UNIVERSITY OF CALIFORNIA, BERKELEY

Miyuki Weldon

E 150 Basic Modeling and Simulation Tools for Industrial Research Applications

Instructor: Tarek Zohdi

Modeling and Simulation of Swarms of UAVs

1 Introduction

In nature, we often see swarms of flying animals, for instance bees or birds, traveling in unlikely patterns to avoid each other and obstacles to reach an objective location. Engineering can be used to model this swarm like behavior with unmanned aerial vehicles (UAVs) or in our case, drones. The successful modeling of this behavior can be used to scope out dangerous situations that are not safe for humans as well as provide more efficiency in other time sensitive tasks. The lives of firefighters can be saved by using UAVs to map the fires. Additionally, UAVs can be used to monitor large solar fields to determine when it is necessary to send technicians. To solve this complex problem, I will examine the dynamics and interactions of each swarm member along with a machine learning algorithm to determine the most optimal direction for each swarm member to travel in at each point in time. The simulation discussed in this report will find the correct parameters for 15 UAVs to take pictures of 100 targets while avoiding 25 obstacles whose locations will be randomly generated. This specific situation has many idealities, but provides the foundation to run more complex simulations.

2 Background and Theory

Dynamics of Each Swarm Member

To model the dynamics and interactions of each swarm member, we look to Newtons second law $\boldsymbol{F} = m\boldsymbol{a}$ or in our notation $m\dot{\boldsymbol{v}} = m\ddot{\boldsymbol{r}} = \boldsymbol{\Psi}$. Bold symbols are used to denote vectors.

For each swarm member we can find its position, velocity, and acceleration using vector equations from a predetermined origin.

$$\begin{aligned} \text{Position:} \quad & \mathbf{r_i} = r_{i1}\mathbf{e_1} + r_{i2}\mathbf{e_2} + r_{i3}\mathbf{e_3} \\ \text{Velocity:} \quad & \mathbf{v_i} = \dot{\mathbf{r}}_\mathbf{i} = \dot{r}_{i1}\mathbf{e_1} + \dot{r}_{i2}\mathbf{e_2} + \dot{r}_{i3}\mathbf{e_3} \\ \text{Acceleration:} \quad & \mathbf{a_i} = \dot{\mathbf{v}}_\mathbf{i} = \ddot{\mathbf{r}}_\mathbf{i} = \ddot{r}_{i1}\mathbf{e_1} + \ddot{r}_{i2}\mathbf{e_2} + \ddot{r}_{i3}\mathbf{e_3} \end{aligned}$$

Characterization of Interactions:

N_m = number of swarm members, N_t = number of targets, N_o = number of obstacles

Member-Target Interactions

Distance from swarm member i to target j where $\mathbf{T}_j$ is its position

$$\|\mathbf{r}_i - \mathbf{T}_j\| = ((r_{i1} - T_{j1})^2 + (r_{i2} - T_{j2})^2 + (r_{i3} - T_{j3})^2)^{1/2} = d_{ij}^{mt}$$

The normal vector of the direction and its weighted counterpart is defined by

$$\mathbf{n}_{i\to j} = \frac{\mathbf{T}_j - \mathbf{r}_i}{\|\mathbf{r}_i - \mathbf{T}_j\|} \qquad \hat{\mathbf{n}}_{i\to j} = (w_{t1}e^{-a_1 d_{ij}^{mt}} - w_{t2}e^{-a_2 d_{ij}^{mt}})\mathbf{n}_{i\to j}$$

where w_{t1} and w_{t2} are weights to rank the importance of each target and a_1 and a_2 are reflect the decay of the exponential functions. The weighted normal vectors are then summed to compute an overall direction to travel with respect to the target locations

$$\mathbf{N}_i^{mt} = \sum_{j=1}^{N_t} \hat{\mathbf{n}}_{i\to j}$$

Repeat this process for member-obstacle interactions and member-member interactions with their respective weights w_{o1}, w_{o2}, b_1, b_2 and w_{m1}, w_{m2}, c_1, c_2 to find $\mathbf{N}_i^{mo}$ and $\mathbf{N}_i^{mm}$ (where in the member-member case $j \neq i$).

Summation of All Interactions

To find the overall best direction for the swarm member to travel in, the overall directions for each type of interaction are weighted and summed to find the best direction and its normalized counterpart

$$\mathbf{N}_i^{tot} = W_{mt}\mathbf{N}_i^{mt} + W_{mo}\mathbf{N}_i^{mo} + W_{mm}\mathbf{N}_i^{mm} \qquad \mathbf{n}_i^* = \frac{\mathbf{N}_i^{tot}}{\|\mathbf{N}_i^{tot}\|}$$

The total forces can be computed by combining the drag force $\mathbf{F}_{drag} = \frac{1}{2}\rho_a C_{d,i} A_i \|\mathbf{v}_a - \mathbf{v}_i\|(\mathbf{v}_a - \mathbf{v}_i)$ and the non controllable (for this situation) thrust force in the direction of our normalized overall direction vector.

$$\mathbf{\Psi}_i^{tot} = F_i \mathbf{n}_i^* + \mathbf{F}_{drag}$$

Using the Forward Euler method described in the next section, the velocity and position of each swarm member can be updated.

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{\Delta t}{m_i}\mathbf{\Psi}_i^{tot}(t) \qquad \mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t)$$

In the event that the updated velocity is larger than the maximum velocity v_{max} that the swarm member can handle, replace the updated velocity with

$$\mathbf{v}_i^{new}(t + \Delta t) = v_{max}\frac{\mathbf{v}_i^{old}(t + \Delta t)}{\|\mathbf{v}_i^{old}(t + \Delta t)\|}$$

In the situation of this project, the drag forces prevented the velocity from going above v_{max}. In fact we can use the drag force to determine the terminal velocity of the swarm members using

$$v_{terminal} = \sqrt{\frac{2mg}{C_d \rho_a A}}$$

Using the numbers given in the project statement we calculate that for a single swarm member $v_{terminal} = 29.3 m/s$.

Remove any targets that have been mapped by checking the distance between each swarm member and target. Remove any crashed swarm members by measuring the distance between swarm members and obstacles as well as swarm members with each other.

$$\|\mathbf{r}_i - \mathbf{T}_j\| \leq Tol \qquad \|\mathbf{r}_i - \mathbf{O}_j\| \leq Tol \qquad \|\mathbf{r}_i - \mathbf{r}_j\| \leq Tol$$

Repeat these processes for the next time step.

Euler's Method

We use the Forward Euler time discretization to find $\mathbf{r}(t + \Delta t)$ and $\mathbf{v}(t + \Delta t)$ using discrete points without having to solve the equations analytically. While it is not as accurate as other differential equation solvers such as Fourth Order Runge Kutta method it is quick and has little error for small time steps Δt. Larger Δt's allow the code to run quicker, however, they carry much more error with them and will less accurately predict the next location and speed of the UAV.

Idealizations

The dynamics of this systems treats each swarm member as a point mass rather than an object with a complex shape and nonuniform mass. To take into account these complexities we would have to use the moment of inertia tensor, increasing the time for the code to run and requiring a much more complex problem set up.

The total force $\mathbf{\Psi}_i^{tot}$ is currently calculated only using the thrust force provided by the UAV and the drag force when the air has zero velocity, but there are many other forces that could be taken into account. For instance, net gravity, lift, or buoyancy could complicate our $\mathbf{\Psi}_i^{tot}$. Lift and buoyancy would require us to take into account the changing properties of the air as well as the geometry of the UAV. Net gravitational forces require us to examine the Earth's gravitational force in addition to the gravitational forces of objects around it. The simplifications of this project are meant to be built upon with more complex situations.

3 Procedure and Methods

To find the most optimal path for the UAVs to follow to map all targets, we must find the appropriate values for all of the weights. Through machine learning, we can use a genetic algorithm to find the best design vector

$$\mathbf{\Lambda} = [W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2]$$

by minimizing the cost function $\Pi(\mathbf{\Lambda})$.

The genetic algorithm minimizes the cost function by randomly generating design vectors, ranking their costs, and mating the top few performers while replacing all low performers with new randomly generated strings. Although that seems inefficient, normal gradient based methods that are much more familiar and quicker have much stricter requirements. Gradient based methods find function minima by finding the root of a function's derivative. Even with numerical methods like Newton's method, the cost function is still required to be continuous and differentiable. Cost functions such as the one used in this project frequently have very jagged lines and corners as well as jumps and discontinuities. Therefore, the genetic algorithm is a much better choice for this problem.

The control model has no rigorously, provable reason that makes it better suited than other models nor will it necessarily converge. The smooth exponential functions perform well for the type of problem we are solving. The positive a, b or c design variables allow the exponentials to rapidly decay. Making those design variables negative could cause the exponential to rapidly increase for some values of d_{ij}.

To remove mapped targets and crashed agents, I recorded a boolean vector of called $rstore$ and one called $Tstore$ that were the length of the remaining swarm members and targets, respectively. After each time step, I recorded true values into the indicies of $rstore$ for crashed agents and $Tstore$ for mapped targets. I used those booleans to remove those swarm members and targets.

For the initial location of the swarm members, I placed them in an evenly spaced line

outside the volume that the obstacles and targets were randomly generated in to avoid crashes before the mapping has even begun.

To speed up the execution of this code, I evaluated the interactions of a single agent with all targets, obstacles, and other agents using vector operations. Each agent, target, and obstacle were stored in their own respective matrices with three columns representing their x, y, and z coordinates. To evaluate $\mathbf{N}_i^{mt}$, I created a N_t by 3 vector of the same x,y,z coordinates of the one agent in each row. I then found the vector $\mathbf{d}_i^{mt}$ of size N_t by 1 using vector operation instead of a for loop. I similarly calculated $\mathbf{n}_{i\to j}$ and $\hat{\mathbf{n}}_{i\to j}$ in N_t by 3 matrices, the latter of which I summed to find $\mathbf{N_i^{mt}}$. It improved my run time and fixed other errors I had in the code at the time.

4 Results and Discussion

After some debugging, my code was able to run successfully in about five minutes. Plots and data from one of my runs is shown below. Note that in this run, the targets were all mapped in the first generation, although that was not true for every run.

Table 1: Best Performing $\mathbf{\Lambda}$ Designs

	W_{mt}	W_{mo}	W_{mm}	w_{t1}	w_{t2}	w_{o1}	w_{o2}	w_{m1}	w_{m2}	a_1	a_2	b_1	b_2	c_1	c_2
1	0.9775	1.1784	1.3110	0.6495	0.7354	1.4593	0.8519	0.2235	0.7933	0.1410	0.8115	1.2405	1.0461	1.0392	0.9666
2	0.9359	1.2035	1.1564	0.6560	0.7497	1.4363	0.8113	0.2034	0.8236	0.1507	0.8249	1.4232	1.0420	1.0559	0.9929
3	0.9801	1.1951	1.3444	0.6524	0.7414	1.4783	0.8411	0.2342	0.8030	0.1415	0.8211	1.2500	1.0454	1.0403	0.9635
4	0.9477	1.2009	1.2626	0.6529	0.7440	1.4498	0.8261	0.2116	0.8163	0.1421	0.8249	1.2814	1.0438	1.0490	0.9828

From the top four performing design vectors, we can see that there are multiple parameters that have very different values among the four vectors. This demonstrates that there are no clearly correct answers for any of these variables, nor could they be determined by pure observation. None of the variables are very close to zero. The control law and genetic algorithm performed successfully because the design strings were able to direct the agents towards mapping all targets without any crashes in less than 30 seconds.

In $Figures 1-4$ below, we can see how the cost quickly decreased in the first few generations and then leveled out once all targets were mapped and no agents crashed. From that point the cost only slightly decreased as the mapping procedure became slightly quicker.

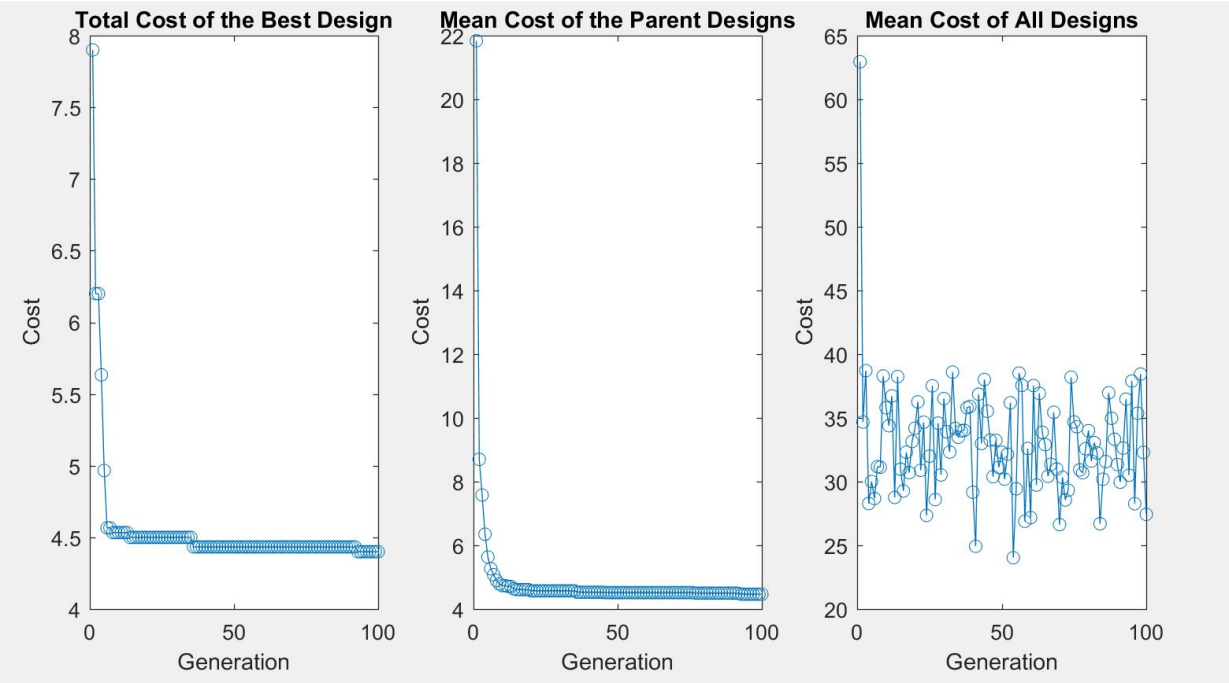


Figure 1: Convergence Plots for the Total Cost

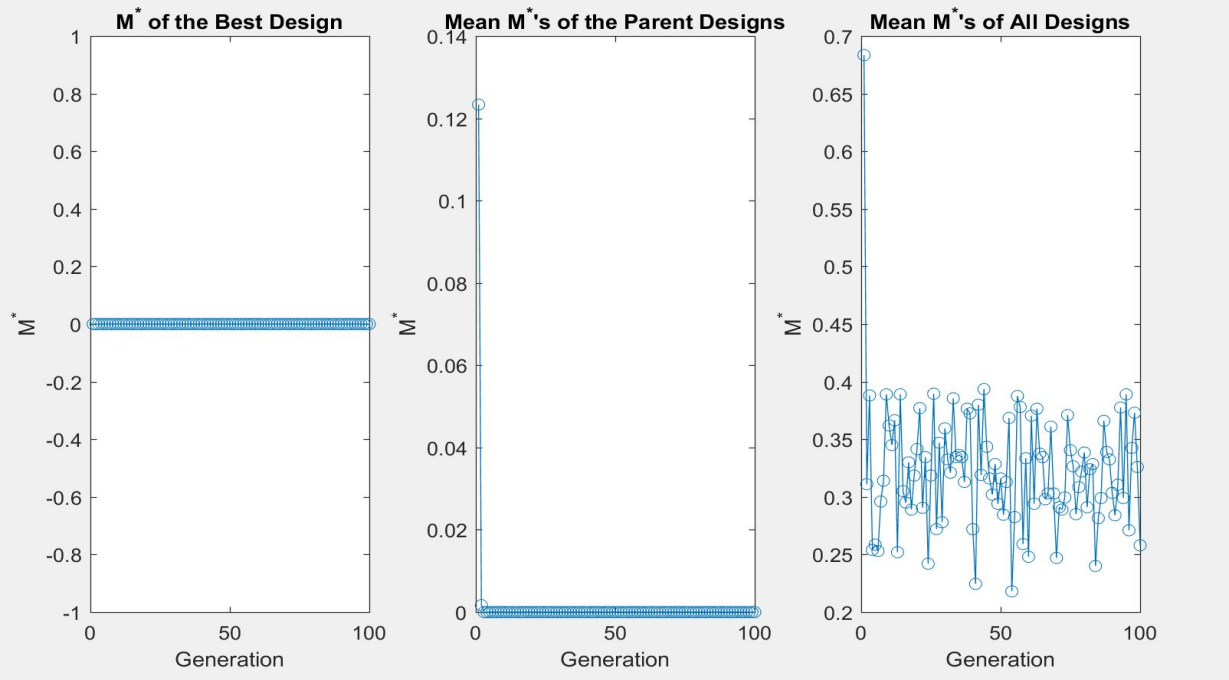


Figure 2: Performance Plots for $M^* = \frac{UnmappedTargets}{TotalTargets}$

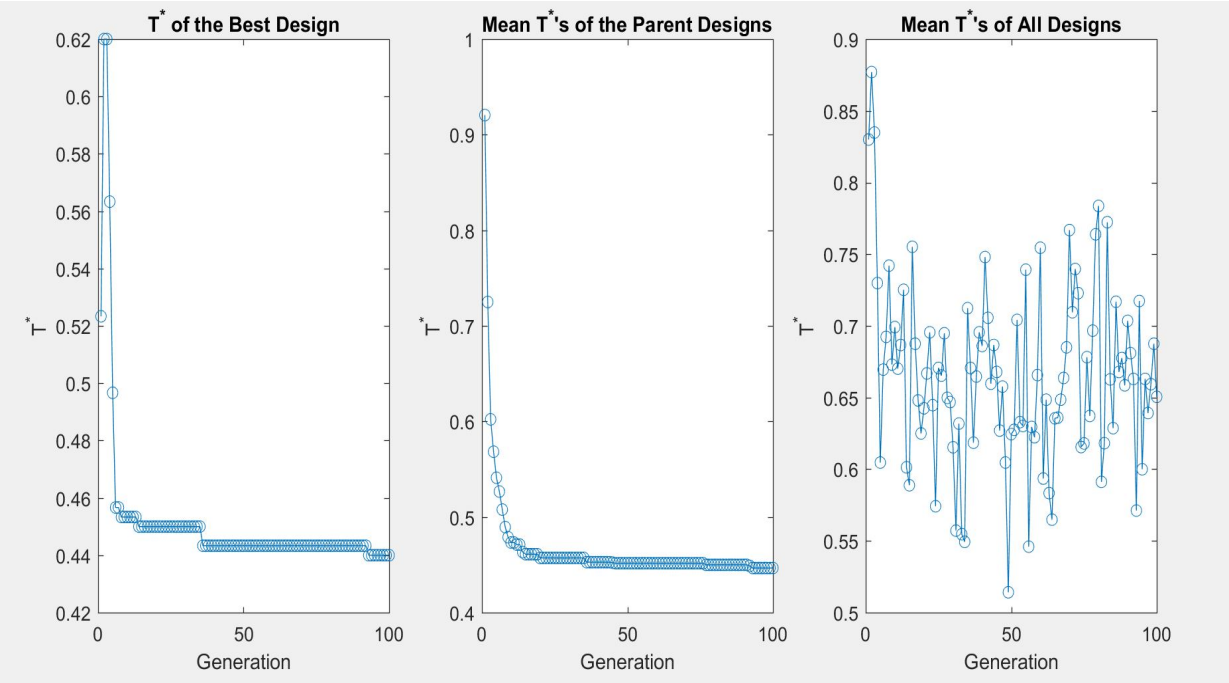


Figure 3: Performance Plots for $T^* = \frac{UsedTime}{TotalTime}$

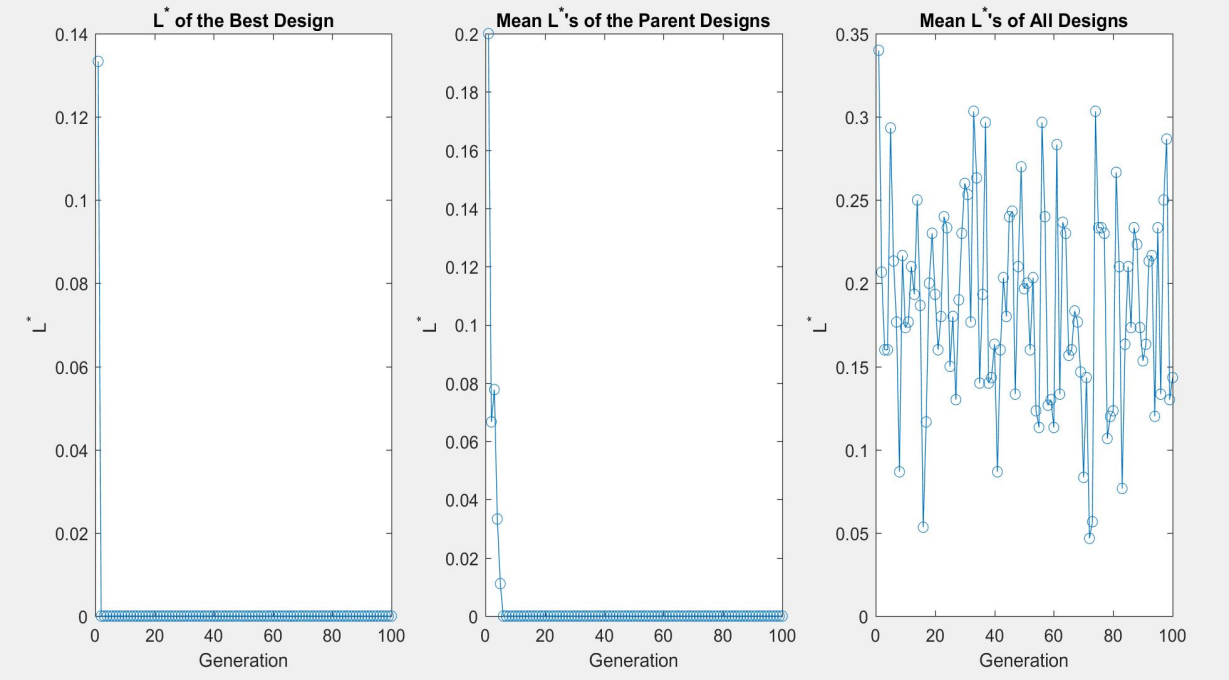
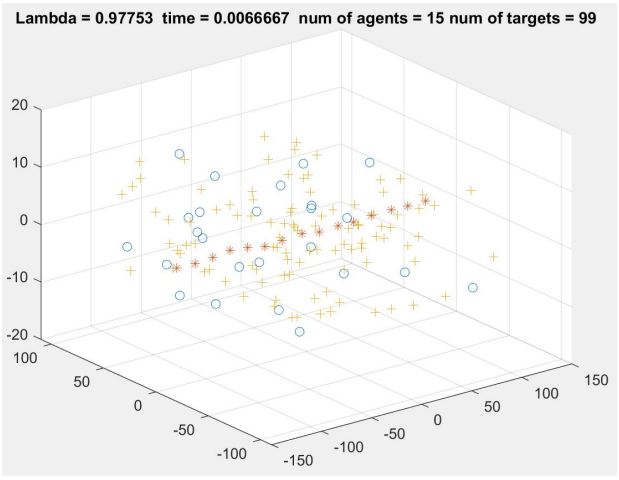
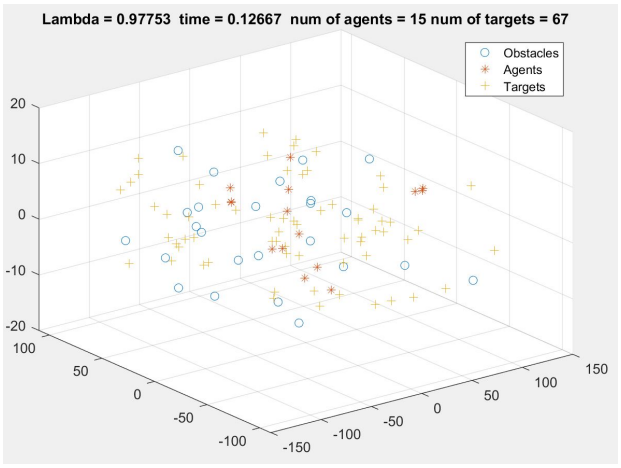


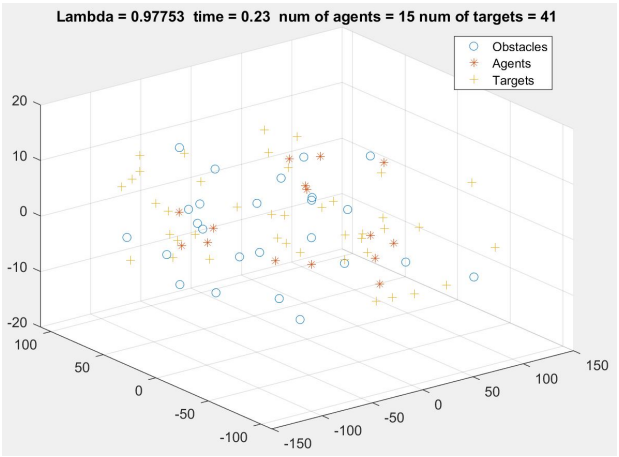
Figure 4: Performance Plots for $L^* = \frac{CrashedAgents}{TotalAgents}$



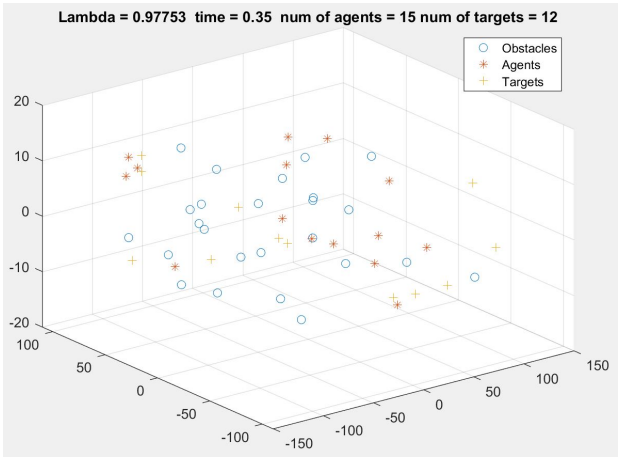
(a) Time = 0.4 sec



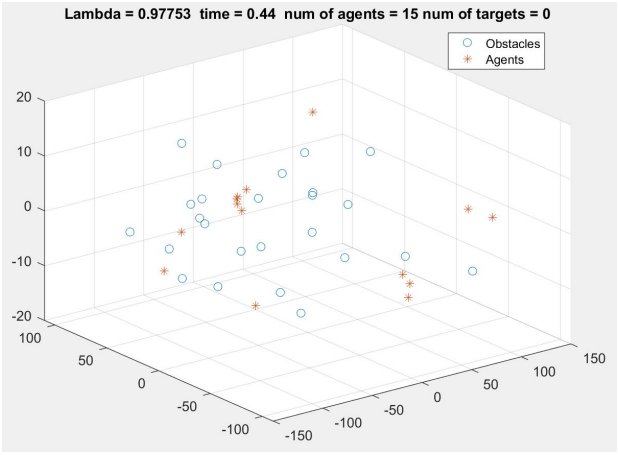
(b) Time = 7.6 sec



(c) Time = 13.8 sec



(d) Time = 21 sec



(e) Time = 26.4 sec

Figure 5: Animation Frames for Best Performing $\mathbf{\Lambda}$

5 Conclusion

In this project, we have seen that the complexities of swarm behavior can be modeled with simplified dynamics, a non unique control law, and a genetic algorithm. Using only these tools, I was able to determine an optimal path for the 15 agents to follow in order to map the 100 targets while avoiding 25 obstacles. Each successive generation improved how many targets were mapped and reduced how many agents had crashed until all targets were mapped with no crashes. The remaining generations reduced the amount of time taken. While an actual implementation of a swarm of drones will not match the model's prediction exactly, more complex methods can be used to improve the model. For instance, more complex dynamics that take into account the swarm member's geometry as well as the variable forces it may encounter can improve the model's prediction in addition to a better control law or a more efficient machine learning algorithm. While the success of this model teaches us quite a lot, there is much more to go to reach the success of bees and birds.