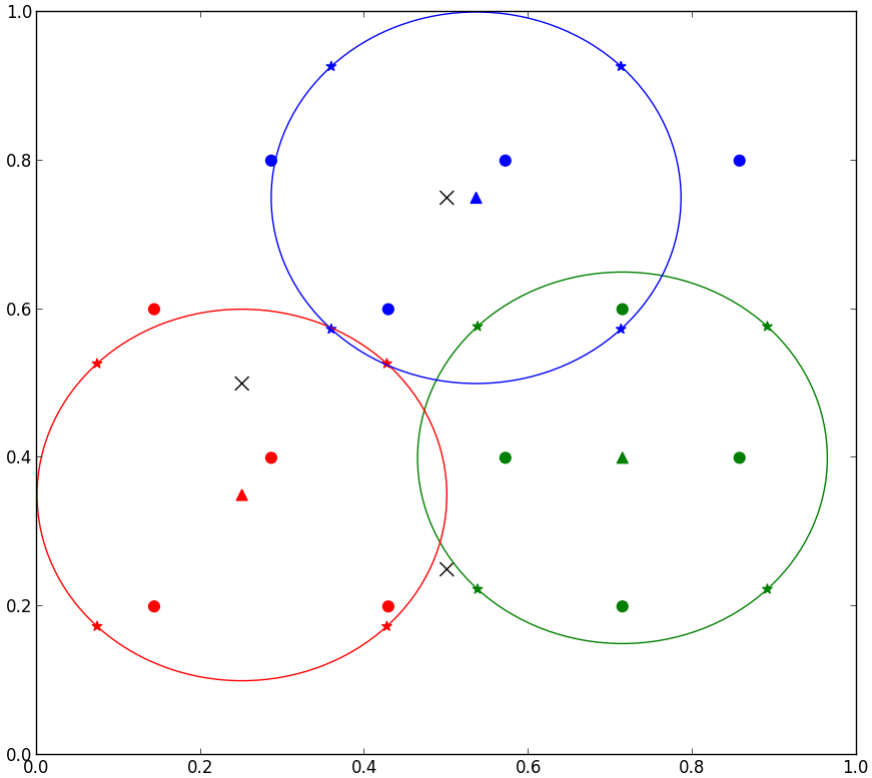


*Figure 5: The grouping of robot's after the first-price auction*

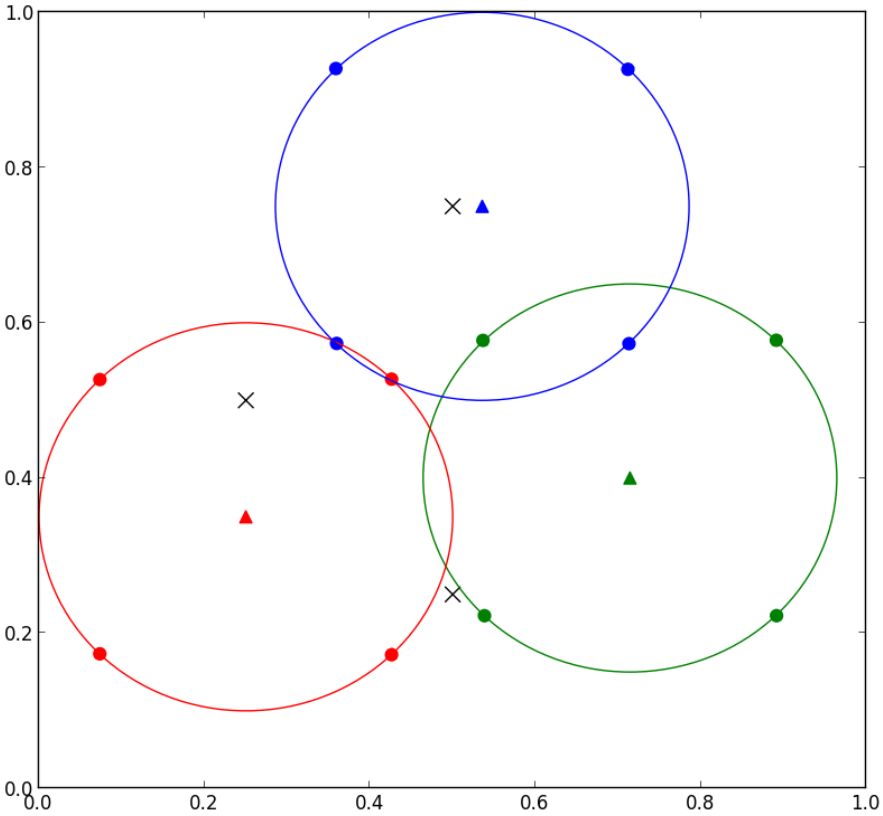


*Figure 6: Robot positions in formation of groups*

## Movement Auction:

From the current location of each robot, we are able to collect a sensor reading for the chemical levels from each vent using a 2D Gaussian function, as described in Equation (1), where $(x,y)$ is the current location of the robot and $(x_0,y_0)$ is the location of the vent.

$$f(x,y) = A exp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)\right) \quad (1)$$

We combine all sensor readings together to get a final sensor reading for the robot. The combined sensor reading as a bid is an accurate representation of how much the robot believes their direction is best. Again, as the robot's bid truly we will use a single first-price auction to decide the movements of the group. We use the sensor readings from each robot in a group as it's bid. The winning robot is that with the highest sensor reading. Each robot in the group request the quadrant number from the winning robot, and using this, moves in both $x$ and $y$ by 0.001 in the direction of this quadrant. After each move of each group, the robots ensure their still in formation using the method above. After 1000 iterations we get the final position of each group, seen in Figure 7. Each robot is very close together but no closer than the minimum 0.05. The point with the highest chemical levels was found at (0.42, 0.4).

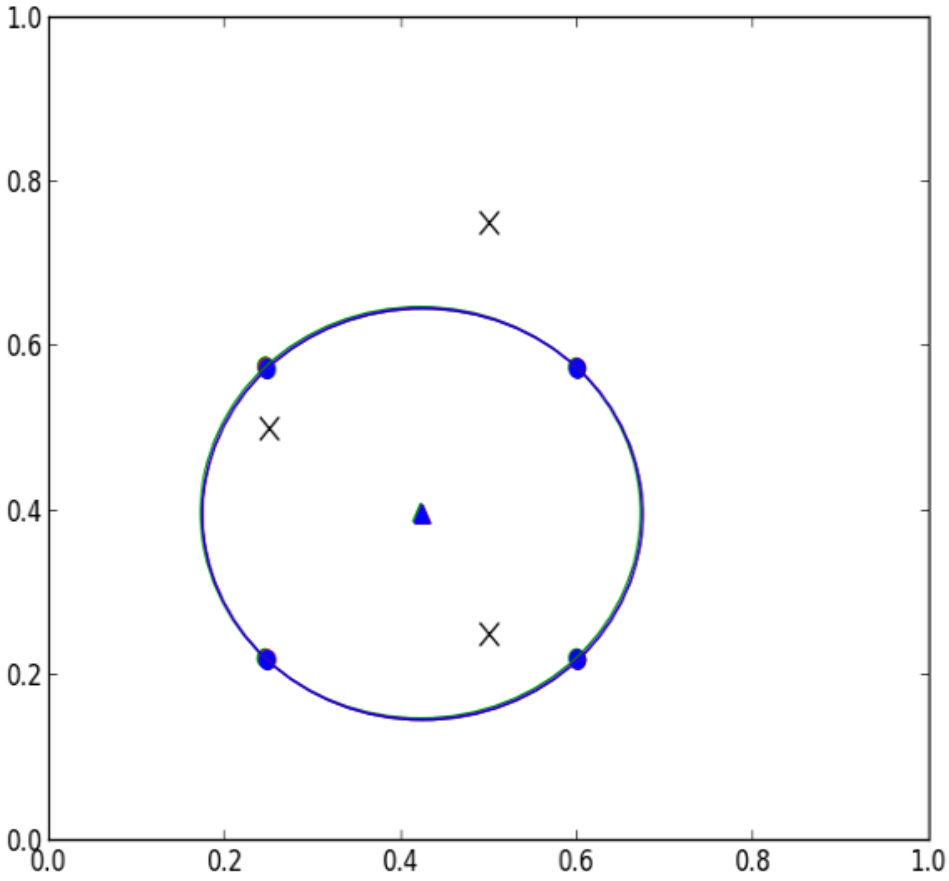


*Figure 7: The final positions of the groups, showing the location of the maximum reading*

# CODE

## Main.py

```
#Author: s0831408
from __future__ import division
from Robot import *
from Group import *
#from Auctioneer import *
from Vent import *
from random import randint

import math
import random
import matplotlib.pyplot as mp

""" This is the main class which sets up the environment, drops the robots,
       and begins the interactions of objects to obtain a final position."""

#-----------Initialise Variables-----------------
numGroups = 3       #Number of groups of robots
numRobots = 12             #Number of robots
numVents = 3        #Number of vents
seabedSize = 1             #Size of oceanbed in miles
radius = 0.25       #The distance from centre each robot should be
maxMove = 0.001            #This is the max a robot can move per turn

numRobotsPerGroup = int(numRobots / numGroups)

#Lists to hold the objects of the environment
groupList = []
robotList = []
ventList = []

#==============================PLOT ENVIRONMENT=================================
def plotEnv():
       groupColours = ['Green','Red','Blue']
       mp.axis([0,1,0,1])

       #Plot vents in black (crosses)
       for vent in ventList:
              mp.scatter(vent.getX(), vent.getY(), s=100, marker='x', color='black')

       #Plot each team of robots in a different colour (plot triangles for centroids)
       for i in xrange(numGroups):
              mp.scatter(groupList[i].getX(), groupList[i].getY(), s=60, marker='^',
                     color = groupColours[i])

              #Plot the formation circle of each group
              circle = mp.Circle((groupList[i].getX(), groupList[i].getY()),
```

```
                        radius, color = groupColours[i], fill=False)
                fig = mp.gcf()
                fig.gca().add_artist(circle)

                robots = groupList[i].getGroupRobotIDs()

                for j in xrange(numRobotsPerGroup):
                        #Plot robots
                        mp.scatter(robotList[robots[j]].getX(), robotList[robots[j]].getY(),
                                s=60, marker='o', color = groupColours[i])

                        #Plot robot's post
                        post = getPost(groupList[i].getX(), groupList[i].getY(),
                                robotList[robots[j]].getQuarter())
                        mp.scatter(post[0],post[1],s=50, marker ='*',color =
groupColours[i])

        mp.show()

#================================GET POST=====================================
def getPost(x, y, seg):
        """ This gets the point the robot should be positioned at, given the quarter
                it is assigned, the radius, and the current centre of the group"""
        #Use trig to get the x and y move size
        post = [0,0]
        xyMove = radius * math.sin(0.785398)

        if seg == 0:
                post[0] = x - xyMove
                post[1] = y + xyMove
        elif seg == 1:
                post[0] = x + xyMove
                post[1] = y + xyMove
        elif seg == 2:
                post[0] = x + xyMove
                post[1] = y - xyMove
        else:
                post[0] = x - xyMove
                post[1] = y - xyMove

        return post

#==============================================================================
def moveToPost(group, robot):
        """ Moves the specified robot within group formation. Returns True if the
                robot needed to be moved"""
        moved = False
        errorVar = [0.0025, 0.000625]     #The sd of error in x and y movement
        current = [0,0]                                 #Hold current x and y
        move = [0,0]                            #Holds move amount in x and y
```

```
	distFromPost = [0,0]		#Holds the x and y distances from R to P

	current[0] = robot.getX()
	current[1] = robot.getY()

	#Get the point the robot should be at from centre of group & quarter
	post = getPost(group.getX(),group.getY(),robot.getQuarter())
	distFromPost[0] = post[0] - current[0]
	distFromPost[1] = post[1] - current[1]

	#If further away than the error variance, move (up to max amount)
	for i in xrange(2):
		if abs(distFromPost[i]) > 0.001: #errorVar[i]
			moved = True

			#Get movement amount
			if abs(distFromPost[i]) > maxMove:
				#Make sure we have correct sign
				if distFromPost[i] < 0:
					move[i] = -maxMove
				else:
					move[i] = maxMove
			else:
				move[i] = distFromPost[i]

			#Update robot position to new position + error
			if i == 0:
				error = random.gauss(0, math.sqrt(errorVar[i]))
				robot.setX(current[i] + move[i] + error)
			else:
				error = random.gauss(0, math.sqrt(errorVar[i]))
				robot.setY(current[i] + move[i] + error)

	return moved
#================================================================================

#-----------------------------Build Environment---------------------------------
#Uniformaly distribute using circles (6 columns, 4 rows - +1 to get away from edges)
colStep = 1/7
rowStep = 1/5


#Create the robots giving unique ID
robot = Robot((1*colStep), (1*rowStep), 0)
robotList.append(robot)
robot = Robot((3*colStep), (1*rowStep), 1)
robotList.append(robot)
robot = Robot((5*colStep), (1*rowStep), 2)
robotList.append(robot)
robot = Robot((2*colStep), (2*rowStep), 3)
```

```
robotList.append(robot)
robot = Robot((4*colStep), (2*rowStep), 4)
robotList.append(robot)
robot = Robot((6*colStep), (2*rowStep), 5)
robotList.append(robot)
robot = Robot((1*colStep), (3*rowStep), 6)
robotList.append(robot)
robot = Robot((3*colStep), (3*rowStep), 7)
robotList.append(robot)
robot = Robot((5*colStep), (3*rowStep), 8)
robotList.append(robot)
robot = Robot((2*colStep), (4*rowStep), 9)
robotList.append(robot)
robot = Robot((4*colStep), (4*rowStep), 10)
robotList.append(robot)
robot = Robot((6*colStep), (4*rowStep), 11)
robotList.append(robot)


#Create the vents from the assignment
ventA = Vent(0.5, 0.75, 0.2)
ventB = Vent(0.25, 0.5, 0.3)
ventC = Vent(0.5, 0.25, 0.4)

ventList.append(ventA)
ventList.append(ventB)
ventList.append(ventC)

#Create the three groups
for i in xrange(numGroups):
	group = Group(i)
	groupList.append(group)


#------------------------------Group Robots-------------------------------------
#Using single auction, group robots
botsToAssign = [0,1,2,3,4,5,6,7,8,9,10,11]
for i in xrange(numGroups-1):
	bids = []
	groupMembers = []

	#Elect a seller for robot's to bid to join
	leaderID = randint(0,11)
	#Add to group list and remove from auction
	groupList[i].addRobot(leaderID)
	botsToAssign.remove(leaderID)

	#Get bids from all other bidders in auction
	for j in botsToAssign:
		#get distance to seller
```

```
                distSeller = math.sqrt((robotList[leaderID].getX()-robotList[j].getX())**2
+
                        (robotList[leaderID].getY()-robotList[j].getY())**2)
                #Append bid and bidder ID
                bids.append([distSeller,j])

        #Auctioneer assigns three lowest bidders the task (group id)
        bids.sort()
        for j in xrange(numRobotsPerGroup-1):
                groupList[i].addRobot(bids[j][1])
                botsToAssign.remove(bids[j][1])

#Assign remaining robots to final group
print botsToAssign
for botID in botsToAssign:
        groupList[(numGroups-1)].addRobot(botID)


#Set initial centroid of group
for group in groupList:
        xs = 0
        ys = 0
        roboIDs = group.getGroupRobotIDs()

        for j in roboIDs:
                xs += robotList[j].getX()
                ys += robotList[j].getY()

        group.setX((xs/numRobotsPerGroup))
        group.setY((ys/numRobotsPerGroup))

#Set what quarter each robot is posted to
for group in groupList:
        roboIDs = group.getGroupRobotIDs()

        count = 0
        for j in roboIDs:
                robotList[j].setQuarter(count)
                count += 1


#------------------Initially rally the groups into formation--------------------
plotEnv()

#Move robots to correct formation (repeats until all teams have converged)
for group in groupList:
        for i in xrange(numRobotsPerGroup):
                robot = robotList[group.getMemberID(i)]

                positionUpdated = True
```

```
            while (positionUpdated == True):
                  positionUpdated = moveToPost(group, robot)

plotEnv()

#------------------------------Begin Move Cycle--------------------------------

for i in xrange(1000):
      #For each robot, get sensor readings
      for robot in robotList:
            readingA = ventList[0].getPDF(robot.getX(),robot.getY())
            readingB = ventList[1].getPDF(robot.getX(),robot.getY())
            readingC = ventList[2].getPDF(robot.getX(),robot.getY())
            sensorReading = readingA + readingB + readingC

            robot.setSensorReading(sensorReading)


      #For each group, bid for which direction to move
      for group in groupList:
            members = group.getGroupRobotIDs()
            bids = []

            for member in members:
                  bids.append([robotList[member].getSensorReading(),member])

            bids.sort()
            #Get the quarter for the winning bidder
            direction = robotList[bids[len(bids)-1][1]].getQuarter()

            #Now move the centres
            if direction == 0:
                  group.setX(group.getX()-0.001)
                  group.setY(group.getY()+0.001)
            elif direction == 1:
                  group.setX(group.getX()+0.001)
                  group.setY(group.getY()+0.001)
            elif direction == 2:
                  group.setX(group.getX()+0.001)
                  group.setY(group.getY()-0.001)
            else:
                  group.setX(group.getX()-0.001)
                  group.setY(group.getY()-0.001)

            #Move robots to correct formation (repeats until converged)
            for i in xrange(numRobotsPerGroup):
                  robot = robotList[group.getMemberID(i)]

                  positionUpdated = True
                  while (positionUpdated == True):
```

```
                        positionUpdated = moveToPost(group, robot)

print "Group 1: ("+str(GroupList[0].getX()) +", "+ str(GroupList[0].getY())+")"
print "Group 2: ("+str(GroupList[1].getX()) +", "+ str(GroupList[1].getY())+")"
print "Group 3: ("+str(GroupList[2].getX()) +", "+ str(GroupList[2].getY())+")"
plotEnv()
```

# Robot.py

```
#Author: s0831408

class Robot(object):
	"""This is the robot class which creates robot objects. Each robot object has an
	ID number, an a group number (which is assigned through the auction).
	They are able to send sensor data to the central decision maker (auctioneer)
	and will move as instructed by the winner. The robots can localise in the
	grid, and using their team member's positions, update their own to stay
	within the correct radius"""

	def __init__(self, x, y, id):
		self._x = x
		self._y = y
		self._id = id
		self._sensor = 0

		#This is the quarter it hold in the group's circle (tl=0,tr=1,br=2,bl=3)
		self._quarterPost = -1

	def getID(self):
		return self._id

	def getX(self):
		return self._x

	def getY(self):
		return self._y

	def getSensorReading(self):
		return self._sensor

	def getQuarter(self):
		return self._quarterPost

	def setX(self, x):
		self._x = x

	def setY(self, y):
		self._y = y

	def setSensorReading(self, i):
		self._sensor = i

	def setQuarter(self, num):
		self._quarterPost = num
```

# Vent.py

```
#Author: s0831408
import math

class Vent(object):
	""" This class represents one of the hydrothermal vents in the task. It has a
	position and standard deviation. It can be used to get the reading of the
	chemical level by passing it a grid reference, it then sends back the
	reading at this point. """


	def __init__(self, x, y, sd):
			self._mean = 0
			self._sd = sd
			self._x = x
			self._y = y

	def getMean(self):
		return self._mean

	def getSD(self):
		return self._sd

	def getX(self):
		return self._x

	def getY(self):
		return self._y

	def getPDF(self, x, y):
		"""Given a location, calculate value"""
		xSide = ((x - self.getX())**2) / 2*self.getSD()**2
		ySide = ((y - self.getY())**2) / 2*self.getSD()**2
		val = math.exp(-(xSide + ySide))
		return val
```

# Group.py

```
#Author: s0831408

class Group(object):
	""" This hold the IDs of the robots in the group and the centroid postion of the
	group. """

	def __init__(self, i):
		self._groupNo = i
		self._x = 0
		self._y = 0
		self._groupRobotList = []

	def getGroupNo(self):
		return self._groupNo

	def getGroupRobotIDs(self):
		return self._groupRobotList

	def getMemberID(self, i):
		return self._groupRobotList[i]

	def getX(self):
		return self._x

	def getY(self):
		return self._y

	def setX(self, x):
		self._x = x

	def setY(self, y):
		self._y = y

	def addRobot(self, id):
		self._groupRobotList.append(id)
```