Genetic Algorithm Trajectory Optimizer for Unmanned Air Vehicle Obstacle Avoidance and Path Planning

Andrew Torgesen

***Abstract*—Path planning and obstacle avoidance given a set of waypoints is an important and common problem for autonomous, unmanned air vehicles. There is a large field of research dedicated to solving the problem of efficiently navigating an obstacle field to arrive at a specified location, with wide-ranging algorithmic solutions. This project aims to solve the problem for a small, fixed-wing unmanned air vehicle by formulating the problem as an a-priori optimal control problem and optimizing over possible trajectories using a Genetic Algorithm. The Genetic Algorithm is found to perform well when the closed-loop dynamics of the unmanned air vehicle are approximated by second-order differential equations and the trajectory optimization is formulated as a single-shooting (as opposed to a direct collocation) problem.**

I. Summary of Findings

The goal of this project was to use a genetic algorithm to find a trajectory for an unmanned aerial vehicle (UAV) to take the UAV from the initial position $\begin{bmatrix} p_{n,i} & p_{e,i} & p_{d,i} \end{bmatrix}^T$ in the North-East-Down inertial coordinate frame (p_d corresponds to negative altitude, h) to a desired final position $\begin{bmatrix} p_{n,f} & p_{e,f} & p_{d,f} \end{bmatrix}^T$. The trajectory needed to honor the dynamic constraints of the UAV (not commanding paths that were impossible to complete) and also avoid static, spherical obstacles in the region.

I transcribed the trajectory optimization problem using the single shooting method, with a genetic algorithm population size of 1000 candidate designs. It was found that the genetic algorithm is able to overcome some fundamental weaknesses of the single shooting method if the population size is large enough.

Table I gives all major parameter values for the genetic algorithm.

Table I: Major parameter values for the genetic algorithm.

Number of time discretization points (N)	30
Population size (P)	1000
Stagnant generation limit	10
Tournament size	10
Crossover probability	80%
Mutation probability	20%
Crossover parameter (η)	0.5
Mutation parameter (β)	0.01
Total mutation generations (M)	700

To test the genetic algorithm optimizer, I ran 50 tests with randomized initial and final trajectory points, as well as randomly placed spherical obtacles with random radii. The average optimizer outputs are given in table II.

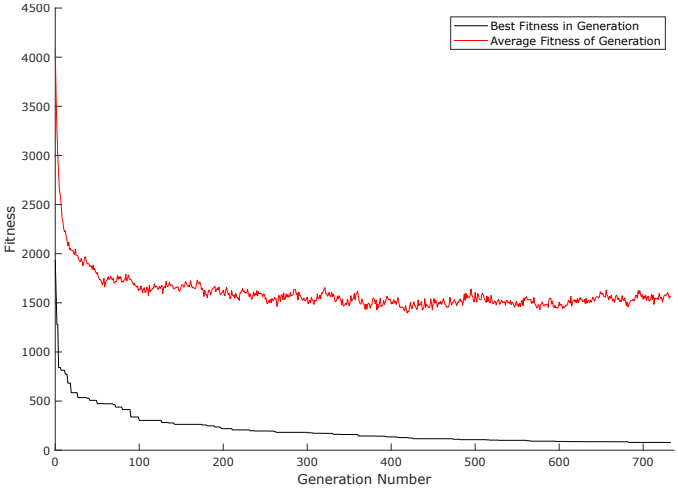

Figure 1: Fitness versus generation number for the best and average design in each generation.

Table II: Average optimizer output values over 50 randomized trials.

Generations	733
Best Fitness	79.27
Average Fitness	1562
Computation Time	2 minutes

Figure 1 gives the fitness versus generation evolution of the design, comparing the best fitness with the average fitness in the generation. Various output trajectories are shown in the Results section.

II. Procedure

To perform this optimization, I developed a simplified model of a fixed-wing unmanned air vehicle (UAV) with a feedback controller for tracking commanded course angles and altitudes, and also developed a potential field model for spherical obstacles. I did research into practical numerical methods for optimal control and experimented with them to determine the best method for turning my path planning problem into an optimization problem. Once the optimization problem was formulated, I designed a genetic algorithm with specific methods for selection, crossover, mutation, and elitism to find a locally optimal trajectory that both took the UAV to the desired location in three-dimensional space and avoided randomly-placed obstacles. The following sections describe the process in detail.

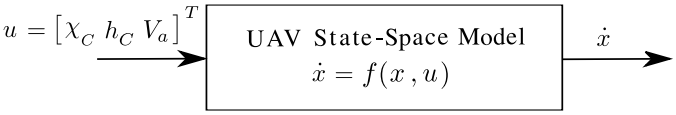

Figure 2: State-space model of UAV dynamics with input u for commanding course angle and altitude and output $\dot{x}$ for evolving the state variable values.

A. Model of UAV Dynamics and Environment

Because I wanted to formulate the path planning and obstacle avoidance problem as an optimal control problem, I needed to develop a state-space model of the dynamics of the UAV, as depicted in figure 2:

Given this model, an optimization problem could be formulated for calculating the optimal inputs to the system $u(t)$ and system states $x(t)$ to traverse from an initial state x_i to a commanded final state x_f, minimizing some cost function related to control effort and obstacle avoidance.

As is common in path planning scenarios, I decided to derive a simplified model of the closed-loop feedback system dynamics of the UAV. Since I would not assume constant altitude flight, I needed to model both the lateral and longitudinal dynamics of the aircraft. In *Small Unmanned Aircraft: Theory and Practice* by Randal Beard and Timothy McLain, the transfer function of the lateral autopilot of a UAV (which attempts to track a commanded course angle) is given in figure 3, and the transfer function of the longitudinal autopilot (which attempts to track a commanded altitude) is given in figure 4.

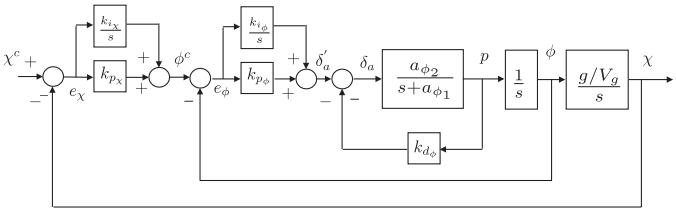

Figure 3: Transfer function of the lateral autopilot of a UAV. The input to the feedback loop is the commanded course angle χ_C, and the output is the actual course angle χ, where the course angle is defined as the counter-clockwise angle from north.

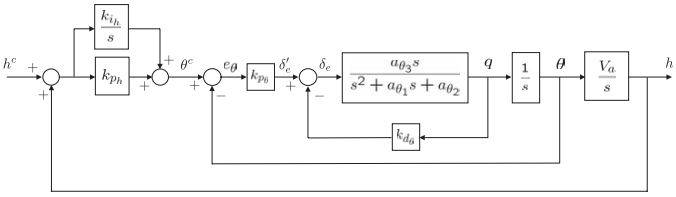

Figure 4: Transfer function of the longitudinal autopilot of a UAV. The input to the feedback loop is the commanded altitude h_C, and the output is the actual altitude h.

Implicit in this model is the assumption that lateral and longitudinal dynamics can be decoupled and linearized adequately about a nominal state. Simulating the step response of the full lateral and longitudinal autopilots using Matlab's *lsim* function, I used a least squares fitting tool to find two simplified second-order transfer functions, T_{lat} and T_{lon}, of the form:

$$T_{lat} = \frac{b_\chi}{s^2 + b_{\dot{\chi}} s + b_\chi}, \; T_{lon} = \frac{b_h}{s^2 + b_{\dot{h}} s + b_h} \tag{1}$$

The comparison between the step responses of the full feedback loop models and the simplified models are shown in figures 5 and 6.

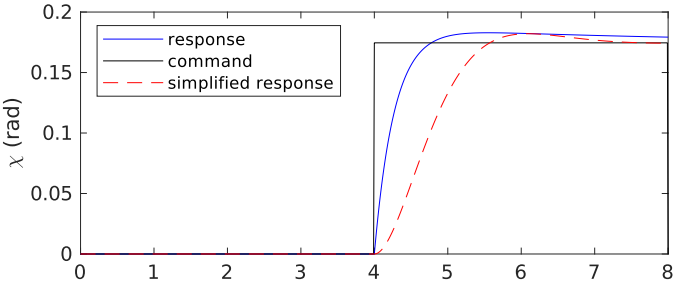

Figure 5: Comparison of the step response of the full lateral autopilot model (depicted in blue) and the simplified lateral model T_{lat} (depicted in red).

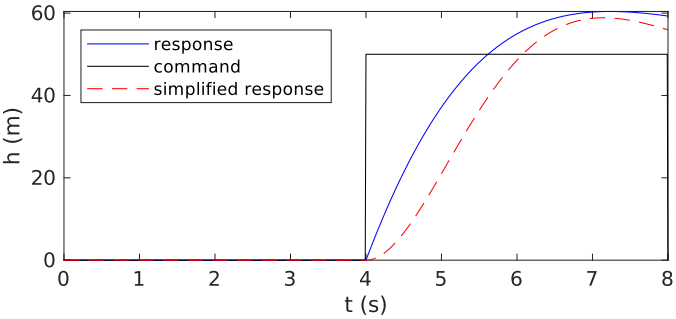

Figure 6: Comparison of the step response of the full longitudinal autopilot model (depicted in blue) and the simplified longitudinal model T_{lon} (depicted in red).

Using the derived models for T_{lat} and T_{lon}, and constructing a system state vector consisting of UAV north position p_n, east position p_e, course angle χ, time derivative of course angle $\dot{\chi}$, altitude h (where $h = -p_d$), and time derivative of altitude $\dot{h}$, I developed the following state-space model for the dynamics of the UAV:

$$\dot{x} = \begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{\chi} \\ \ddot{\chi} \\ \dot{h} \\ \ddot{h} \end{bmatrix} = \begin{bmatrix} V_a \cos\chi \\ V_a \sin\chi \\ \dot{\chi} \\ -b_{\dot{\chi}}\dot{\chi} + b_\chi(\chi_C - \chi) \\ \dot{h} \\ -b_{\dot{h}}\dot{h} + b_h(h_C - h) \end{bmatrix} = f(x, u) \tag{2}$$

where u is the input vector to the system, consisting of a commanded course angle χ_C, a commanded altitude h_C, and a commanded airspeed V_a.

I also required a mathematical model for spherical obstacles so that proximity of the UAV to obstacles could be quantified

(and penalized) by an objective function. I chose to model spherical obstacles at position $(x_{obs}, y_{obs}, z_{obs})$ with radius r_{obs} using a gaussian droplet function in three variables (x, y, z):

$$\Omega(x,y,z) = A\exp(-\frac{B}{r_{obs}^2}((x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2)) \tag{3}$$

where A and B are scaling factors. Given equation 3, an obstacle potential field can be calculated from the gradient:

$$\nabla\Omega(x,y,z) = -2B\Omega(x,y,z)\begin{bmatrix} x-x_{obs} \\ y-y_{obs} \\ z-z_{obs}\end{bmatrix} \tag{4}$$

Using equation 2, the design variables, or the inputs to the UAV system from figure 2, can be transformed into a UAV trajectory in three-dimensional space that can be penalized according to the dot product of the trajectory velocity and the obstacle field as described by equation 4.

B. Optimization Problem Formulation

1) Transcription Method: The process of "transcribing" a trajectory optimization or optimal control problem into a pure optimization problem is an area of much interest, and I had to decide between different methods for doing so. The first method I considered and tested for formulating the optimization problem is known in the literature as *direct collocation*. With direct collocation, the trajectory problem is discretized in time with $k = 1, 2, \ldots, N$, and the design variables are both the system inputs u_k *and* the system states x_k at each discretized time step k. At first, this may seem counter-intuitive, since figure 2 and equation 2 demonstrate that the evolution of x_k can be calculated deterministically with a given u_k. Indeed, the deterministic relationship between u_k and x_k must be enforced by creating equality constraints (known as "defect constraints") that set the simulated states $\hat{x}_k$ from a given set u_k equal to x_k.

While this may seem like extra work for the optimizer, direct collocation has its advantages. One of the main advantages is that optimizing over states in addition to inputs allows for the easy implementation of path constraints, such as ensuring that the final state x_N coincides with the desired final state in the trajectory, and not a random point in the configuration space. For this reason, direct collocation is popular when used with gradient-based optimizers that depend on an initial guess for the design variables. However, when I formulated the optimization problem with direct collocation, the design space proved too large for my genetic algorithm to converge in a timely manner. Discretizing my trajectory problem with $N = 10$ time steps resulted in a design variable vector with 121 elements, and the optimizer would not converge on a solution after one hour of running.

I considered an alternative to direct collocation known as *single shooting*. With single shooting, the trajectory problem is once again discretized in time, but this time the design variables are just the system inputs u_k. This method is analogous to aiming and shooting a cannon (hence the name), where u_k is chosen without any path constraints as a guide, and the system is simulated forward in time to obtain x_k. Because the design space is so large, it is clear that the single shooting method suffers without a good initial guess when using a gradient-based optimizer. However, I suspected that using a genetic algorithm optimizer might allow me to overcome this weakness due to its sampling of the entire design space from the outset. This method proves effective when paired with a genetic algorithm for trajectory optimization, as is detailed in the rest of this report.

2) Fitness/Objective Function and Constraints: Using the single shooting transcription method, I required an objective function which penalized the following characteristics of a candidate design variable vector u:

- Error between the final simulated trajectory state x_N, simulated with fourth-order Runge Kutta (RK4), and the desired final state $x_{f,desired}$.
- Large jumps between the values of u_k and u_{k+1}.
- Proximity to obstacles.

To penalize these three things, I designed the following objective function:

Algorithm 1 Compute the fitness of a candidate design.

Input Candidate design $u = \begin{bmatrix} u_1 & u_2 & \ldots & u_N \end{bmatrix}$
Output Scalar cost value; large value = poor fitness

1: **for** $k = 1, \ldots, N$ **do**
2: Derive $x_k = \begin{bmatrix} p_n & p_e & \chi & \dot{\chi} & h & \dot{h} \end{bmatrix}$ using u_k, equation 2, and RK4
3: Final state error $x_{f,e} = norm(x_N - x_{f,desired})$
4: Calculate obstacle field gradient $\begin{bmatrix} f_x & f_y & f_z \end{bmatrix} = \nabla\Omega(x_k)$
5: Directional derivative of trajectory in obstacle field $s = 1/(1+\dot{h}^2)\begin{bmatrix}\cos\chi & \sin\chi & 1\end{bmatrix}\begin{bmatrix} f_x & f_y & f_z \end{bmatrix}^T$
6: cost $+= w_1 * s^2 + w_2$*diff$(u)^2 + w_3 * x_{f,e}$
7: check for violated constraints and add segregation term to cost if violated
8: **end for**

To clarify the segregation term statement in the algorithm, the algorithm checks to see if one of the constraints from equation 6 is violated, then adds a large term (I used 200) to the fitness to reflect the undesirability of constraint violation.

3) Design Variables and Constraints: With single shooting, I packed the commanded course angle χ_C (in radians) at each time step k, commanded altitude h_C at each time step (in meters), and the commanded airspeed V_a (in meters/second) into a single design input vector u with $2N + 1$ elements:

$$\begin{aligned} u &= \begin{bmatrix} u_1 & u_2 & \ldots & u_N \end{bmatrix} \\ &= \begin{bmatrix} \chi_{C,1} & \cdots & \chi_{C,N} & h_{C,1} & \ldots & h_{C,N} & V_a \end{bmatrix} \end{aligned} \tag{5}$$

As suggested by equation 5, a single commanded airspeed value is chosen over the entire trajectory. The $\chi_{C,k}$ values determine the UAV lateral trajectory, the $h_{C,k}$ values determine the UAV longitudinal trajectory, and the V_a value determines the length of the overall trajectory over the specified time

interval. To test the functionality of the genetic algorithm, I determined that $h_{C,k}$ and V_a must take on integer values.

Additionally, I placed constraints on the values of u to avoid commanding erratic flight paths:

$$\begin{bmatrix} -3\pi/2 \\ \min(h_i, h_f) - 20 \\ 15 \end{bmatrix} \leq \begin{bmatrix} \chi_{C,k} \\ h_{C,k} \\ V_a \end{bmatrix} \leq \begin{bmatrix} 3\pi/2 \\ \max(h_i, h_f) + 20 \\ 25 \end{bmatrix} \tag{6}$$

C. Genetic Algorithm Design

Equation 5 defines the design representation for the genetic algorithm, with the exception that the fitness value is also appended to u within the algorithm. At the start of the optimization, an initial population is created by evenly distributing the population across the feasible space defined by equation 6. To create an effective genetic algorithm, I implemented the following methods for selection, crossover, mutation, and elitism. These methods encourage fitness pressure, inheritance, and diversity, which make genetic algorithms far more effective than ad hoc sampling algorithms.

- **Selection:** I chose the tournament method to carry out selection of parents, with a tournament size of 10. The order of the population is shuffled randomly, and random indices are then used to pair two members of the population together, selecting the one with the better fitness. With a tournament size of 10, the fitness pressure is relatively small relative to the large size of the population (10/1000). I found this to not pose an issue, however, as the results demonstrate that the overall fitness of the population improves quite rapidly over the course of 30 generations or so.
- **Crossover:** Once two parents are selected to create children, my algorithm uses a semi-blended crossover technique, which comes from setting the crossover η parameter value to 0.5, as explained in the course notes. I did this to allow for more variation in the inherited values given from the parents to the children, while still anchoring the values somewhat to the values of the parents. My thought was that partially anchoring the values of the children to the values of the parents would help prevent infeasible trajectories from being generated from feasible trajectories.
- **Mutation:** I valued the ability of mutation to help a child design improve over either of its parent designs and introduce diversity in the population. However, I was also aware that a high mutation probability could negate the effects of crossover. I settled on the highest mutation probability I felt I could get away with, which was 20%. I elected to implement dynamic mutation, which begins with uniform mutation, then favors mutation values near the current gene as the number of generations increases. Running the optimization many times and seeing an average generation number of around 700, I set M, the total generation number for the mutation calculation, to 700, and the mutation parameter β to 0.01 to make the dynamic mutation effect small.
- **Elitism:** After all children for the next generation have been created and mutated, the children are combined with all parents from the previous generation, and only the top half of the population (as sorted by fitness value) survives for the next generation.

One key design decision was that instead of specifying a set number of generations for which to carry out the simulation, I found that I achieved the best overall output trajectories in the timeliest manner possible if I instead made the algorithm stop after a certain number of *stagnant generations*, or generations for which the best fitness did not improve. I settled on a stagnant generation limit of 10, which seemed to give a nice tradeoff between output desirability and computation time.

D. Empirical Testing for Determining Optimal Parameters

It would not be inaccurate to state that at least half of my development time was spent attempting to find the best parameters for transcription, the cost function, and the genetic algorithm to obtain a desirable trafeoff between trajectory accuracy, feasibility, and computation time. The majority of my parameter determination efforts went to the following parameters:

- **Number of trajectory time discretization points (N):** The solve time for the genetic algorithm increases exponentially as a function of this parameter. The time range for the simulation is $0 \leq t \leq 10$s, so I initially chose $N = 10$, corresponding to a time step value of one second. This resulted in a decision vector size of 21 and a solve time of ≈ 10 seconds. However, the resulting trajectory path often included jagged edges, which indicates that the time discretization was not fine enough to adequately incorporate the dynamic constraints of the UAV. I kept increasing N until the output trajectories were always smooth for randomized trials. I eventually settled on $N = 30$, which resulted in a solve time of ≈ 2 minutes.
- **Genetic algorithm population size (P):** This was the second most important parameter for solve time after N. That being said, solve time seemed to increase more closely to a linear rate with respect to P, and increasing this parameter tended to allow shorter overall paths to emerge as a locally optimum solution. I settled on $P = 1000$.
- **Cost function weights (w_1, w_2, w_3):** The relative values of these weights determine how the optimizer prioritizes avoiding obtacles versus minimizing input variation versus minimizing the error between the final point in the trajectory and the desired final position, $x_{f,e}$. Through experimentation, I found that the obstacle avoidance weight w_1 needed to be much greater than the other two weights, or the resulting trajectories would liberally run right through obstacles. I found that the input variation weight w_2 could be composed into two separate weights for commanded course angle and commanded altitude, with the commanded course angle weight being greater than the commanded altitude weight. Finally, the final position error weight w_3 needed to be greater than the largest w_2

weight. I arrived at $w_1 = 200, w_2 = \begin{bmatrix}1.0 & 0.5\end{bmatrix}, w_3 = 2.0$, which gave desirable trajectory outputs.

III. Results

To test the genetic algorithm optimizer, I ran 50 tests with randomized initial and final trajectory points, as well as randomly placed spherical obstacles with random radii. The average optimizer outputs are given in table III.

Table III: Average optimizer output values over 50 randomized trials.

Generations	733
Best Fitness	79.27
Average Fitness	1562
Computation Time	2 minutes

Figure 7 gives the fitness versus generation evolution of the design, comparing the best fitness with the average fitness in the generation. The overall fitness of the population improves drastically within the first 30 generations or so, then the average fitness levels off as the best fitness slowly improves over the next hundreds of generations. Another interesting fact to note from the figure is that there is an immense disparity between the best and average fitness in each generation.

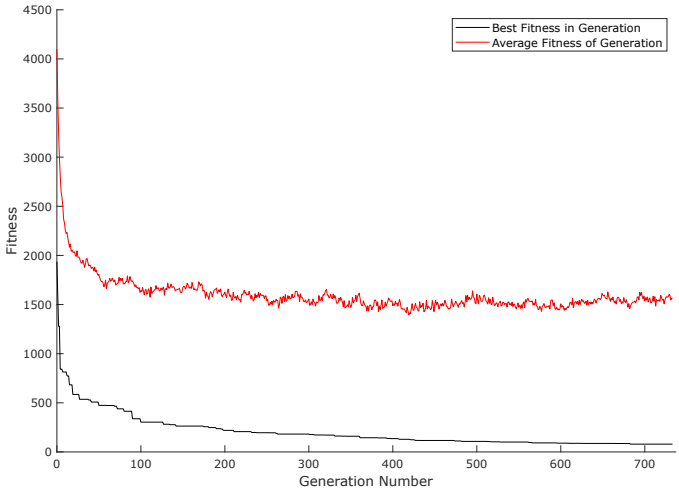

Figure 7: Fitness versus generation number for the best and average design in each generation.

With the proper parameters discussed in the previous section, the genetic algorithm demonstrates the ability to generate UAV trajectories that arrive at the desired final position, avoid obstacles, and conform to the dynamic constraints of the aircraft. Figures 8-10 show some sample output trajectories over the 50 randomized trials.

Of 50 randomized trials, the generated trajectory ended within one meter of the final desired position all 50 times. This is remarkable, considering the fact that single shooting is known to have trouble generating trajectories that conform to path constraints without a well-chosen initial guess. Of all trials, the generated trajectory crossed through a random obstacle 4 times, giving an obstacle avoidance success rate of 92%. Finally, of all trials, the generated trajectory conformed

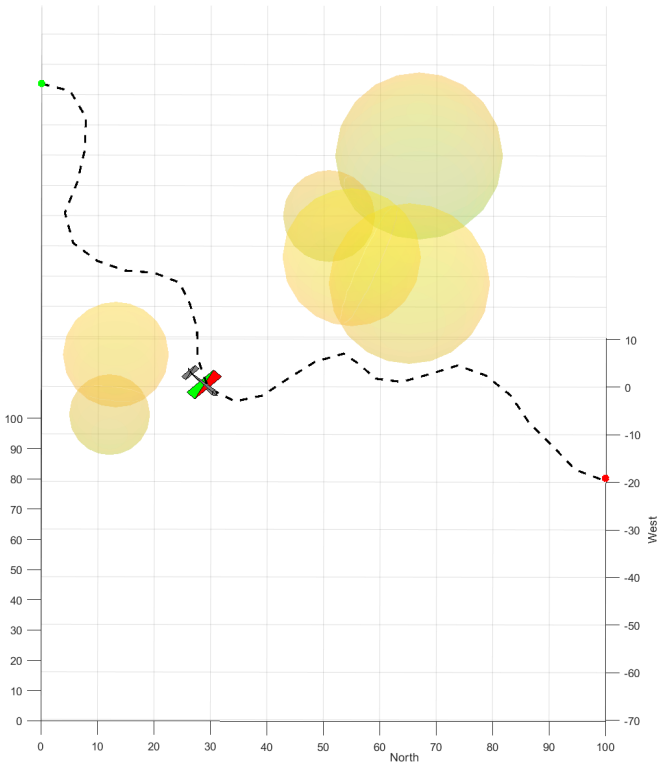

Figure 8: Genetic Algorithm output UAV trajectory for randomized trial #5.

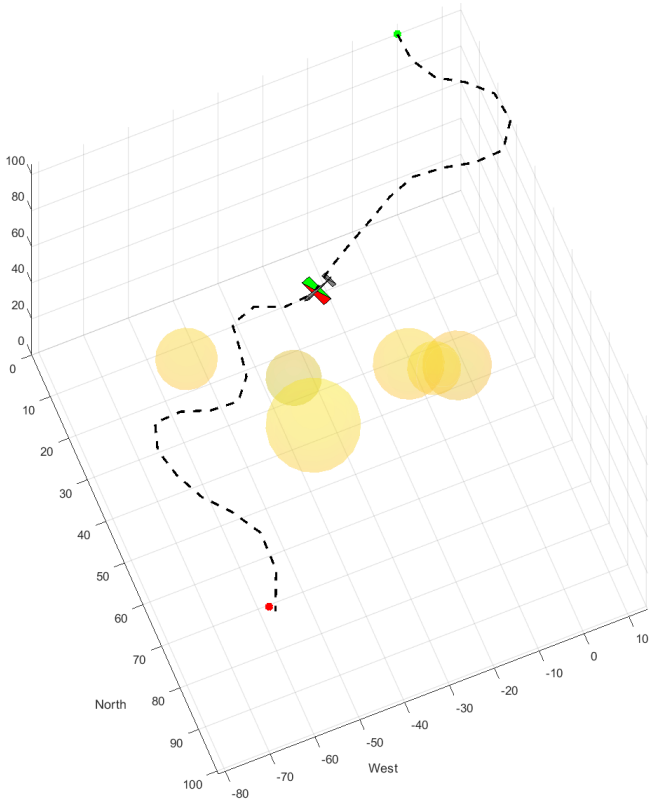

Figure 9: Genetic algorithm output UAV trajectory for randomized trial #20.

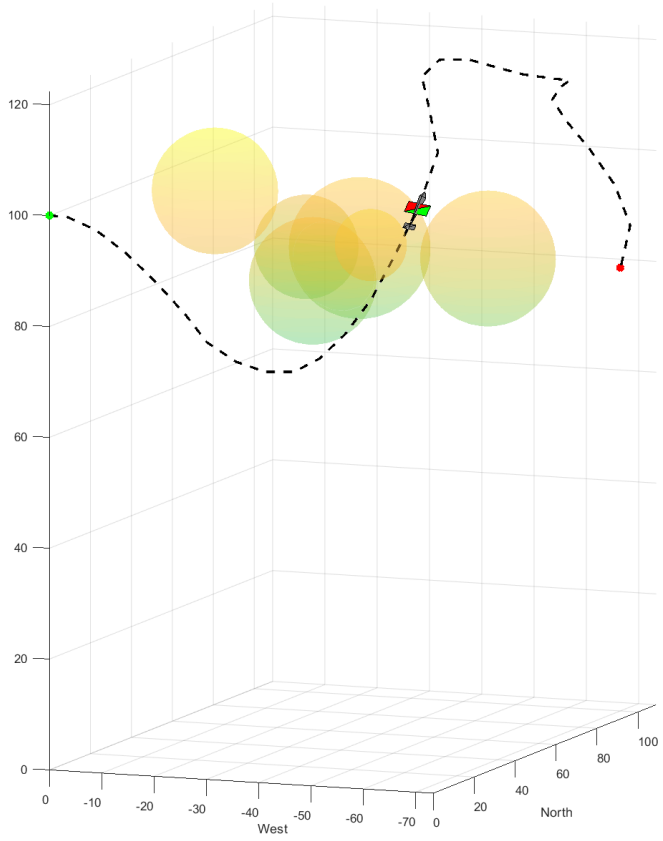

Figure 10: Genetic algorithm output UAV trajectory for randomized trial #41.

to the dynamic constraints of the UAV all 50 times. It should also be noted that the constraints on the design variables were at times binding, but never violated in the final designs.

IV. Discussion

Figure 7 demonstrates that after a relatively small number of generations, the disparity between the fittest design and the population at large becomes huge, and the average fitness of the population grows stagnant as the fitness of the best design steadily improves over many future generations. This pattern seems to suggest that the algorithm would benefit greatly from resampling, in which only a top percentage of the population is allowed to continue on and "reproduce." Other algorithms, such as the particle filter, utilize some form of resampling, and it often leads to much faster computation times. Thus, resampling may allow for still greater initial population sizes, which would lead to even better-conditioned final trajectories without the associated huge computational costs. The steady decrease in the objective value across generations also suggests that the chosen genetic algorithm parameters are adequate.

The sample output trajectories demonstrate a few heartening characteristics of the optimizer. Perhaps the most striking observation is that the genetic algorithm appears to have no problem converging to a trajectory that arrives at the final desired position x_f, despite the fact that single shooting transcriptions for optimization problems tend to struggle to accomplish this. This counterintuitive result is probably a result of how the genetic algorithm initializes candidate designs by uniformly distributing the initial population across the design space. Thus, the genetic algorithm doesn't fall victim to the same dependence on an initial design guess that gradient-based methods tend to share.

Figure 10 clearly demonstrates the ability of the optimizer to explore all of the dimensions of the configuration space, and the jump from 2D to 3D path planning under this scheme is trivial, unlike with other path planning schemes such as the Rapidly Exploring Random Tree method. Moreover, the ability of the optimizer to repeatedly simulate the UAV system using RK4 integration ensures that the resulting trajectory is feasible for a UAV to fly, as long as time is discretized in a sufficiently fine manner. It should be noted, however, that the output trajectories tend to have many turns, resulting in relatively convoluted paths. There are a couple of possible remedies for this. Perhaps the optimization space could be limited to regions where clusters of obstacles are present, favoring straight-line paths outside of these regions. Additionally, the cost function could penalize the length of the output trajectory, possibly by penalizing the commanded airspeed value.

One final observation gained from this project is that the successful execution of the optimizer requires the tuning of many different parameters having to do with the objective function, the transcription, and the genetic algorithm itself. It can be easy to get lost in all of the parameters, spending a lot of time trying to obtain the ideal tradeoff between competing algorithm requirements and objectives. A possible help on this front may come from formulating the trajectory optimization as a two-objective problem, splitting obstacle avoidance and input smoothness/x_f error into two different objectives. This would entail the use of a Pareto front, and would allow for more explicitly playing with the tradeoffs between a desirable trajectory shape and the distance from obstacles.

Genetic algorithms are known to be extremely versatile in their ability to tackle a wide array of complex optimization problems. This is further confirmed by the success of the genetic algorithm in solving the trajectory optimization problem with such a high degree of accuracy. The main drawback of these algorithms observed in this project is the steep tradeoff between performance and computation time. An average of two minutes for computing a trajectory flown in ten seconds makes genetic algorithms considerably slower than other trajectory optimization methods. However, as discussed above, there are possible remedies to this long computation time, and it is likely that genetic algorithms will continue to be widely used for similar applications, especially as processors improve and algorithmic innovation persists.