1 Introduction

UAV swarms have numerous applications in our modern world. They can facilitate mapping, observing, and overseeing large areas with ease. It is necessary for these agents to autonomously navigate potentially dangerous terrain and airspace in order to log information about areas. In this project we simulate the motion of a swarm of drones, and their interaction with obstacles, and targets.

The goal of this project is to optimize this simulation using a Genetic Algorithm to maximize the number of targets mapped, minimize the number of crashed drones, and minimize the amount of time used in the simulation. To do this we will use 15 design parameters to determine the dynamics and kinematics of each drone, using Forward Euler time discretization to update drone positions.

A Genetic Algorithm will be used to train this swarm of drones by generating random strings of design parameters, and breeding new strings to find the optimal string for this simulation.

In this paper, I will present relevant background information, and equations to describe the simulation. I will also describe the process by which I go about optimizing the Genetic Algorithm. Finally, I present the outcome of the simulation and Genetic Algorithm, and discuss the relevance of my results.

2 Background and Theory

The kinematics of each member is characterized by the following equations in a fixed Cartesian basis:

$$\boldsymbol{r} = r_1\boldsymbol{e_1} + r_2\boldsymbol{e_2} + r_3\boldsymbol{e_3}$$

$$\boldsymbol{v} = \dot{\boldsymbol{r}} = \dot{r_1}\boldsymbol{e_1} + \dot{r_2}\boldsymbol{e_2} + \dot{r_3}\boldsymbol{e_3}$$

$$\boldsymbol{a} = \ddot{\boldsymbol{r}} = \ddot{r_1}\boldsymbol{e_1} + \ddot{r_2}\boldsymbol{e_2} + \ddot{r_3}\boldsymbol{e_3}$$

The dynamics of each member is described by Newton's second law:

$$\boldsymbol{\Psi}_i^{tot} = m_i\boldsymbol{a}_i$$

The magnitude, and direction of the interaction force between an agent, i, and another object, j, is given by the equation:

$$\hat{\boldsymbol{n}}_{i\to j} = (w_1 e^{-x_1 d_{ij}} - w_2 e^{-x_2 d_{ij}})\boldsymbol{n}_{i\to j}$$

$$d_{ij}^{mo} = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|,\ \boldsymbol{n}_{i\to j} = \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{\|\boldsymbol{r}_j - \boldsymbol{r}_i\|}$$

The w1 term scales the force of attraction the member experiences from the object, and the w2 term scales the force of repulsion the member experiences from the object.

The total interaction vector on each member is the weighted summation of the interaction force between a member and all targets, a member and all obstacles, and a member and all other members.

$$\boldsymbol{N}_i^{mt} = \sum_{j=1}^{N_t} \hat{\boldsymbol{n}}_{i\to j}^{mt},\ \boldsymbol{N}_i^{mm} = \sum_{j=1,\ j\neq i}^{N_t} \hat{\boldsymbol{n}}_{i\to j}^{mm},\ \boldsymbol{N}_i^{mo} = \sum_{j=1}^{N_t} \hat{\boldsymbol{n}}_{i\to j}^{mo}$$

$$\boldsymbol{N}_i^{tot} = W_{mt}\boldsymbol{N}_i^{mt} + W_{mt}\boldsymbol{N}_i^{mm} + W_{mo}\boldsymbol{N}_i^{mo}$$

The direction of the force of propulsion experienced by the member is:

$$\boldsymbol{n}_i^* = \frac{\boldsymbol{N}_i^{tot}}{\|\boldsymbol{N}_i^{tot}\|}$$

And the total force, including drag, is:

$$\boldsymbol{\Psi}_i^{tot} = F_{p,i}\boldsymbol{n}_i^* + \boldsymbol{F}_{d,i}$$

$$\boldsymbol{F}_{d,i} = \frac{1}{2}\rho_a C_{d,i} A_i \|\boldsymbol{v}_a - \boldsymbol{v}_i\|(\boldsymbol{v}_a - \boldsymbol{v}_i)$$

Including the forces of lift, and buoyancy would require the orientation, shape, and size of each member to be taken into consideration, therefore we consider these forces as secondary, and leave them out in order to idealize the members as point-masses.

The propulsive force of each member is constant, therefore the terminal velocity can be calculated by equating the magnitude of the propulsive force and the magnitude of the drag force.

$$F_{p,i} = \frac{1}{2}\rho_a C_{d,i} A_i \|\boldsymbol{v}_a - \boldsymbol{v}_i\|^2$$

In our simulation, the air velocity is zero. Plugging in the values for each variable, and solving for terminal velocity yields:

$$v_{term} = \|\boldsymbol{v}_i\| = 36.140\ m/s$$

Once the total force on a member is known, its position and velocity are updated using the Forward Euler method.

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

Forward Euler time discretization is a useful tool for simulation. It will never lead to a perfect solution for the position of the members, but it allows us to avoid solving the complicated differential equations that are the member's kinematic equations. If the time step is small, the Forward Euler solution will be close to the actual value. A large time step will speed up our code, but will lead to a small number of discrete steps, and will cause a large amount of error.

3 Procedure and Methods

The cost function for this simulation is described by the following equation:

$$\Pi = 70M^* + 10T^* + 20L^*$$

$$M^* = \frac{\text{Unmapped targets}}{\text{Total targets}}, T^* = \frac{\text{Used time}}{\text{Total time}}, L^* = \frac{\text{Crashed agents}}{\text{Total agents}}$$

There are 15 design parameters involved in this simulation, so we chose to use a Genetic Algorithm to minimize the cost function. A gradient-based approach would result in a poorly optimized solution because there are likely many thousands of local minima in this 15-dimensional space. Gradient-based approaches will settle in the local minimum that is closest to the initial guess, making it extremely unlikely to find the global minimum.

A Genetic Algorithm is much better suited to this kind of optimization problem because it tests numerous randomly generated strings of design parameters against each other, and mates the top strings to create the next generation. Over many generations the best randomly generated, or bred strings will rise to the top, and plunge the depths of the cost function. Although the solutions aren't minima, they are more likely to reach a global minimum than a gradient-based solution.

The control-model we use in this project is relatively effective, but some parameters may be unnecessary. There may or may not be circumstances in which two members should feel an attractive force between them, or when a target should repel a drone, so further tests should be performed to see if some parameters can be left out or held constant during the Genetic Algorithm.

If any of the a, b, or c design variables became negative, the force between the member and object would become directly proportional to the distance between the two. This would be bad for the performance of the swarm because it would cause members to be more drawn to objects farther away than those close. As they get close to those far away objects, the force of attraction would become smaller, and would become overpowered by the ones that were initially closer.

My strategy for removing mapped targets and crashed agents was to move the agent, or target to Inf, and remove it from the position array at the end of the time-step.

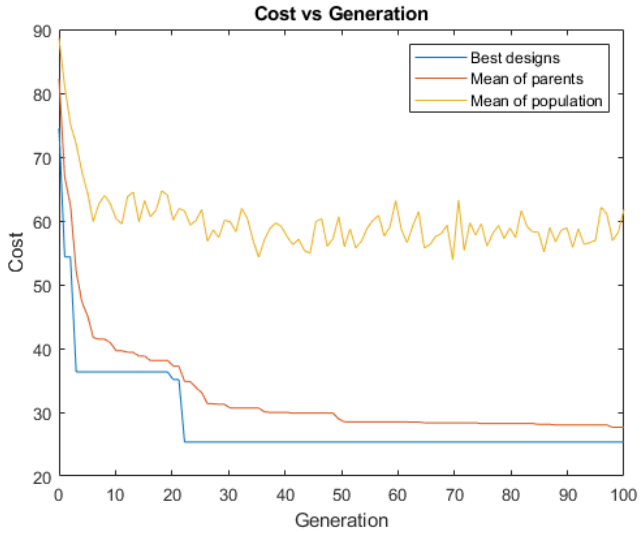
My initial strategy for removing mapped targets and crashed agents was to initialize boolean arrays called "crashed," and "mapped," as false, then change the index of the crashed member, or mapped target to true when they meet the criteria for crashing or mapping. I left the crashed members, and mapped targets in their position arrays, but skipped over any targets or members with a "true" value in their corresponding boolean array. However, this results in a sub-optimal simulation because the size of the position arrays we loop over never get any smaller. In the early generations when plenty of drones are crashing, and in the later generations

when most targets have been mapped the simulation will run much faster if the position arrays decrease in size.

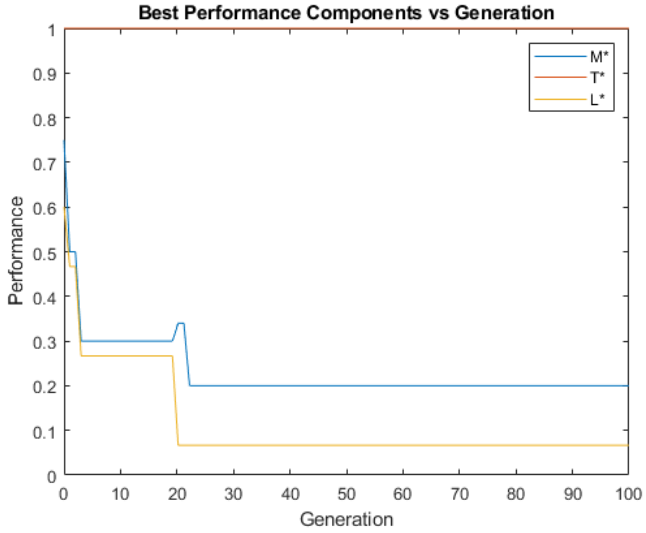
I initially place the agents as far apart as possible within the starting zone. If the members are far apart they are less likely to be drawn to the same target in the beginning of the simulation. If they all vie for the same target they are more likely to crash into each other. However, placing the members near each other has its benefits because the force of repulsion between members is higher, and they could potentially spread out more.

In order to speed up the Genetic Algorithm, I do not reevaluate the design parameter strings of parents from the previous generation. Instead I copy their cost function value into the next row of PI before evaluating their children, and the new, randomly generated strings.

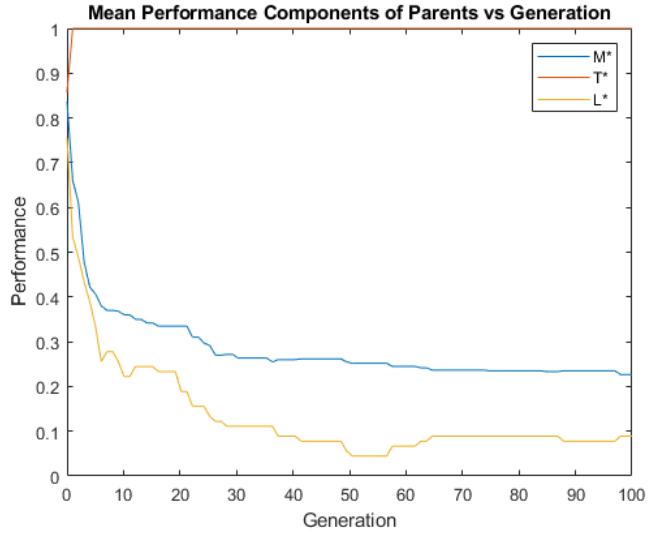
4 Results and Discussion

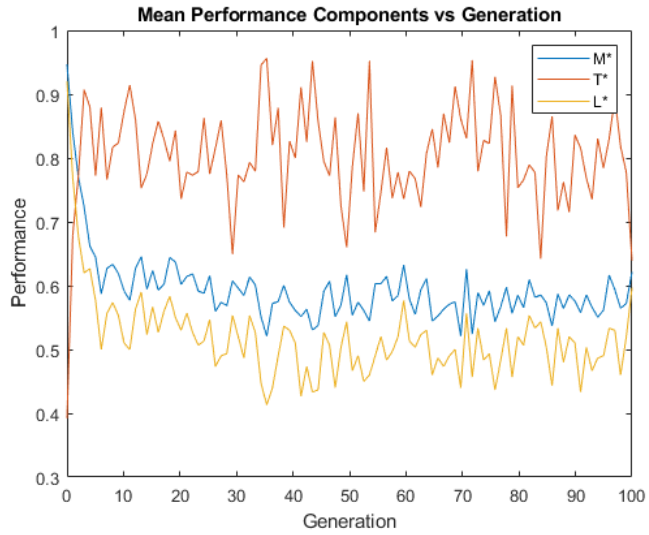


The best design converges on its final value around generation 20. This could be caused by inbreeding, wherein the best design parameters for multiple generations in a row are all related, and all converge to a steady state value. The only way to decrease the cost further would be for a randomly generated string to beat these strings.



The effects of inbreeding can be seen here as well. The performance components all plateau early on. If inbreeding were discouraged the performance would likely decrease a couple more times within the final 80 generations. The least successful part of this optimization was the amount of time used by the simulation. A potential way to decrease T* could be to weigh it higher in the cost function. There could also be some improvements made in the simulation to optimize time.





There is a slight decrease in the average performance of all strings across all generations. This is expected because most randomly generated strings, and many children perform poorly. Only a small number of strings in each generation perform very well.

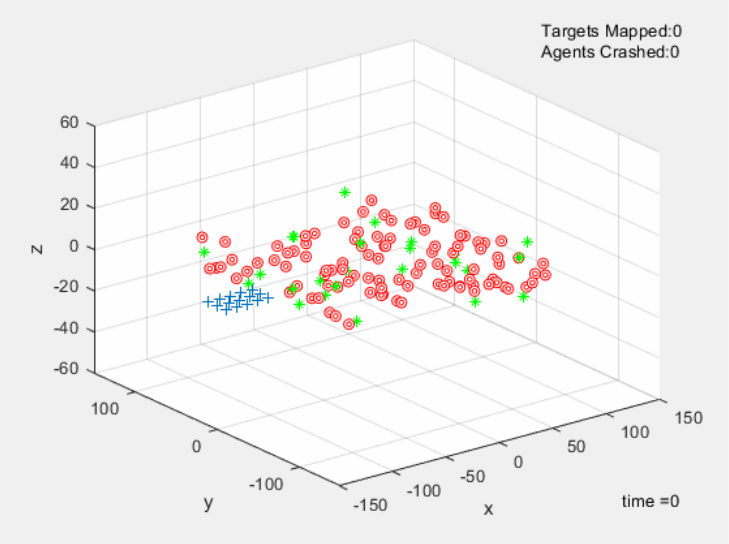
Des	1	2	3	4
Λ_1	1.3991	1.4329	1.3722	1.4000
Λ_2	0.9738	1.0017	0.9517	0.9745
Λ_3	1.3406	1.3727	1.3152	1.3415
Λ_4	1.1311	1.1660	1.1034	1.1321
Λ_5	0.3433	0.3446	0.3422	0.3433
Λ_6	1.1199	1.1536	1.0933	1.1209
Λ_7	1.0116	1.0353	0.9928	1.0122
Λ_8	1.0966	1.1311	1.0693	1.0976
Λ_9	0.9070	0.9324	0.8868	0.9077
Λ_{10}	0.0407	0.0414	0.0402	0.0407
Λ_{11}	0.3147	0.3175	0.3125	0.3148
Λ_{12}	0.3274	0.3298	0.3255	0.3275
Λ_{13}	1.3218	1.3576	1.2935	1.3228
Λ_{14}	0.4016	0.4123	0.3932	0.4019
Λ_{15}	0.1308	0.1339	0.1284	0.1309
Cost	25.3333	26.1000	26.8000	28.0000

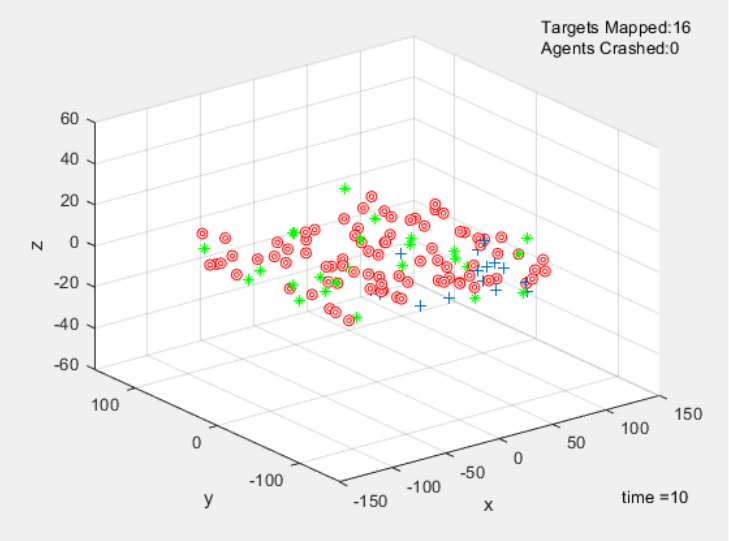
Table 1: Top 4 Design Parameter Strings

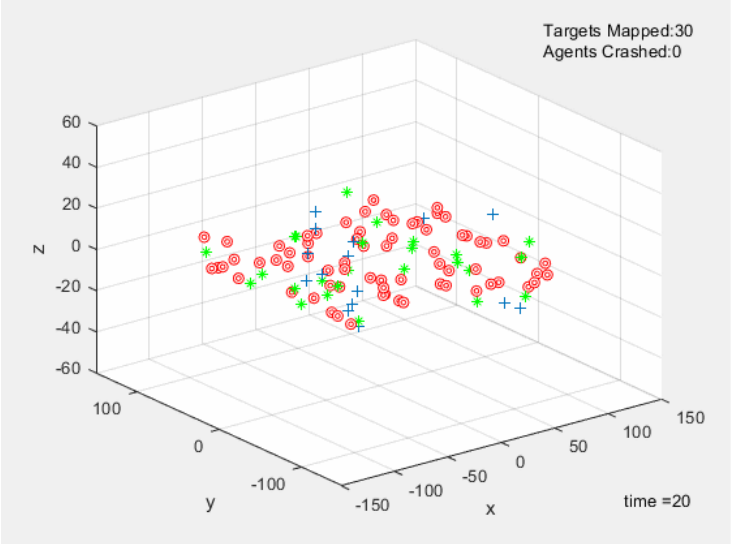
The top 4 design parameter strings all have similar design parameters. No negative values are present because the random strings are initialized between 0 and 2, and all children have parameters between their parents'

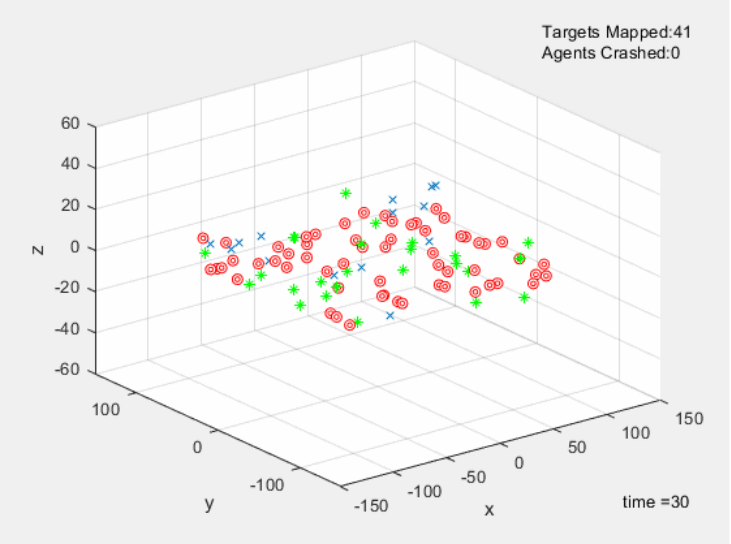
parameters. If any negative values were present the repulsive and attractive forces could be switched, and the relationship between distance and magnitude of force could be inverted. Design parameters 10, and 15 are the smallest, approaching 0. Respectively, these cause the members to feel a very strong attractive force from targets, and a strong repulsive force from other members. Many of the design parameters are around 1. These mostly include the w scalars in front of the exponentials in the attractive and repulsive forces. This could mean that these parameters are unimportant, and that the parameters in the exponentials have a greater effect on the simulation. For future testing we could try setting these at a constant value of 1, and only optimizing the other parameters.

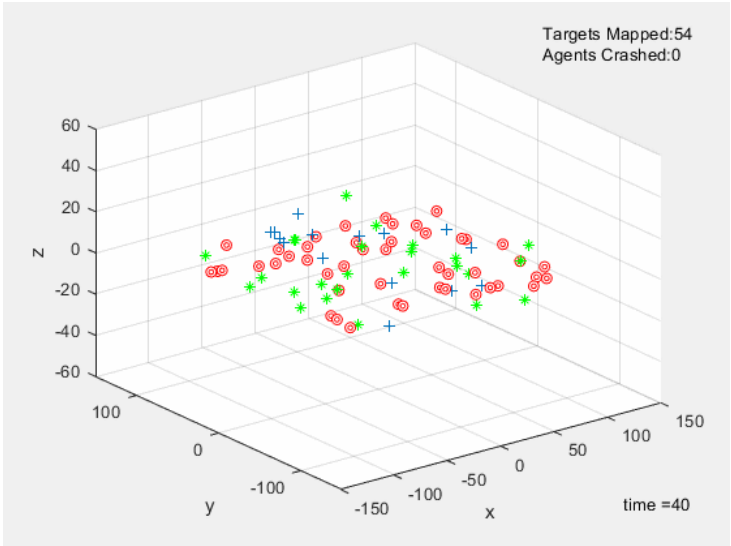
4.1 Simulation:

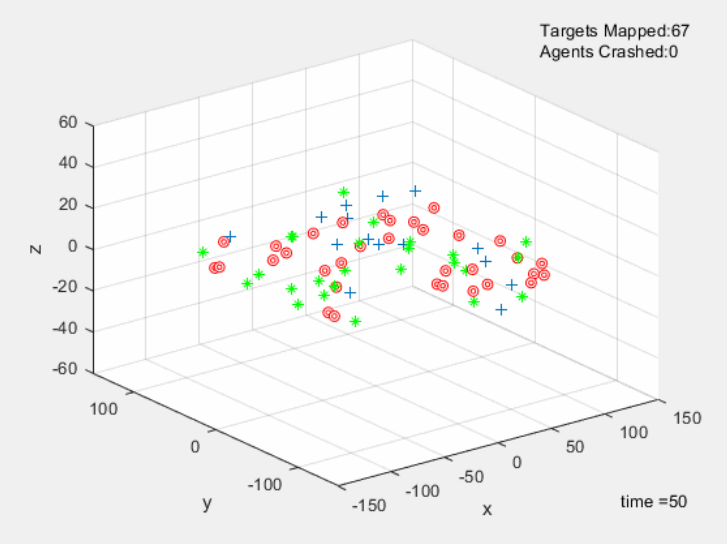


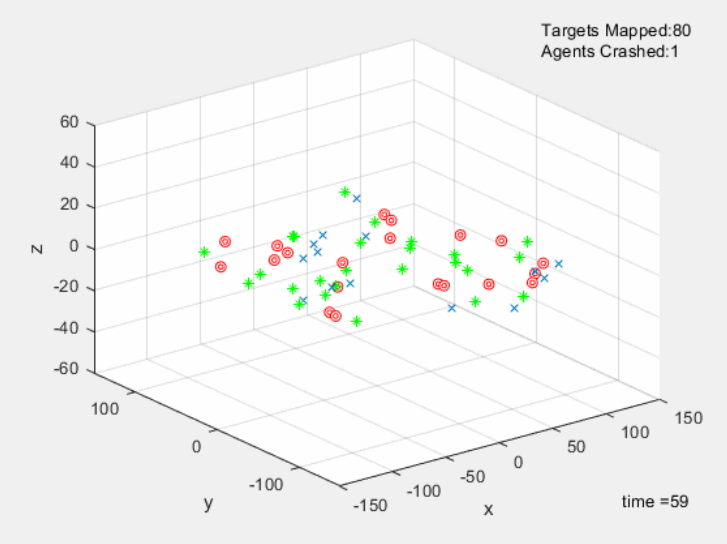



Targets Mapped:30
Agents Crashed:0
z
60
40
20
0
-20
-40
-60
y
100
0
-100
x
-150
-100
-50
0
50
100
150
time =20


Targets Mapped:41
Agents Crashed:0
z
60
40
20
0
-20
-40
-60
y
100
0
-100
x
-150
-100
-50
0
50
100
150
time =30


Targets Mapped:54
Agents Crashed:0
z
60
40
20
0
-20
-40
-60
y
100
0
-100
x
-150
-100
-50
0
50
100
150
time =40





5 Conclusion

The simulation was successfully optimized using a Genetic Algorithm. The dynamics, and kinematic equations of 15 UAVs were used to simulate the movements of these drones. Each drone was affected by a repulsive, and an attractive force from each other member, target, and obstacle. These equations utilized 15 design parameters that were optimized by a Genetic Algorithm. Random strings of design parameters were created at the beginning, passed into the simulator, and the resulting success of the strings was evaluated. The strings were then ranked, and the top strings were mated with each other, resulting in child strings. The next generation was filled out with more randomly generated strings, and the process was repeated for 100 generations. The GA resulted in a final cost of 25.3333. Additional optimizations for T*, and prevention of inbreeding could significantly decrease the cost, and should be implemented in future work on this, and other projects.