VARIABLE GLOSSARY

Table 1: Dynamics and Integration Parameters

Symbol	Type	Units	Value	Description
A_i	scalar	m^2	1	agent characteristic area
$C_{d,i}$	scalar	none	.25	agent coefficient of drag
m_i	scalar	kg	10	agent mass
$\boldsymbol{F}_{p,i}$	3×1 vector	N	Eqn. 6	Prop. force vector
$\boldsymbol{n}_i^*$	3×1 unit vector	none	Eqn. 24	Prop. force direction
$F_{p,i}$	scalar	N	200	Prop. force mag.
$\boldsymbol{F}_{d,i}$	3×1 vector	m	Eqn. 6	Drag vector
$\boldsymbol{r}_i$	3×1 vector	m	Eqn. 1	agent i position
$\boldsymbol{v}_i$	3×1 vector	m/s	Eqn. 2	agent i velocity
$\boldsymbol{a}_i$	3×1 vector	m/s^2	Eqn. 3	agent i acceleration
$\boldsymbol{v}_a$	3×1 vector	m/s	$\mathbf{0}$	Air velocity
ρ_a	scalar	kg/m^3	1.225	Air density
Δt	scalar	s	.2	Time step size
t_f	scalar	s	60	Maximum task time

Table 2: Objects and Interactions Parameters

Symbol	Type	Units	Value	Description
`agent_sight`	scalar	m	5	Target mapping distance
`crash_range`	scalar	m	2	agent collision distance
N_m	scalar	none	15	Number of initial agents
N_o	scalar	none	25	Number of obstacles
N_t	scalar	none	100	Number of initial targets
$\boldsymbol{O}_j$	3×1 vector	m	Eqn. 10	Obstacle j position
$\boldsymbol{T}_j$	3×1 vector	m	Eqn. 10	Target j position

Table 3: Genetic Algorithm Parameters

Symbol	Type	Units	Value	Description
`children`	scalar	none	6	Strings generated by breeding
`parents`	scalar	none	6	Surviving strings for breeding
`S`	scalar	none	20	Designs per generation
`G`	scalar	none	100	Total generations
$L*$	scalar	none	Eqn. 26	Fraction of agents lost
M^*	scalar	none	Eqn. 26	Fraction of targets remaining
T^*	scalar	none	Eqn. 26	Fraction of time used
w_1	scalar	none	70	Weight of mapping in net cost
w_2	scalar	none	10	Weight of time usage in net cost
w_3	scalar	none	20	Weight of agent losses in net cost

DELIVERABLES

Introduction:

1. Outline the scenario being modeling by this project.

   The genetic algorithm determines the optimal parameters for a drone simulation of a swarm of autonomous vehicles (agents) whose task is to map a set of targets, avoid collisions with obstacles, and avoid collisions with each other in the fastest possible way in a given domain. (See Introduction)

2. Explain the specific goals of the project.

   (See Objectives).

3. Specify the methods that will be explained and the structure of the paper section.

   This paper explains the implementation of the drone simulation function, the genetic algorithm, the parameters to be initialized, the values we will converge to (the minimum of the cost function), and the plots we will produce along with the output of the optimal parameters. (See Procedures and Code Implementation)

Background and Theory:

1. Present and explain the key equations used to model the dynamics of the agents.

   See equations 1-9, 11-24 and their explanations.

2. Determine the terminal velocity of the agents.

   The terminal velocity of the agents is the highest achievable velocity by a drone determined by the force balance. The terminal velocity is given by a force balance of Ftot (the total or net force) = ma when the acceleration is 0. Hence the terminal velocity occurs when the net force = 0, and -Fp,i = Fd,i from Equation 5.

   Terminal velocity (vterminal):

   $$vterminal = ((2F_{p,i}\boldsymbol{n}_i^*)/(\rho_a C_{d,i} A_i) + v_a^2)/(2v_a - 1) \tag{29}$$

3. Discuss the use of Forward Euler time discretization. What trade-offs does it have?

   The use of Forward Euler time discretization is to calculate the velocities and the positions at a later time for each agent i, given the velocity/position at the current time, the Δt, the mass, and the net force on the agent. We are trying to integrate the total or net force equation to a final time that we have defined as tf through the Forward Euler equations. The benefits of this method are that in the two Forward Euler equations for updating the position and velocity, all the quantities are known. To calculate the velocity at a later time, we know from the previous iteration the velocity at a time t, our Δt is set to 0.2, the mass of each drone is constant at 10, and we know the net force at v(t), t. To calculate the position of an agent i at a later time we know its previous position at a time t from the previous iteration and we know its velocity at a time t. The advantage of the Forward Euler method is that it gives an explicit update equation. However some disadvantages of this method is that its approximation error is proportional to the time step size Δt. So good approximation can only be obtained with a very small Δt, otherwise the algorithm will require a larger amount of time discretization leading to a larger computation time.

4. What are the advantages and disadvantages of using a larger value for Δt?

   The disadvantages of using a larger value for Δt is that decreases the accuracy of the Forward Euler method for updating our positions and velocities of each agent i. The error in the method is roughly proportional to the step size Δt, at least for fairly small values of the step size. Increasing the Δt would

mean that we would need many more iterations to obtain the same level of accuracy for updating our positions and velocities. This large number of steps would entail a high computational cost. The advantages of using a higher value for Δt is that roundoff errors may decrease which will improve accuracy. Additionally, the estimation of the discretization error will not be as computationally expensive if the time step sizes are smaller than the optimum step size.

5. What must be true for the point-mass idealization to be reasonable? What is left out?

   Throughout the simulation, the agents are assumed to be small enough to be considered (idealized) as point-masses. It must be true that the effects of their rotation with respect to their mass center is considered unimportant to their overall motion. So we leave out the effects of rotation and we treat the point-mass agents as dimensionless, not taking up space. Spatial extension of each point-mass is left out.

6. How would adding net gravity, lift, or buoyancy complicate this problem?

   Adding net gravity, lift, or buoyancy would complicate this problem by impacting the point-mass idealization discussed above. The simulation neglects the effect of these three factors. The net gravitational forces acting on a point mass cancel out, and if we add in net gravity we would have then have to take that into account in equation 5 which defines the net force to be the prop force + the drag force. Similarly we would also have to take into account the lift in Equation 5, as it contrasts with the drag force. Buoyancy would depend on the density of the objects, and so we would have to consider that also when thinking how the buoyant force would have to be accounted for in Equation 5.

Procedure and Methods:

1. Comment on the specific cost function chosen for this problem. Why are gradient-based methods poorly suited to optimizing it (even numerically)? Why is a GA a good choice?

   The specific cost function chosen for this problem is presented by equation 25. The weights are given in the equation and the cost of each generation in the genetic algorithm is calculated by multiplying these weights by the performance of the simulation in terms of mapped targets, used time and crashed agents and adding these factors. The function penalizes agents that crash with other agents and with obstacles, rewards agents that quickly map targets, and penalizes lost agents. The function output (the cost) is most heavily affected by the number of unmapped targets, then by the number of crashed agents, and then by the time the simulation takes to complete. The cost function is non convex in the design parameters space and is non smooth.
   Gradient based methods are poorly suited to optimization because the behavior of the objects is complex, and the function is non convex and non smooth. The minimization of a non convex function is difficult with gradient methods as it can get stuck or veer off in the wrong direction, and it is more computationally expensive even numerically. The GA is a good choice because it is a non derivative search method. It is the simplest scheme to optimize the cost function, as it is well suited for non convex, non smooth, multi component functions.

2. Comment on the choice of control model. Is there any rigorous, provable reason to use this framework over something else? Why might we expect it to work for some parameter values?

   The benefit of using a GA to determine the parameter values for this particular model is as mentioned above, the cost function is non convex and non smooth and the GA handles this model very well. The benefits of using it to minimize the cost function is that it uses the objective function information and not derivatives (in calculating and updating the positions and velocities for each agent i in each new iteration). It is good for a noisy environment like this, where the drones will need to traverse the domain to map the targets and avoid the obstacles and each other. It is robust with regards to local minima and it is stochastic. There are many reasons to use this framework over other search and optimization methods. The GA searches parallel from a population of points, so it has the ability to avoid being trapped in a local optimal solution like traditional methods that search from a single

point. The GA uses probabilistic selection rules, not deterministic ones. The algorithm works on the potential solution's parameters, rather then the parameters themselves, and the algorithm does not need to compute derivatives at any point.
However it does have some limitations in that designing an objective cost function like we did and getting the representation and operators right can be difficult. The GA is also computationally expensive in determining the parameter values for this model. The solution quality also may deteriorate with the increase of the problem size. The parameters of a GA like population size, generations, and others must also be chosen carefully as this all affects convergence to the optimal parameters.

3. What would happen if any of the a, b, or c design variables became negative? Would this be good or bad for the performance of the swarm?

   If any of the a, b, or c design variables became negative, this would be bad for the performance of the swarm. The cost produced by the genetic algorithm would start off much higher than usual and would take much longer to converge to a minimum. We could not guarantee that the algorithm would converge within a certain time if we made any of these design variables negative.

4. Explain your strategy for "removing" mapped targets and crashed agents.
   My strategy for removing mapped targets and crashed agents is shown below:

```
mtHit = np.where(mtDist < agent_sight)
moHit = np.where(moDist < crash_range)
mmHit = np.where(mmDist < crash_range)
# check for lost agents

xLost = np.where(np.abs(posM[:, 0]) > xmax)
yLost = np.where(np.abs(posM[:, 1]) > ymax)
zLost = np.where(np.abs(posM[:, 2]) > zmax)

xy = np.append(xLost[0], yLost[0], axis = 0)
xyz = np.append(xy, zLost[0], axis = 0)
mLost = np.unique(xyz)

tarMapped = np.unique(mtHit[1])

a = np.append(mmHit[0], moHit[0], axis = 0)
b = np.append(a, mLost)
mCrash = np.unique(b)

# remove crashed agents
posM = np.delete(posM, (mCrash), axis=0)
velM = np.delete(velM, (mCrash), axis=0)
n_t = n_t - len(tarMapped)
n_m = n_m - len(mCrash)

mtDist = np.delete(mtDist, (mCrash), axis=0)
mtDiff = np.delete(mtDiff, (mCrash), axis=0)
mmDist = np.delete(mmDist, (mCrash), axis=0)
mmDist = np.delete(mmDist, (mCrash), axis=1)
mmDiff = np.delete(mmDiff, (mCrash), axis=0)
mmDiff = np.delete(mmDiff, (mCrash), axis=1)
moDist = np.delete(moDist, (mCrash), axis=0)
moDiff = np.delete(moDiff, (mCrash), axis=0)


posTar = np.delete(posTar, (tarMapped), axis=0)
mtDist = np.delete(mtDist, (tarMapped), axis=1)
mtDiff = np.delete(mtDiff, (tarMapped), axis=1)
```

   First I used mtDist, moDist, and mmDist calculated from the previous for loop. These 2 dimen-

sional matrices contain the normalized distance between each agent and target, agent and obstacle and agent and agent respectively. I checked where these matrices have distances that are less than agent sight for mtHit, and distances that are less than crash range for moHit and mmHit. I did the same thing for xLost, yLost and zLost. I checked where the values in the 1st column corresponding to the x values on the position matrix are out of the domain, and I did the same for the y values and z values. The function np.where returns the row and column indices for the values that satisfy the boolean expression. To determine which agents were lost, I indexed into the first element of xLost, yLost, and zLost to get just the row indices for the agents that went out of the domain, since even if only the x coordinate of an agent is outside its domain the agent is lost. I used np.append and np.unique to create mLost, an array with row indices of the lost agents.
Then for the mapped targets, I indexed into the second element of mtHit to access the column indices for the agents that successfully mapped their targets and assigned that to tarMapped. To calculate mCrash, I appended the row indices of mmHit, moHit and lastly mLost to create an array of unique row indices for agents that either crashed into other agents, obstacles, or became lost.

To remove the crashed agents I used mCrash and np.delete to delete the rows in the position matrix that corresponded to "crashed" agents, and also deleted the corresponding velocities in the velocity matrix. I updated nt and nm accordingly. Then I similarly used np.delete to delete rows (and columns for mmDist and mmDiff) in mtDist, mtDiff, mmDist, mmDiff, moDist, and moDiff. I used tarMapped to delete rows from the target array that corresponded to mapped targets and deleted rows from mtDist and mtDiff that corresponded to the mapped targets.

5. Explain how you will initially place the agents.

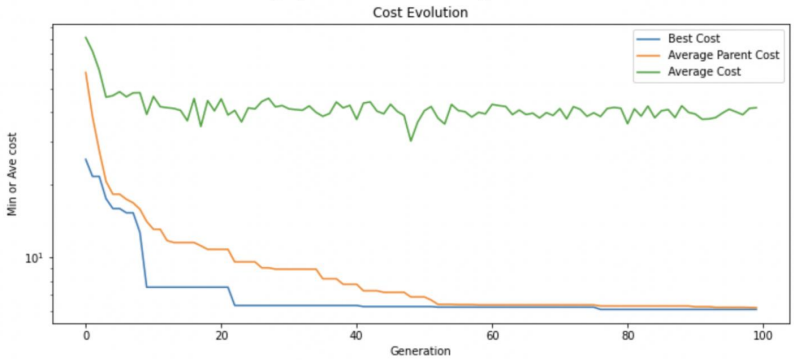
   I initially placed the agents by creating a random arrangement at the beginning of the trial to ensure that they begin within our domain and are far enough apart so as not to crash into each other immediately. In our setup, the agents begin only on the x-y plane, as their z coordinates are all 0. All their x coordinates are 142.5, and all their y coordinates are linearly spaced within the y domain. I used np.linspace, np.zeros, and np.array for setting up this array.

6. Explain any other significant choices you made in creating your code. If you have gone above and beyond in improving your genetic algorithm, explain what you implemented and why.

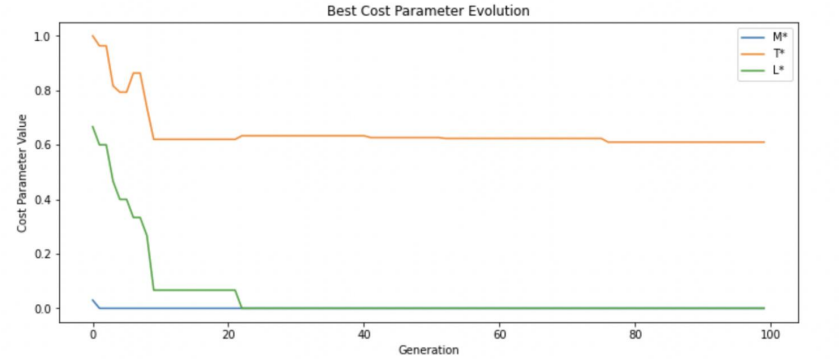
   I used a while loop in my drone simulation since I wanted to keep track of each iteration more explicitly than I could in a for loop. I also created the 5 time spaced visualizations of the simulation after calling the genetic algorithm and drone simulation, to ensure I was not affecting the position or target arrays in the simulation. I used np.delete to delete rows and columns from the position and target arrays instead of filling them with nan values.

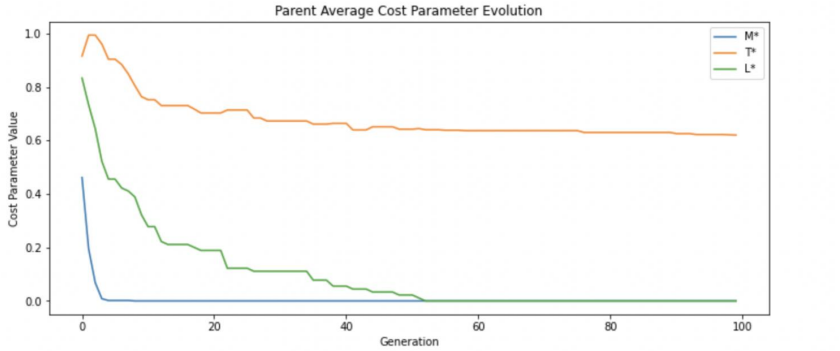
Results and Discussion:

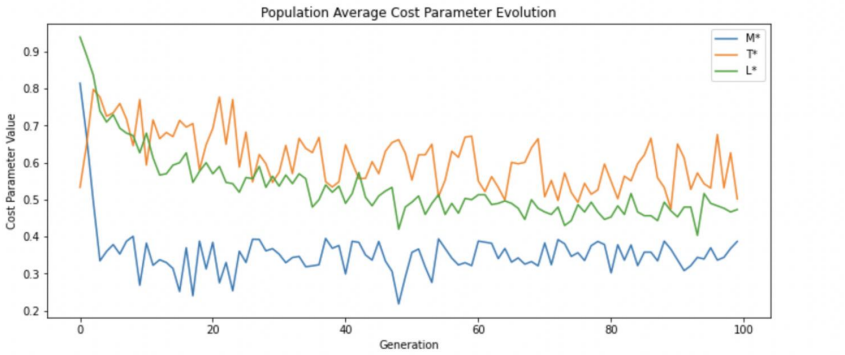
1. Provide a convergence plot showing the total cost of the best design, the mean of all parent designs, and the mean of the overall population for each generation.



2. Provide a plot showing the individual performance components (i.e., plot M^*, T^*, and L^*), for the overall best performer, mean parental population, and overall population. Discuss any important observations.







I noticed that across the generations, the cost parameter value of M^* is the lowest, L^* cost value is higher, and T^* is the highest. This is true for the overall best performer, mean parental population, and overall population. Additionally for the best cost parameter evolution and the parent average cost parameter evolution, the values of M^*, T^*, and L^* follow smoother curves and change gradually. However for the population average cost parameter evolution, the values change drastically and the curves are not smooth at all.

3. Report your best-performing 4 designs in a table similar to the following.

9]:

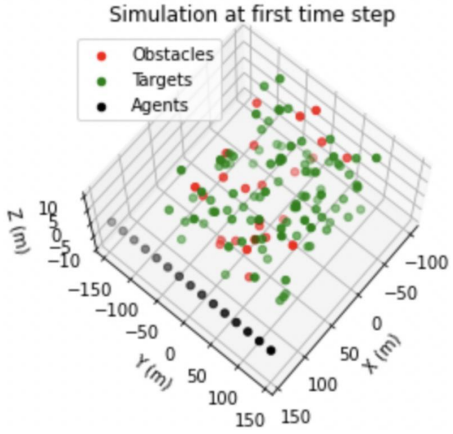
	Wmt	Wmo	Wmm	wt1	wt2	wo1	wo2	wm1	wm2	a1	a2	b1	b2	c1	c2	Pl
1	0.984374	1.585350	0.692763	1.244505	1.815098	1.063979	1.010151	1.497160	0.435556	0.119423	0.813796	1.018599	0.600427	1.604196	0.454480	6.1
2	0.984798	1.586496	0.692760	1.244508	1.815205	1.063185	1.010152	1.497095	0.435790	0.119417	0.813864	1.018554	0.602422	1.604213	0.452394	6.2
3	0.984680	1.586330	0.692764	1.244503	1.815740	1.063695	1.010159	1.498010	0.435499	0.119422	0.813877	1.018590	0.600861	1.604213	0.455744	6.2
4	0.984782	1.586444	0.692764	1.244516	1.814995	1.063598	1.010159	1.497062	0.435551	0.119422	0.813929	1.018571	0.601140	1.604201	0.454896	6.2

4. Discuss the results. How much variation does each parameter have between your top performers? Do any parameters have negative values? Are any values very near zero? Is anything else notable about the results and the behavior they should create?

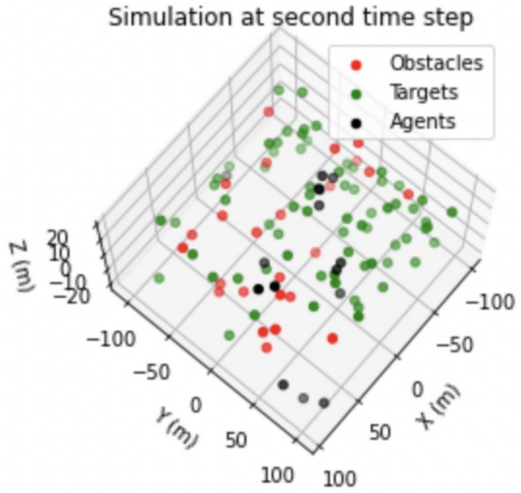
   Between my top parameters, each parameter does not have much variation. In my top four, each of the fifteen parameters are quite close to each other as is the cost. None of the parameters have any negative values. There are some values that are near zero. For example the a1 parameter values are close to 0. The parameter values seem to be in between 0 to 2. We can see that the top performing costs are all the lowest costs from the genetic algorithm output, in order from lowest cost to highest cost. The minimum cost ranges to be somewhere around 6.

5. Include a series of plots showing how your best design moves the agents. Include 5 approximately evenly-spaced frames from $t = 0$ until the final time for your system. Agents, targets and obstacles should all be clearly shown with distinct marker styles, axes should be labeled and the title should contain the time at which the snapshot occurs.

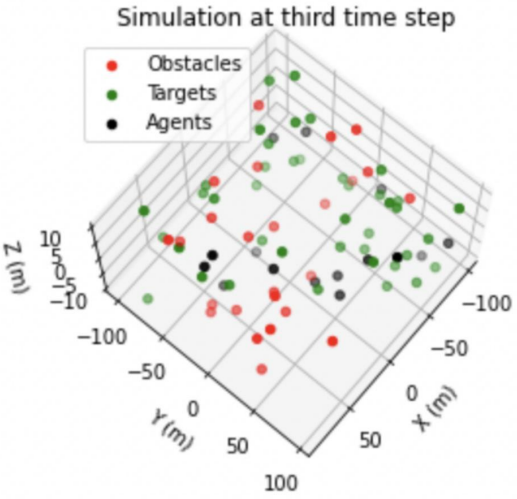
12]: `Text(0.5, 0.92, 'Simulation at first time step')`



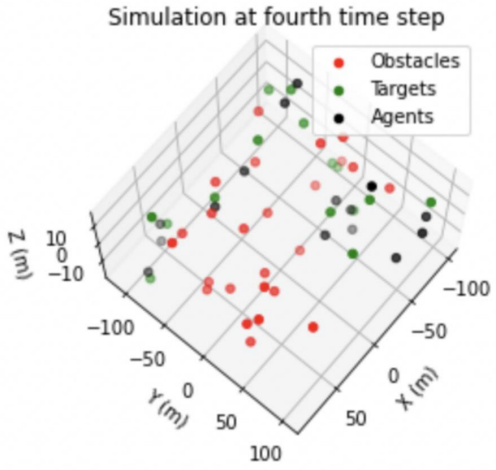
]: Text(0.5, 0.92, 'Simulation at second time step')



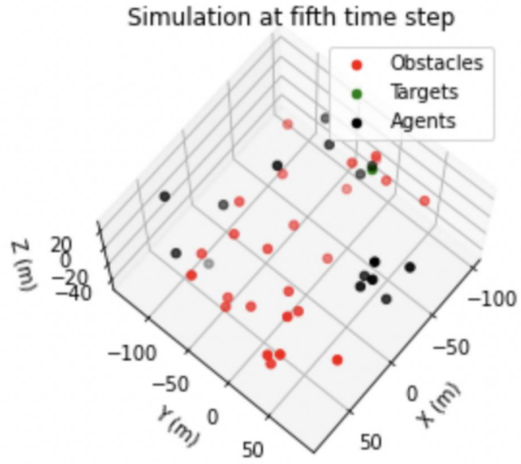
]: Text(0.5, 0.92, 'Simulation at third time step')



]: Text(0.5, 0.92, 'Simulation at fourth time step')



: Text(0.5, 0.92, 'Simulation at fifth time step')



Conclusion:

1. Briefly summarize your goals, methods, and key results.

   In this project we implemented a drone simulation function and a genetic algorithm function to optimize a set of fifteen parameters that would produce a simulation of a set of drones that most efficiently maps a set of targets. In order to map this region we initialized the locations of the targets, the locations of the obstacles, and the locations of the agents. For each agent we then determined the distance and directed normal to each target, obstacle, and other agent. For each agent we determined the interaction functions. For each agent we also determined the net force acting upon it, and we integrated the equations of motion to produce its position and velocity. We determined if any targets have been mapped by checking the distance between each agent and target (if less than a given a tolerance). If any targets satisfied this criteria we took them out of the system for the next time step

so that no agents resources are wasted by attempting to map them. We do the same for agents and obstacles, except we don't remove the obstacles as they are stationary. We do the same for agent and other agent collisions, and lost agents, removing them from the system. We repeat the process for the next time step. This is the essence of the drone simulation function.
In the genetic algorithm function we generated a system population of 15 parameters, computed the fitness/performance of each genetic string and ranked them, and mated the pairs and produced offspring. We then eliminated poorly performing genetic strings, keeping the top parents and genetic offspring. We then repeated the process for the new gene pool and new random genetic strings for the next time step, until we converge to a minimum cost for the objective function. We make sure to call the drone within the genetic algorithm function.
Our results show that our genetic algorithm model tends to converge rather quickly to a minimum cost in the range of 5-8, usually around 6. Our plots show the method in which the agents cover the area and map targets while avoiding obstacles. Overall the algorithm works efficiently to model this drone problem.