Using Genetic Algorithm to Modify Scheduled Flights to Track Regular Traffic

Leonardo Piñero-Pérez[*]
Bell Flight, Arlington, TX, 76019

A methodology for tracking the traffic of moving objects is developed by using existing fleet operations of air vehicles, particularly unmanned aerial vehicles. This work proposes a solution to the search-and-track problem in the case of unknown, regularly-occurring traffic while ensuring minimal disturbance to existing fleet operations by means of a dynamically-adjusted genetic algorithm. The performance metric used is the percentage of the flight time for which the target object is within the line of sight of the vehicle. In the best case, where target travel is nearly parallel to the flight path, experimental results yield a 13 percent improvement in this performance metric after 3 generations (24 flights) and a 58 percent improvement after 80 generations (640 flights).

I. Nomenclature

α	=	hyperparameter to determine tuning update
β	=	tuned scaling parameter
δ	=	scaling vector for alteration vector
η	=	mutation factor
λ	=	hyperparameter for determining total cost
μ	=	tuned value of symmetry for shaping
θ	=	angle between distance vector and simulated object path
ρ	=	tuned power-scaling bias
σ	=	tuned standard deviation parameter of activation function
b	=	index of child set of parameters
B	=	batch size
C	=	cost metric for a given flight path
$\vec{d}$	=	distance vector from node to anchor point
e	=	denotes last “known” point location (last anchor point) for simulated object point
i	=	iteration of an initial or altered path
I	=	total number of main loop iterations
j	=	index of a node for a given path leg
k	=	index of a contacted object point
K	=	set of k contacted object points
$\vec{n}$	=	node location
$\vec{o}$	=	target object location
P	=	tracking performance for a given flight path
$\text{randn}(B)$	=	outputs B normally distributed random numbers with a mean of 0 and standard deviation of 1
s	=	denotes first “known” point location (first anchor point) for simulated object path
t	=	location along time axis
T	=	discretized locations along time axis representing a section of the path’s flight time
$\vec{u}$	=	orthonormal basis vector for the planes of alteration
x	=	location along East/West axis
X	=	discretized locations along East/West axis representing a path section
y	=	location along North/South axis
Y	=	discretized locations along North/South axis representing a path section

[*]Engineer I, Flight Control Laws, leonardo.pinero42@gmail.com, AIAA Member.

II. Introduction

A methodology for tracking moving objects with an existing fleet of aircraft is developed. An "object" is defined as any detectable information moving spatially as a *cyclical* function of time, such as automobile traffic for a given weekday, other aircraft on a given schedule, or air pollution for a given a week. In real-world applications, a fleet of unmanned aerial vehicles is the intended platform for obtaining measurements. The approach provides:

1) Operational flexibility
2) Ability to operate in dense urban environments
3) Ability to achieve low and slow cruise conditions
4) Potential to achieve high density measurements per square-mile

The resulting algorithm is agnostic to the type of measuring vehicle as long as its spatial path or cruising velocity can be freely altered within a given set of constraints. To meet this criterion, an aerial platform (drone) will be considered, though special use cases could also include submersibles or maneuverable spacecraft.

This methodology would likely be inadequate to implement on ground fleets which have to cope with overly-constrained guidance variables related to navigating through non-flat terrain. It can be seen that vehicles operating at radar altitudes below 100 meters will have a short LOS (Line of Sight) due to ground obstructions [1].

A drone's minimum and maximum velocity defines limits on how its path can be altered as a function of time. To establish a given drone's field of view, a "sight parameter" is used to define a "sight" distance that depends on the drone's altitude and features of the terrain (including any tall buildings) that obstruct the drone's full view. In general, the drone's cone of sight will cover more terrain as altitude increases. [2]

The LOS or altitude have a trigonometric relationship to the sight parameter by the visual cone angle provided by the on-board instrumentation. For the purposes of this methodology, these parameters are constant, so we are assuming a constant altitude and constant LOS.

III. Applications

As discussed in the introduction, a fleet of unmanned aerial vehicles provides the best platform in real-world applications of this methodology. This fleet will have consistent locations where the aerial vehicles will be scheduled to arrive at and depart from at specified times. Given this schedule, the goal is to not disturb the existing paths such that the side benefit of using the fleet for survey operations does not interfere with the main mission of this fleet, whatever it may be. This section will propose some potential side-benefits and applications of the proposed methodology.

A. Google Maps Validation

The Google Maps application has a solution to the problem of tracking automobile traffic. The application aggregates the real-time location data generated by drivers using Google Maps or Android phones while having location services enabled [3]. The methodology proposed in this paper can be used to assess and validate the accuracy of this existing system. The validated data could also be used to determine the number of users actually broadcasting their anonymous location data via Google Maps or Android.

B. Ecological Observation

Another potential use case for this system is to track the regular movements of animal species. Using unmanned aerial vehicles has been shown to be a feasible non-intrusive way to survey smaller animals for various platforms [4], as satellite surveying has resolution limitations. For ecological surveillance, satellites have both a low spatial and temporal resolution: Landsat 7 requires a wait time of 16 days to revisit the same location, whose spatial resolution is such that each pixel represents 15 meters in length (a relatively high resolution) [5].

C. Atmospheric Survey

While the previous use case is used to validate existing data, the platform may be used to generate novel data for non-visual objects, such as air pollution. Fleet vehicles may be equipped with instrumentation designed to detect the presence of particulates. The sight parameter could be the expected size of a particulate cloud or the region of airspace for which detection will represent a cloud's presence. For this use case, the parameter has a slight change in definition from that stated in the introduction. This is due to the fact that particulate detection would occur at the exact location of the aerial vehicle, rather than at a distance via ranged vision.

This can actually also be applied to ranged vision. For example, if the ranged vision of the platform is small, but is targeting something such as algal blooms, coming into contact with the edge of a bloom could represent the centroid of it, which would be outside of the range of vision of the platform. At this scale, however, this only garners an advantage with temporal resolution. Because of this, satellite imagery appears to be a more appropriate platform for these sorts of surveys due a low minimum resolution required [6].

IV. Background

Topics associated with using a UAV platform to survey a geographical region can include [7]:

1) Full-view coverage
2) Camera coverage
3) Optimal camera location/orientation
4) Search and tracking (SAT)
5) Testbed implementations

This paper wishes to address the topic of SAT. One kind of solution involves probabilistic modelling. With this methodology, the possible locations of an object are determined from its expected velocity and last known location. These possible locations are represented as a probability distribution [8]. This is useful because the target cannot realistically be known ahead of time.

The problem of tracking multiple unknown objects is addressed by this work. While the tracked objects can be modelled with probabilistic motion, this methodology relies on tracking the assumed regular behaviors of the target by iterating the platform's paths for each visit. This is useful for the applications mentioned in the previous section, but would be unhelpful for searching for and tracking randomly-moving objects, such as those within a battlefield.

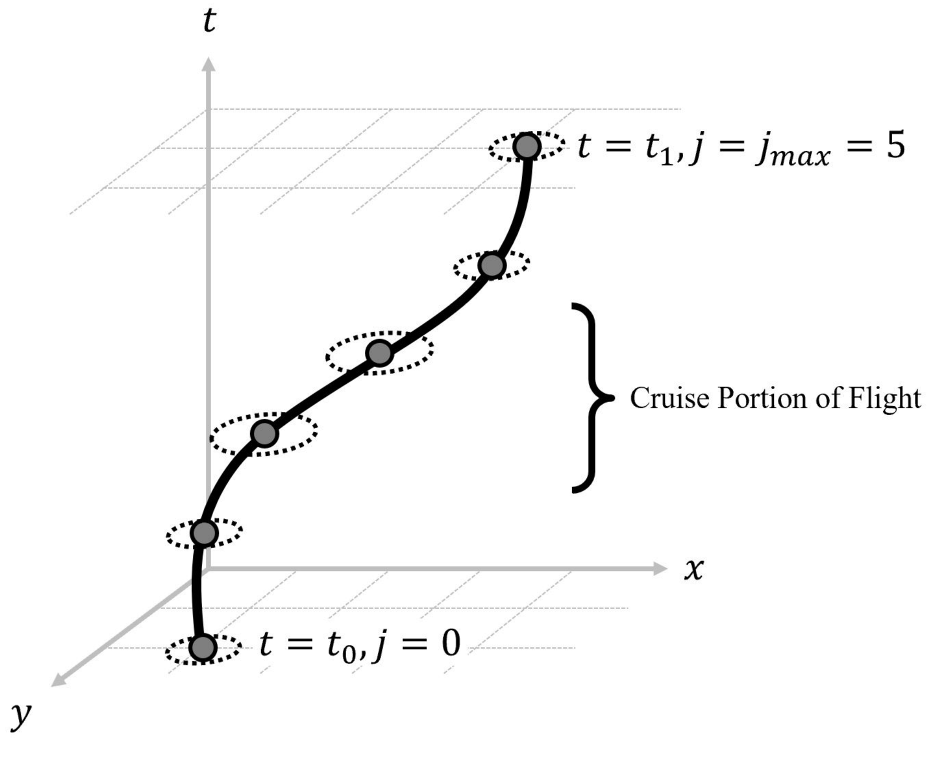

Fig. 1 Methodology's representation of a drone flying through space and time.

The methodology approaches the problem of object tracking by visualizing the entire system as a 3-dimensional volume shown in Fig. 1, where the xy-plane represents a region over which the drone fleet is to operate. In Fig. 1, the z-axis is the time-domain. As a result, drone flight paths and times of contact with the target object are visualized.

With this 3-dimensional definition of drone and object, location vectors will be defined to represent the spatial

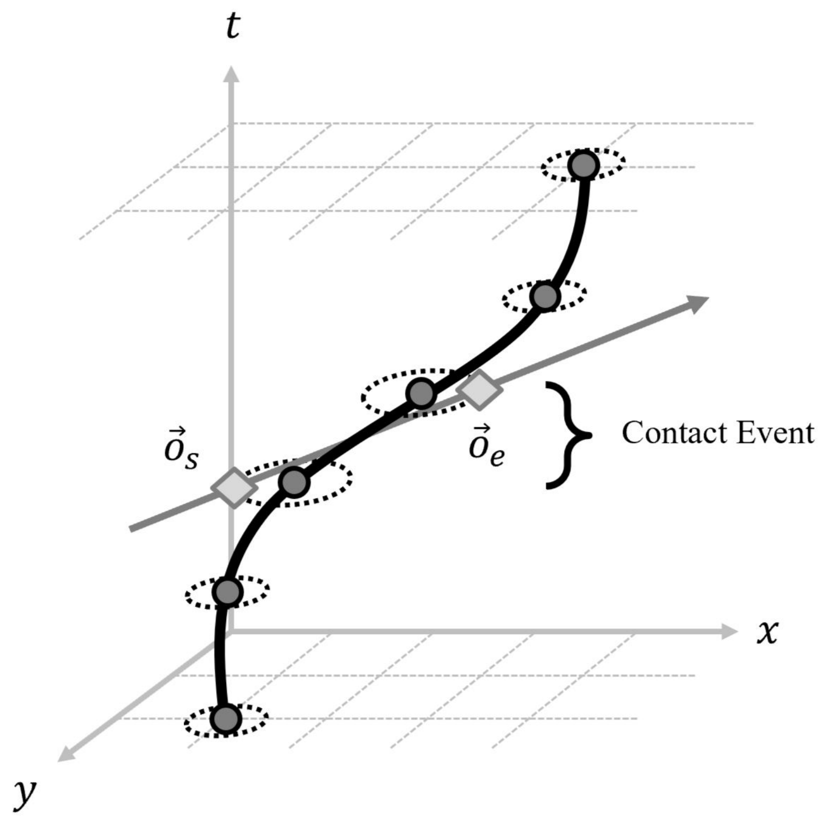

Fig. 2 Representation of contact event.

location of the drone in the time-domain, in the form of $\vec{n} = [x, y, t]$, where $(x, y, t) \in (X, Y, T)$.

The sight parameter manifests itself as the radius of a thin circular slice parallel to the xy-plane. The small height (along the time axis) of this slice is equal to the time-step between each evaluated path node. Any simulated target object within this cylindrical volume is considered to have made "contact" with the manipulatable path node. This contact is referred to as a "contact event", being defined by the location/time of first contact and the last known location/time, as shown by Fig. (2).

V. Methodology

The following section outlines the methodology used to determine the optimal path alterations by means of a genetic algorithm. The algorithm and its user interface is run on a Flask server, which extracts and stores simulation data via connection to a MySQL database (structured query language database) hosted on the local machine. The source code can be found at `https://github.com/Leonardo767/aiaa2019-publication`, where everything except the static cascading styling sheets (for aesthetics) is original work of the author. Inside the source code sample folder, there are two main sections: "lib" and "src", which hold the code for application utilities and the genetic algorithm, respectively. For the script handling the application's data transfer and database connection, refer to "routes.py".

A. Application Architecture

The application is run on a Flask server on the local machine. This was built to accelerate the development process by providing a means for the author to assess the incremental results generated by the optimization algorithm. The application was developed in the order of the workflow shown in Fig. 3, beginning with the geographical data. This data contains information about the geography to be surveyed, such as the boundary dimensions, references to airports contained, and the flight manifests referenced by said referenced airports. From here, flight paths can be constructed by extracting and manipulating the stored geographical information.

The next step is to input simulated objects. At the top level, the user can select simulation suites which represent the total movement of all objects for a given geographical region. It is important to note that this data is unknown to the fleet operator, and is only implemented here for the sole purpose of this *simulated* data to act as an evaluation of the algorithm from an omniscient perspective. The individual objects themselves and their respective path points as a function of time can be edited within the MySQL database.

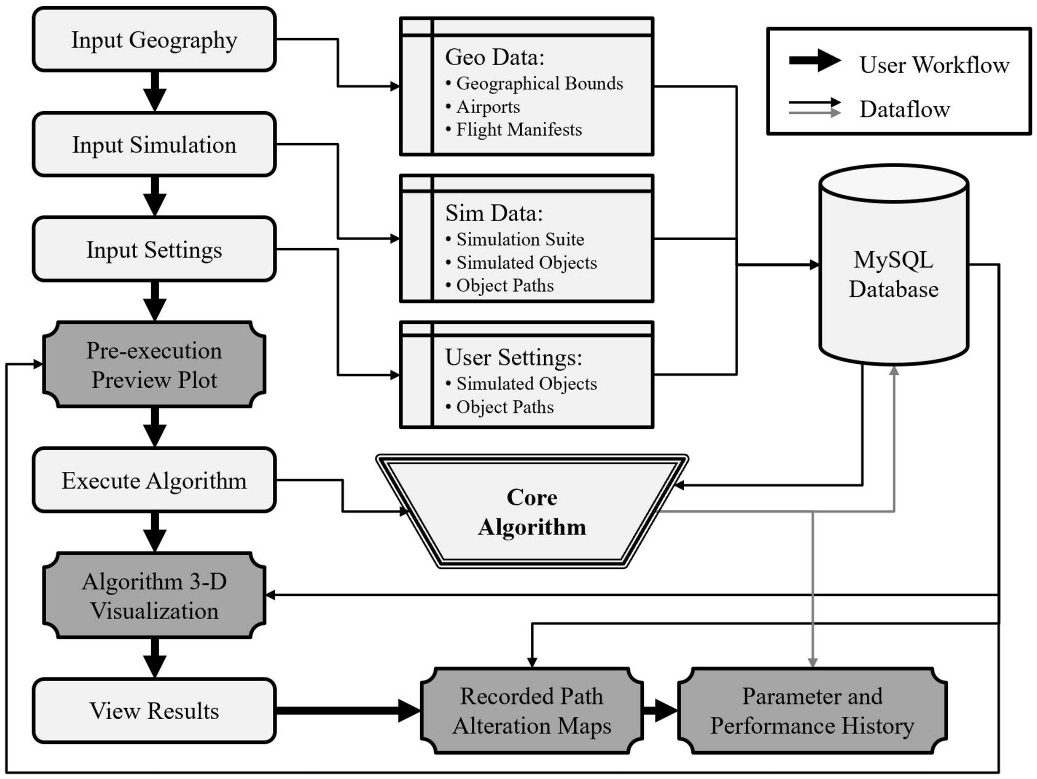

Fig. 3 Application architecture built to interface with core algorithm.

The settings input is used to add any additional simulation settings such as the load card used or fleet vehicle properties such as allowable velocity and range of sight. Within this page, an overall preview map is available of the region.

These data are fed into a MySQL database that the Flask app can interact with. It is from this database that the algorithm pulls information and packages its results per iteration. From these results, the plots of the altered paths are generated. Each database object's table holds references to other tables according to the structure shown in Fig. (4). The light-colored portions in italic font are associated with the simulation and are only used to compute the contact points. For simplicity and isolation of variables, the sight schedule is held constant in this implementation. Again, the genetic algorithm is unaware of simulation data outside of what the contact event is composed of.

The application then prompts the user to an execution page, where the number of iterations and type of execution is defined by one of four selected run-modes:

1) *Reset*: resets iteration value and does one full iteration
2) *Iteration++*: runs the next iteration
3) *Execute Completely*: runs for iterations defined in execution settings
4) *Debug (no plot)*: outputs via machine terminal, used for development

Upon termination, the resulting altered fleet paths can be viewed in the 3-dimensional environment alongside the simulated paths. Additionally, historical plots are generated for the path alteration and parameter updates per iteration.

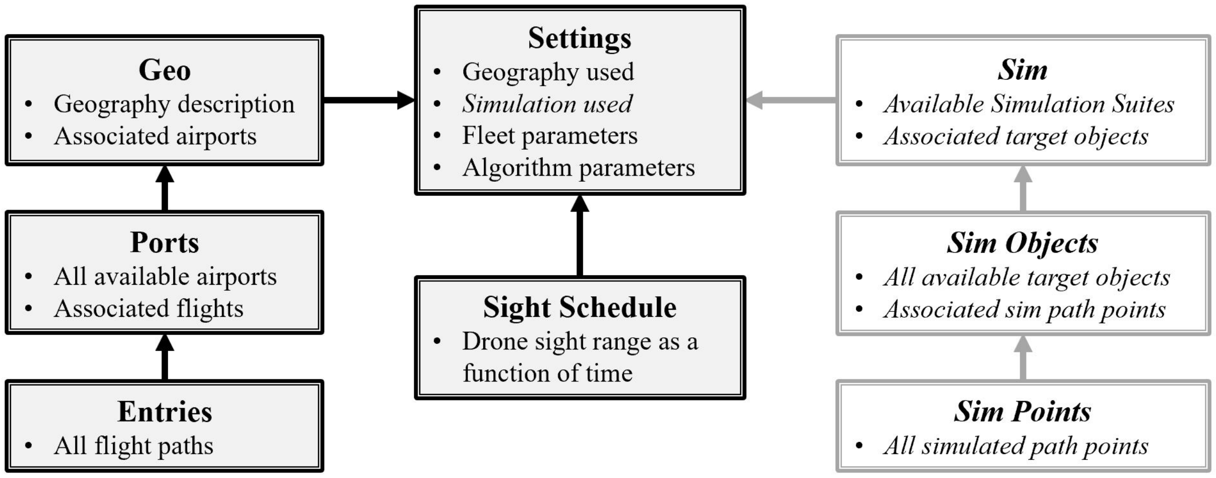

Fig. 4 MySQL Database Structure for Application.

B. Algorithm Architecture

The algorithm is situated in the development application to mimic how it would operate in reality: the only information it interacts with is the known operational flight path and the simulated contact with the target objects. The fleet will fly its initial flight paths, and the coincidental contact made with the target objects (based on the sight parameter) provide the first iteration's input. The following sub-sections will further elaborate each labeled component in Fig. 5. For reproducibility, note that the PyTorch seed value for the random mutations is set to zero at the start of the optimizer's main function.

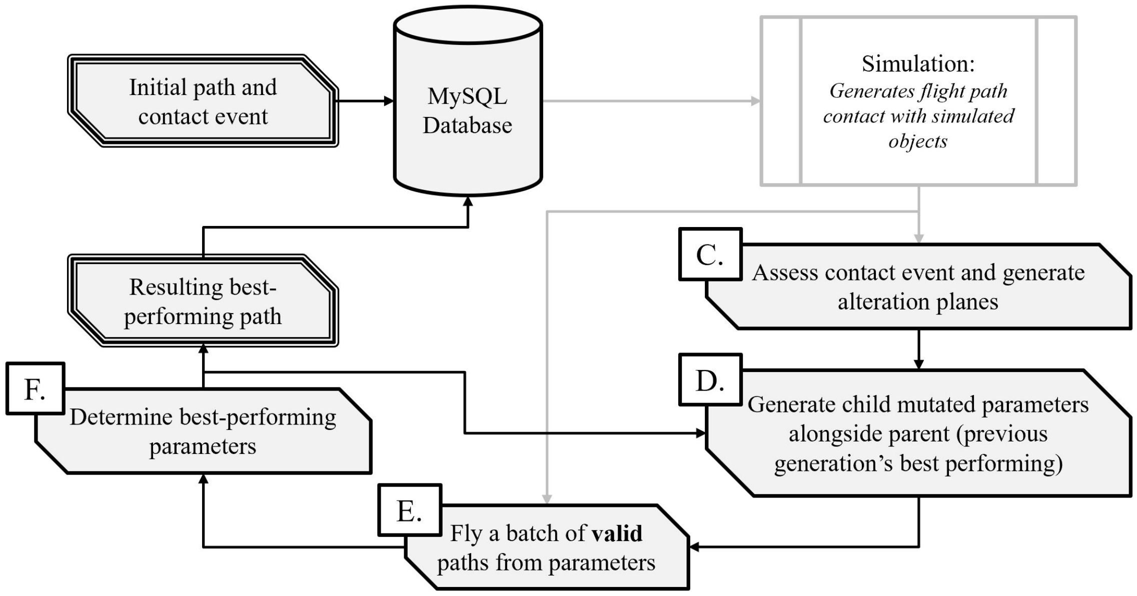
Initial path and contact event
MySQL Database
Simulation: *Generates flight path contact with simulated objects*
Resulting best-performing path
C. Assess contact event and generate alteration planes
D. Generate child mutated parameters alongside parent (previous generation's best performing)
F. Determine best-performing parameters
E. Fly a batch of **valid** paths from parameters

Fig. 5 Top-level architecture of core algorithm.

C. Assess Contact Event

Upon the completion of a contact event, the situation is first described as two data objects in $\mathbf{R}^3$ space: the path of the simulated object and the path of the flight leg. Both are a collection of vectors describing the location of each path point.

For the generation of alteration vectors, only the points representing first and last contact are used, as these define the boundaries of the drone's observation (these points will be referred to as "anchor points"). A collection of distance vectors is generated as the differences between the flight path vectors and anchor points using Eq. (1). Figure 6 shows this relationship between the flight path and tracked object.

$$\vec{d}_{j_{\{s,e\}}} = \vec{o}_{\{s,e\}} - \vec{n}_j \tag{1}$$

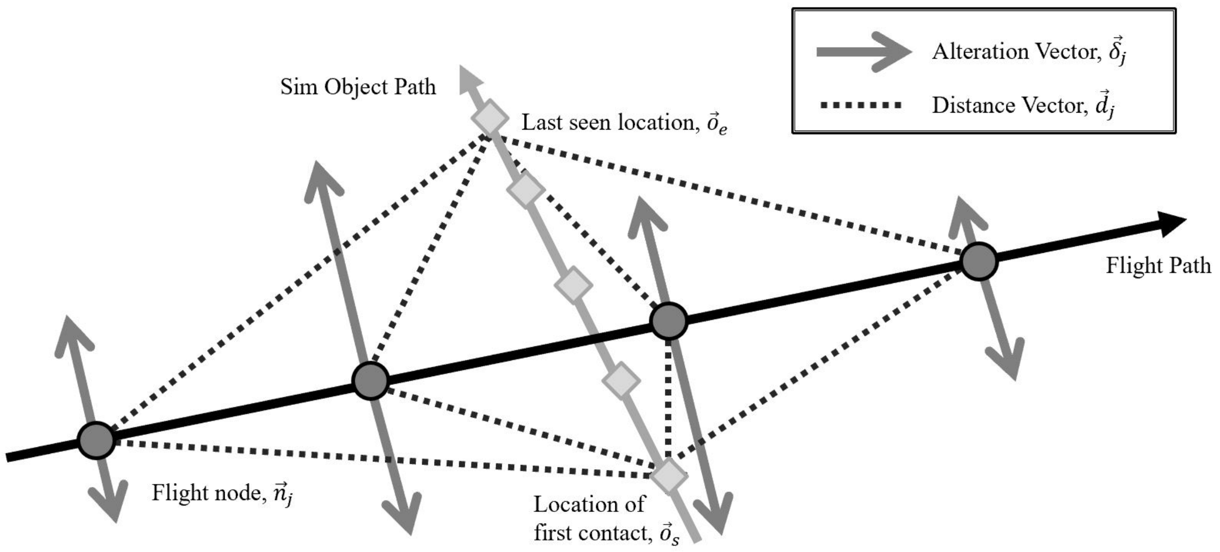
Alteration Vector, $\vec{\delta}_j$
Distance Vector, $\vec{d}_j$
Sim Object Path
Last seen location, $\vec{o}_e$
Flight Path
Flight node, $\vec{n}_j$
Location of first contact, $\vec{o}_s$

Fig. 6 Relationship between flight and object path.

It can be seen that $\vec{d}_j$ is divided by the set $\{s, e\}$. For each object point $\vec{o}_s$ and $\vec{o}_e$, a planar subspace is generated with basis vectors $\vec{d}_j$ and $(\vec{n}_{j+1} - \vec{n}_j)$. From here, the Gram-Schmidt process is employed to generate an orthonormal basis using $(\vec{n}_{j+1} - \vec{n}_j)$ as one of the basis vector directions. The basis vector normal to the plane is used as the direction for that node's alteration vector, keeping in mind that this is with respect to the object point $\vec{o}_{\{s,e\}}$. These basis vectors are computed by Eqs. (2) and (3), where the latter computes the direction of the alteration vector.

$$\vec{u}_1 = \frac{\vec{n}_{j+1} - \vec{n}_j}{\|\vec{n}_{j+1} - \vec{n}_j\|} \tag{2}$$

$$\vec{u}_{2_{\{s,e\}}} = \frac{\vec{d}_{j_{\{s,e\}}} - \left(\vec{d}_{j_{\{s,e\}}} \cdot \vec{u}_1\right) \circ \vec{u}_1}{\left\|\vec{d}_{j_{\{s,e\}}} - \left(\vec{d}_{j_{\{s,e\}}} \cdot \vec{u}_1\right) \circ \vec{u}_1\right\|} \tag{3}$$

Note that $\vec{u}_1$ is common to both $\{s, e\}$ planes. They only differ by the direction of the orthogonal vector $\vec{u}_2$, such that the common line of intersection between these two planes is the flight path itself.

D. Generating Child Paths

For both planes, a set of alteration vectors are generated along $\vec{u}_2$ for each flight path node. The nodes are altered along each plane's second orthonormal basis using Eq. (4).

$$\vec{n}'_{j_{\{s,e\}}} = \vec{n}_{j_{\{s,e\}}} + \delta_{j_{\{s,e\}}} \vec{u}_{2_{\{s,e\}}} \tag{4}$$

The basis $\vec{u}_2$ (normal to the flight path) is scaled by δ_j, a scalar value specific to each $\vec{n}_j$ per node index. Each δ_j is determined as a function of j, the node's index on the total flight path. This function (shown as Eq. (5) is driven by three shaping parameters β, σ, and μ. This function takes on the shape of the normal distribution (scaled by β) to allow for a distinct, flexible peak deviation and gradual decline to zero deviation at both ends (the fixed flight path points).

$$\delta_{j_{\{s,e\}}} = \frac{\beta_{\{s,e\}}}{\sqrt{2\pi\sigma^2_{\{s,e\}}}} \exp\left(-\frac{\left(j - \mu_{\{s,e\}}\right)^2}{2\sigma^2_{\{s,e\}}}\right) \tag{5}$$

The shape of each alteration set is independent within its respective plane. After the alteration set is generated by Eq. (4) for each plane, the sets are merged into one new path by averaging the alteration vectors. The degree to which a node will be influenced by an alteration plane is weighted by a node's distance to that plane's respective anchor point ($\vec{o}_s$ or $\vec{o}_e$) using Eq. (6).

$$\vec{n}'_j = \frac{\|\vec{d}_{j_e}\|}{\|\vec{d}_{j_s}\| + \|\vec{d}_{j_e}\|}\vec{n}'_{j_s} + \left(1 - \frac{\|\vec{d}_{j_e}\|}{\|\vec{d}_{j_s}\| + \|\vec{d}_{j_e}\|}\right)\vec{n}'_{j_e} \tag{6}$$

Note that the parameters β, σ, and μ are what define the shape of the $\vec{n}'_j$ nodes. To initialize these parameters, educated guesses are made to approximate a good fit based on the initial contact using the heuristic Eqs. (7), (8), and (9).

$$\vec{\beta}_{\{s,e\}_{i=0}} = 16\min(\vec{d}_{\{s,e\}}) \tag{7}$$

$$\vec{\sigma}_{\{s,e\}_{i=0}} = \frac{1}{16} j_{max} \tag{8}$$

$$\vec{\mu}_{\{s,e\}_{i=0}} = j_{\min(\vec{d}_{\{s,e\}})} \tag{9}$$

After initial parameters are heuristically selected, they are mutated slightly to form a batch of paths that can be attempted and assessed. The purpose of this is to eventually replace the old parameters with the best-performing of this batch. Because each attempted flight requires the passage of a cycle, the batches are kept small (no more than 10) for faster improvement. Depending on cycle times and time constraints, an operator may find it infeasible to take something like 20 flights for each iteration's batch. For continuous improvement, a batch size of 1 may be selected, but this may lead to settling at a local optimum rather than exploring for a better overall shape. The larger the batch size, the more exploratory the algorithm would be before selecting an optimal parent parameter to select from. For these reasons, a batch size of 8 was selected during the experiments.

According to the batch size, sets of randomly mutated shaping parameters $\beta'_{b_{\{s,e\}}}$, $\sigma'_{b_{\{s,e\}}}$, and $\mu'_{b_{\{s,e\}}}$ are generated from their parent using Eq. (10). Note that randn(B) is a PyTorch function which outputs B normally distributed random numbers with a mean of 0 and standard deviation of 1.

$$\{\beta, \sigma, \mu\}'_{\{s,e\}} = \{\beta, \sigma, \mu\}_{\{s,e\}}\left(1 + \eta\left[\text{randn}(B)\right]\right) \tag{10}$$

Once the batch of parameter sets are determined, the change to the each mutated set's path is computed per node as explained by Eqs. (4), (5), and (6). With the resulting contact events from the modified path nodes $\vec{n}'_j$ driven by $\{\beta, \sigma, \mu\}'_{\{s,e\}}$, the performance of each path is now ready to be assessed.

E. Flying Batch of Valid Paths

Before flying each mutated path $\left[\vec{n}'_j\right]_b$, a flight must first be assessed to ensure that the proposed alterations are within the vehicle's flight envelope. This flight envelope, defined by parameters v_{min} and v_{max}, will set constraints to node alterations as shown by Fig. (7).

Conceptually, the cones of possibility in Fig. (7) are a function of the flight envelope, where all points along possible paths will satisfy Eq. (11), which defines the velocity of a given drone in terms of the two spatial axes and time axis. This connects the fleet's vehicle constraints to the allowable path shape.

$$v_{min} < \sqrt{\left(\frac{\partial x}{\partial t}\right)^2 + \left(\frac{\partial y}{\partial t}\right)^2} < v_{max} \tag{11}$$

In the numerical implementation, for each node segment, a Boolean value is generated if Eq. (12) holds true.

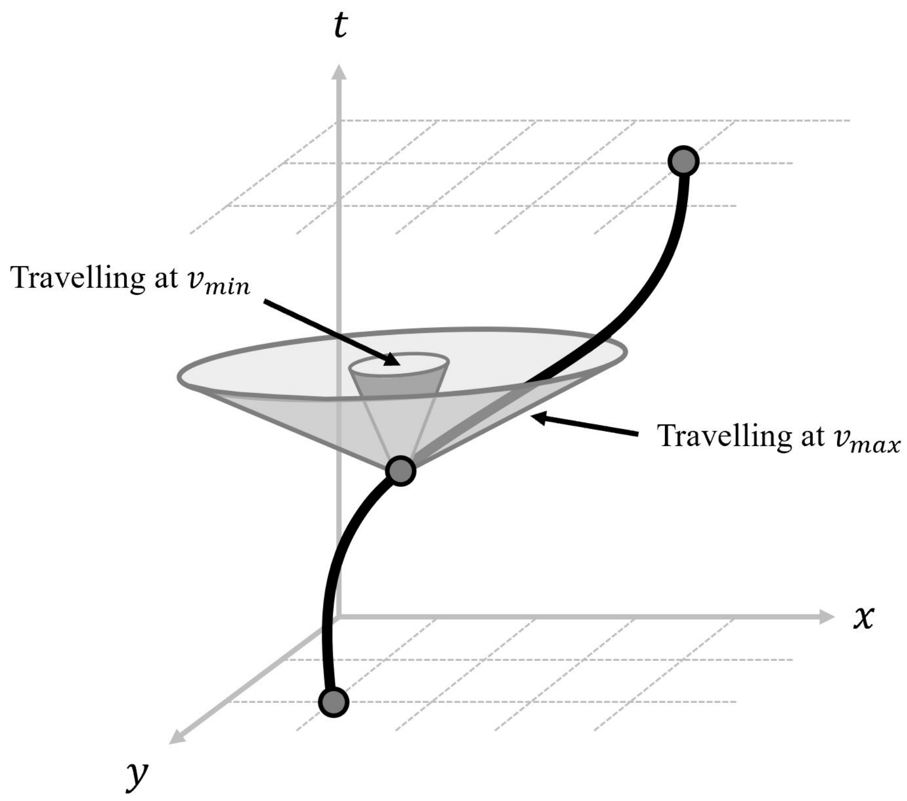

Fig. 7 A flight path node bound by vehicle flight envelope

$$v_{min} < \frac{\sqrt{\left(\vec{n}[x]'_{j+1} - \vec{n}[x]'_j\right)^2 + \left(\vec{n}[y]'_{j+1} - \vec{n}[y]'_j\right)^2}}{\vec{n}[t]'_{j+1} - \vec{n}[t]'_j} < v_{max} \tag{12}$$

If all the nodes for a proposed path passed through Eq. (12) return True, the path makes it into the batch to fly. If none of the parameters make it into an iteration's batch, a new set of proposed paths is generated with a more conservative mutation parameter. This continues until a proposed mutated path makes it into the flying batch or the number of attempts time out. If this time out happens, the algorithm terminates and requests that the user increase the margin between the original path and the flight envelope. For example, if a drone's original flight path is flying at its maximum velocity for the entirety of the cruise, any marginal alteration to the path with respect to the time axis would make that alteration invalid to fly.

F. Determine Performance

Performance is determined by the percentage of the flight time where the drone is in contact with target object. The flight time is computed by Eqs. (13) and (14), or as Algorithm (1).

$$\{k_{valid}\} = k \in K, \text{if } (\vec{o}_{k+1}[t] - \vec{o}_k[t] < t_{min}) \ \& \ (k \neq k_{max}) \tag{13}$$

$$P_b = \frac{1}{\left[\vec{n}[t]_{j_{max}} - \vec{n}[t]_{j_{min}}\right]} \sum_{k=\{k_{valid}\}} t_k \tag{14}$$

The parameters $\{\beta, \sigma, \mu\}_b$ associated with $\max(P_b)$ will be the parent parameter set for the next iteration. Also, a mutation of the original informed parameters $\{\beta_0, \sigma_0, \mu_0\}$ will continuously have one child slot reserved in the batch. This ensures that there is a second, informed point of reference from which a high-performing child-path might arise.

Algorithm 1 Calculating Performance for a Mutated Path $\left[\vec{n}'_j\right]_b$

```
INPUT: Discovered contact event nodes for child $[\vec{o}_k]_b$; total flight time $\vec{n}[t]_{j_{max}} - \vec{n}[t]_{j_{min}}$.
OUTPUT: Performance for child $P_b$.
  Contact Time $t_K = 0$
  Continuous contact threshold $t_{min}$
  for all $k$ in $(0, K.length - 1)$ do
    if $\vec{o}_{k+1}[t] - \vec{o}_k[t] < t_{min}$ then
      $t_K = t_K + \vec{o}_{k+1}[t] - \vec{o}_k[t]$
    end if
  end for
  Performance $P_b = t_K / \left[\vec{n}[t]_{j_{max}} - \vec{n}[t]_{j_{min}}\right]$
```

After many iterations, however, this "oddball" child spawned from the original parameters is unlikely to perform better than its highly-evolved counterparts. However, in the beginning, having this second opinion ensures that all the children don't stagnate at one local point of high performance.

The continuous contact threshold t_{min} is defined as the amount of time for which two adjacent instances of contact with a target object are counted as continuous. For a given path, the drone may come into contact with the target multiple times. The running sum t_K is kept to tally only the instances where the object is continuously observed for a given amount of time. This process is shown conceptually in Fig. (8).

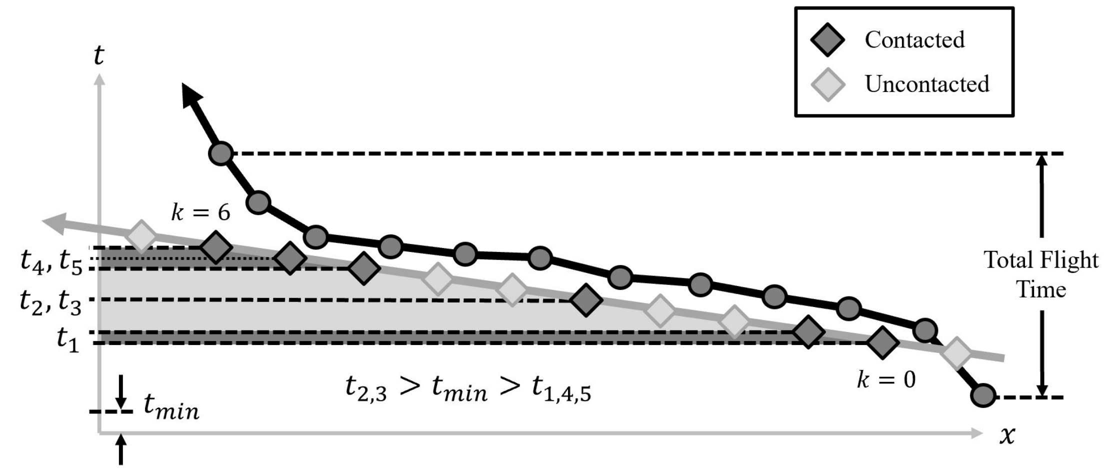

Fig. 8 Conceptual diagram for calculating P_b.

G. Dynamic Hyperparameters

The optimum may be reached faster by increasing the mutation factor η. However, a large η will likely deviate the path more than necessary. Additionally, the resulting large alterations to the original path are more likely to generate invalid proposed paths outside of the flight envelope.

On the other hand, an η which is too small will require many iterations or flyable batches to significantly increase performance. In practice, this may be too expensive or outside of survey schedules.

The correct η must be selected as a response to certain conditions. If performance has not significantly improved between iterations, η is doubled. Otherwise, η is decayed by a factor of 0.8 per successful iteration until it reaches one fifth of the original value. This continues until the minimum allowable η is reached.

Within an iteration, if a proposed batch does not contain any flyable proposed paths, η is lowered to generate more conservative paths. This is discussed at the end of subsection E.

VI. Experimental Results

After presenting an example 2-dimensional set of results to clearly demonstrate the methodology, the next two subsection will discuss the path alteration applied to a small simulated drone fleet in a test geography. Figure (9) displays a map showing that there are two initial contact events at the start of the simulation. Only the portions of the flight schedule which contain the contact event are affected by the algorithm.

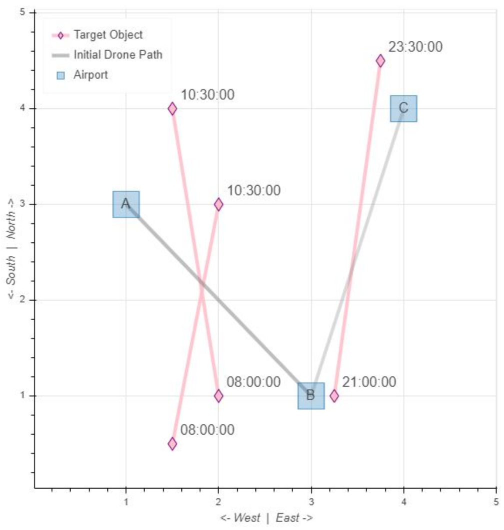

Fig. 9 Map of simulation's initial conditions.

The annotated times at each object point give information of the target's direction of travel along the time-axis. In this simulated geography, drones travel between airports A, B, and C. Because of timing, only one of the two western simulated targets is contacted. This object encounters the drone's range of sight at a steeper angle than the later, eastern encounter. This will lead to differences in the initial heuristic approximations $\{\beta, \sigma, \mu\}_0$. The 3-dimensional results after 80 generations is shown in Fig. (10), where each altered leg is further examined in the later subsections.

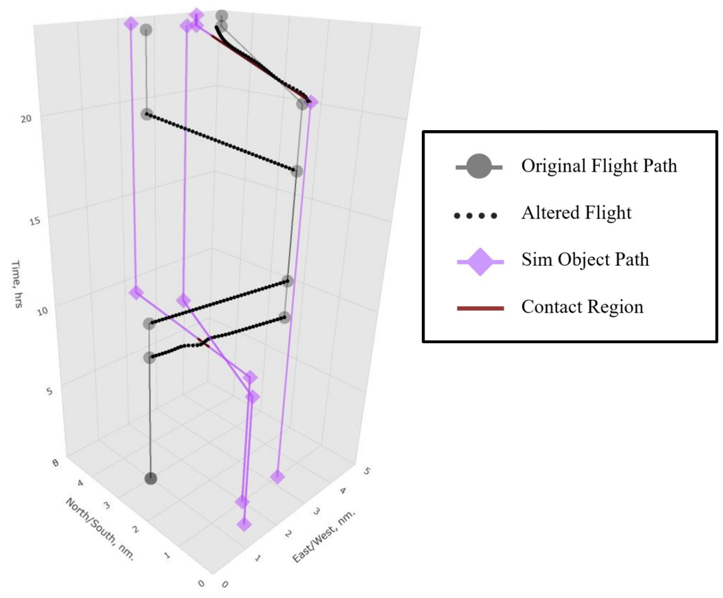

Fig. 10 Total simulation results after 80 generations.

A. Introductory 2-D Example

On static, non-interactive images (such as those displayed in this work), it is difficult for a reader to interpret the plotted 3-D space. For this reason, the results of a 2-D pseudo-experiment are shown here. The purpose of this subsection is to provide context and introduce the reader to the plotted results of the following sections.

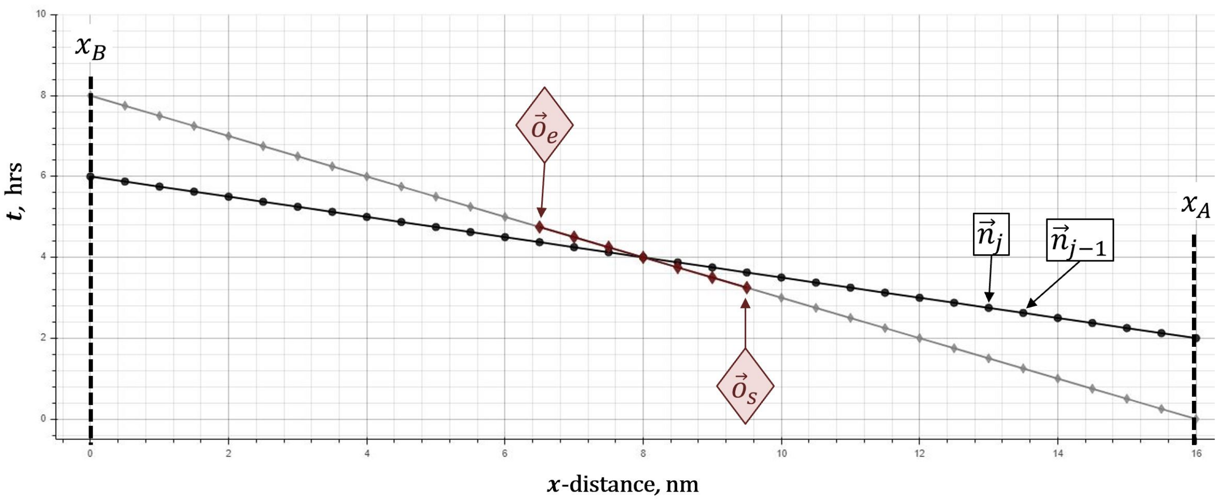

Fig. 11 Contact event.

Suppose there are two locations in a fictional 1-dimensional geography, called x_A and x_B. Both the simulated target and the drone are travelling from x_A to x_B. The drone is travelling faster than the target, so while the target did depart x_A at an earlier time, it arrives at x_B later than the drone. The resulting interaction between the drone and target is plotted in Fig. (11).

Upon first contact, the the anchor nodes are chosen and distance vectors for each node are created. An orthonormal basis is created for each plane anchored by $\{\vec{o}_s, \vec{o}_e\}$, as illustrated by Fig. (12). Note, in this 2-D example, these two anchored planes are co-planar, but may not necessarily have the same basis. Here they have parallel bases (but opposing planar normal vectors) since the paths are straight (the second basis vector is directed along the path segment).

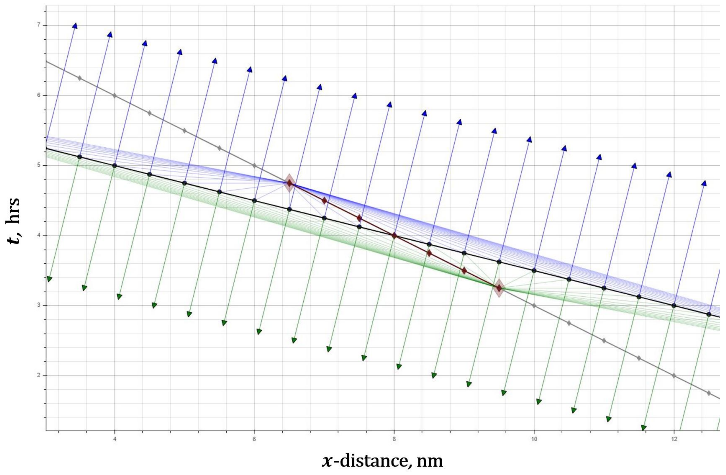

Fig. 12 Creating orthogonal bases for each $\{\vec{o}_s, \vec{o}_e\}$.

Informed parameters are now chosen and mutated to create a batch of proposed paths. The paths which violate the flight envelope are eliminated, as shown in Fig. (13).

The best-performing path is selected after flying the batch. From here, the parameters which created this path

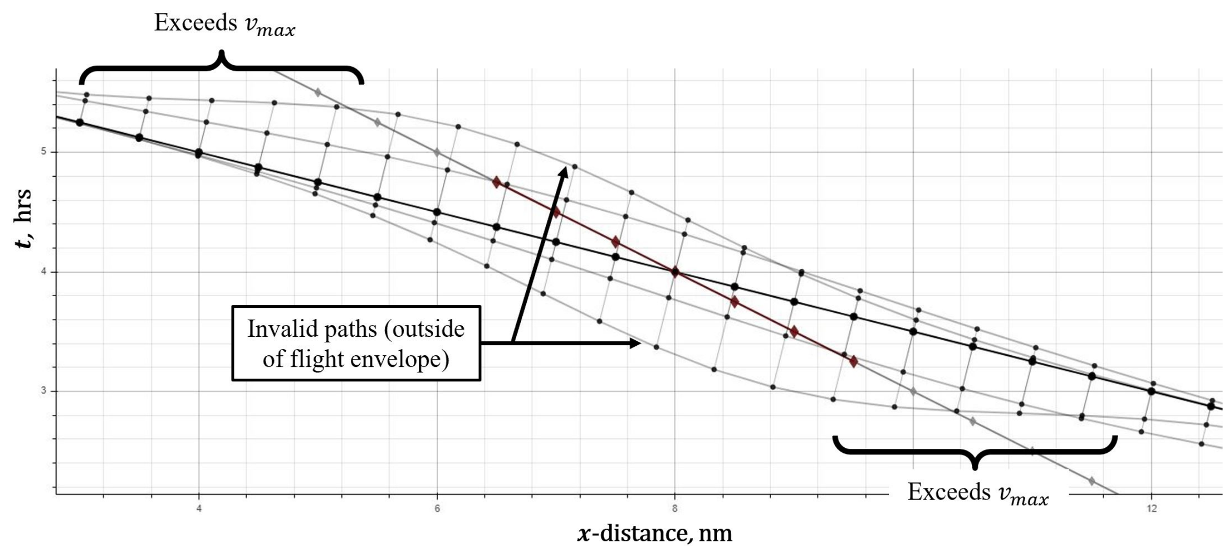

Fig. 13 Two child-pairs with proposed flight path alterations.

are mutated, and the iteration loop is continued. The next iteration will mutate the parameters which created the best-performing flight path shown in Fig. (14).

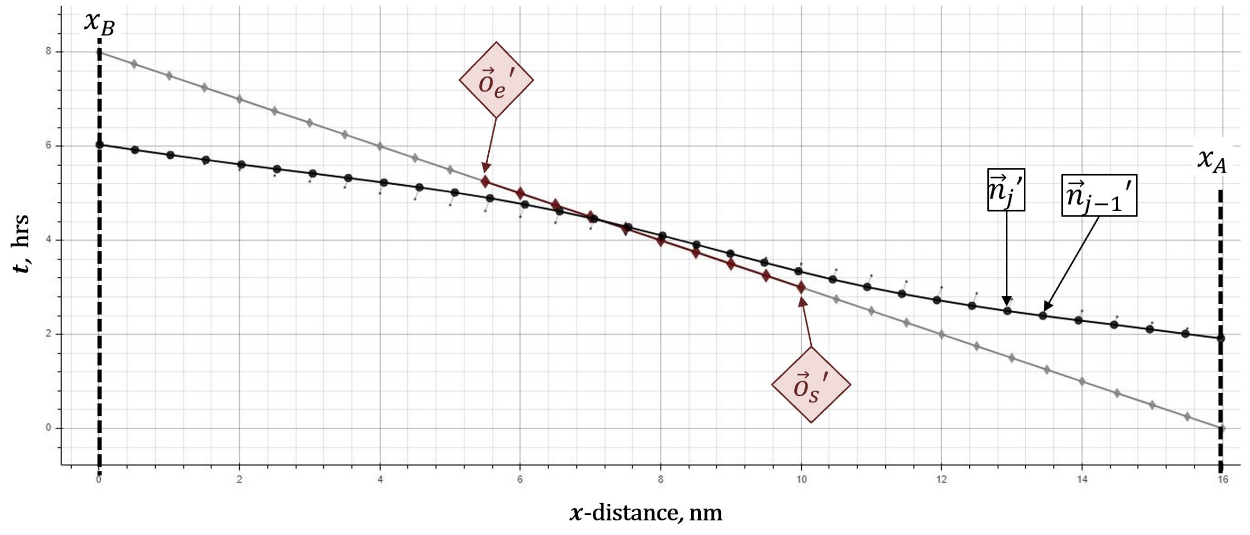

Fig. 14 Best performing flight path at the end of the iteration.

B. Parallel Encounter

The first 3-D experiment presented is an encounter with a target travelling along a similar path to that of the drone. The expectation is that the drone path should be modified so that it creates a slight S-curve to match the path of the target. Figure (15) illustrates the informed distribution generated upon first contact (the value of β is doubled for display clarity) and how mutations are handled to produce a high-performing path.

A history of the evolving parameters driving the distributions of $\delta_{\{s,e\}}$ are plotted in Fig. (16). The corresponding performance measures for each set of winning parameters are plotted in Fig. (17). There is a 13% improvement after 3 generations (24 flights), and a 58% improvement after 80 generations (640 flights).

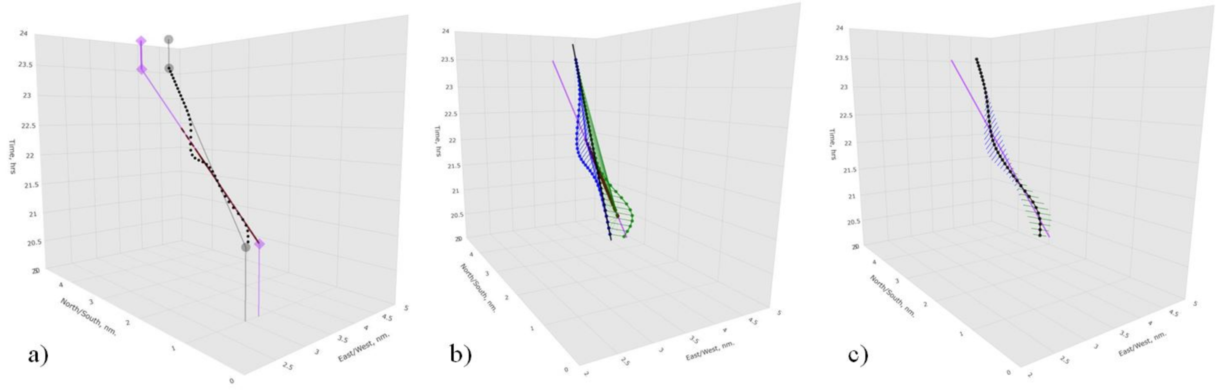

Fig. 15 a) initial heuristic; b) successive mutation; c) resulting high-performing path

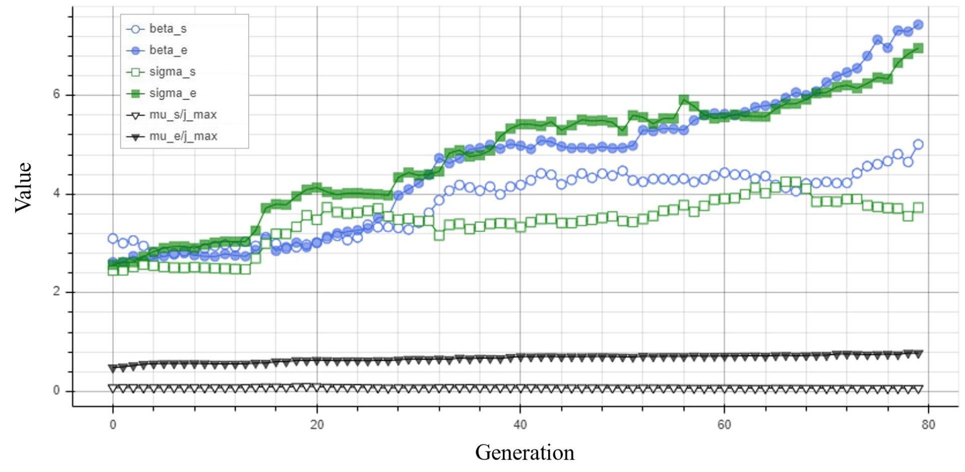

Fig. 16 Parameter evolution for Flight 2 at time 21:00 hrs

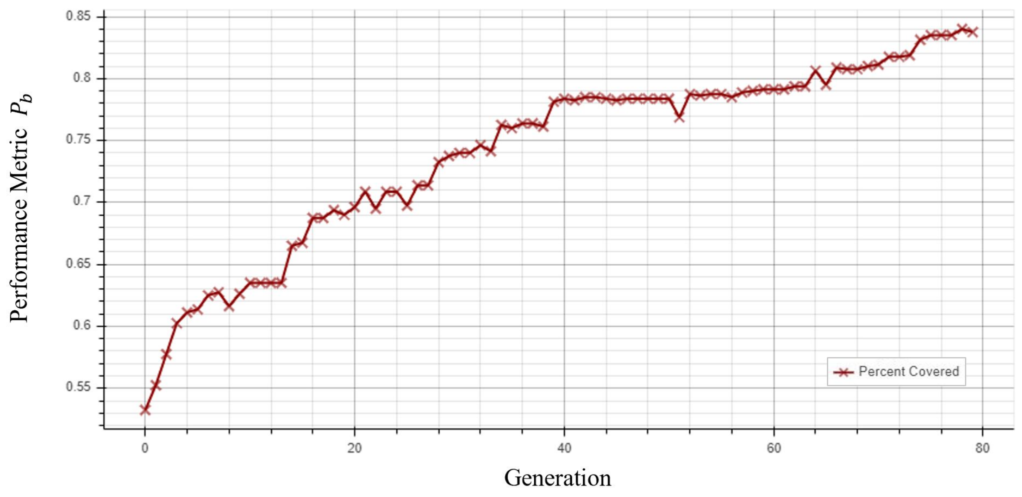

Fig. 17 Performance of best parameters for Flight 2 at time 21:00 hrs

C. Perpendicular Encounter

This next experiment was an encounter with an object travelling perpendicular to the flight path. The resulting altered path is expected to contain a sharp S-curve that attempts to capture whatever contact it can for a brief amount of time.

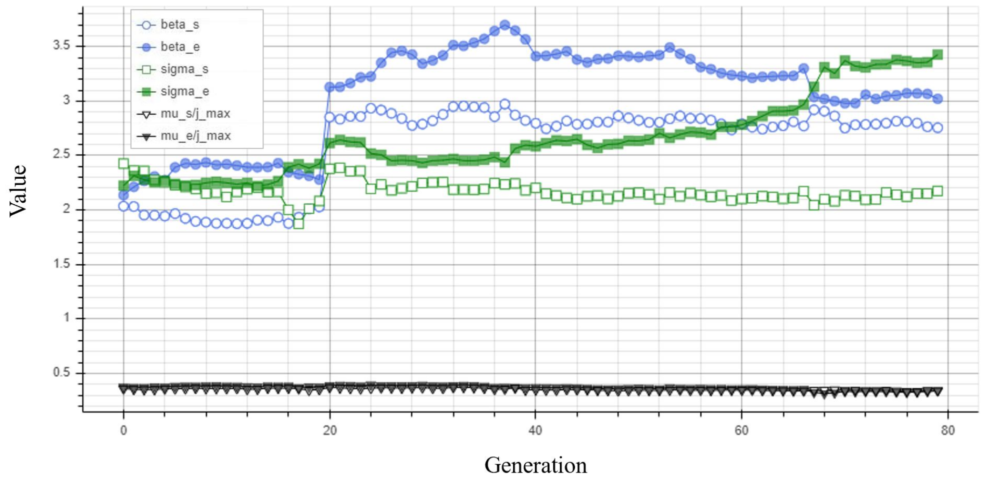

Fig. 18 Parameter evolution for Flight 1 at time 08:00 hrs

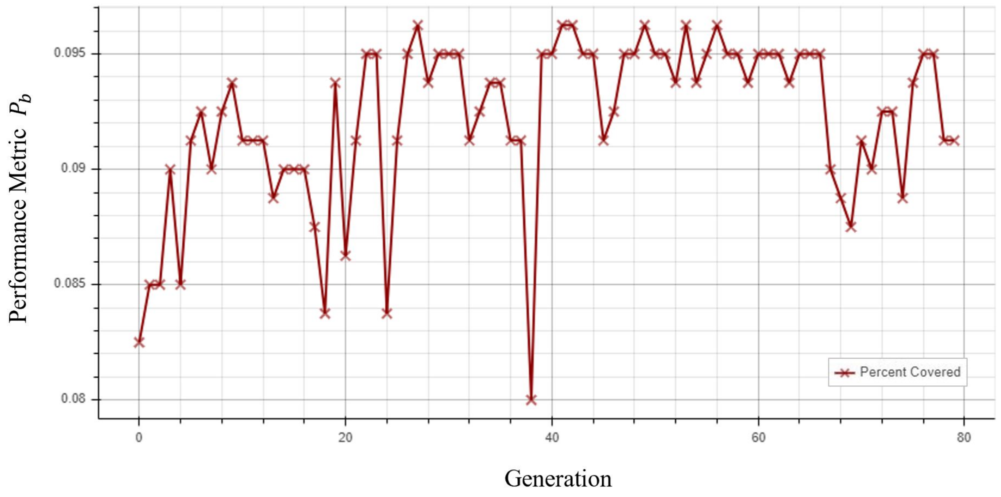

Fig. 19 Performance of best parameters for Flight 1 at time 08:00 hrs

It can be seen in Fig. (19) that there is not an appreciable change in performance (an 11% increase after 640 flights). Because of the steep angle, μ_s and μ_e are similar and stay at nearly equal values throughout. Perhaps a revision of this heuristic could be informed by the angle of approach and drive the starting μ set further apart so that the path is given time to "prepare" for the encounter. However, this is likely to be a costly deviation from the original path, so this would need to be counter-balanced with another optimization for path deviation, to be discussed in section VII.

VII. Future Work

A genetic algorithm was selected due to the unpredictable, non-differentiable relationship between the mutated shaping parameters and the performance measure (which was the result of a simulation unknown to the algorithm). However, there is a known, differentiable relationship between these parameters (or associated node locations) and the path length used. While η was scheduled so that the path deviation would not be too severe, the algorithm can be extended to directly optimize the best performing path for travel length on each iteration. Eqs. (15) and (16) show the underlying process of this extension.

$$C_i = \frac{1}{j_{max} - j_{min}} \sum_{j=j_{min}}^{j_{max}} \frac{\|\vec{n}_{i_j} - \vec{n}_{0_j}\|}{\|\vec{n}_{0_{j_{min}}} - \vec{n}_{0_{j_{max}}}\|} \text{ where } \vec{n}_{i_j} = f\left(\beta_i, \sigma_i, \mu_i\right) \tag{15}$$

$$\{\beta, \sigma, \mu\}_{i+1} = \{\beta, \sigma, \mu\}_i - \alpha \left(\frac{\partial C_i}{\partial \{\beta, \sigma, \mu\}_i}\right), \text{ if } P_b\left(\beta, \sigma, \mu\right)_{i+1} = P_b\left(\beta, \sigma, \mu\right)_i \tag{16}$$

Figure (20) shows where the extension should be located in the current implementation. The extension should be run at the end of each generation (iteration). In this way, the shaping parameters are further optimized within the differentiable bounds before the next mutation. This conserves energy and unnecessary wear-and-tear on the vehicle.

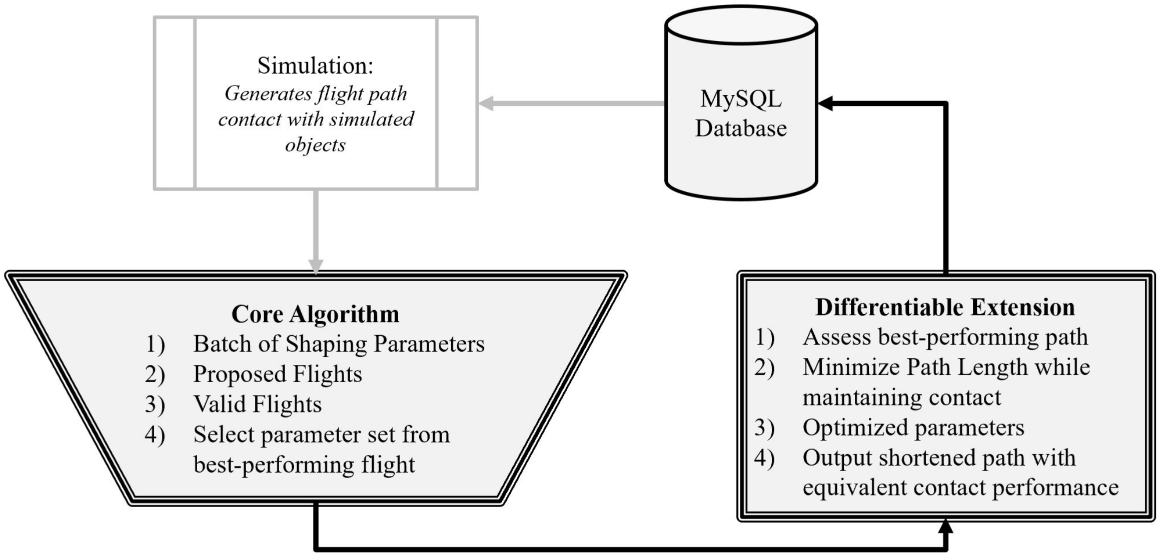
Simulation:
Generates flight path contact with simulated objects
MySQL Database
Core Algorithm
1) Batch of Shaping Parameters
2) Proposed Flights
3) Valid Flights
4) Select parameter set from best-performing flight
Differentiable Extension
1) Assess best-performing path
2) Minimize Path Length while maintaining contact
3) Optimized parameters
4) Output shortened path with equivalent contact performance

Fig. 20 How proposed extension would interact with current implementation.

One difficulty would be that this algorithm would create a slight disconnect between the shaping parameters and the resulting parent path, but genetic algorithms should be able to handle unknown circumstances anyway. Additionally, this extended differentiable algorithm will be bounded such that any optimized path must contain all the contact nodes.

The implementation of this extension is facilitated by the fact that all the node alteration operations described in this paper are conducted on PyTorch Tensor objects. A machine learning framework, PyTorch preserves information of the gradients for each forward operation as a computation graph. This preserved information is then used to automatically compute the gradients used to descend the cost function (path length).

VIII. Conclusion

This work presented a methodology for tracking moving objects with regular schedules by means of a genetic algorithm. Suggested applications of this algorithm include traffic validation, ecological observation, and atmospheric survey as a result of regular activity. Any kind of regular motion to be tracked would be applicable.

This algorithm evolved the shaping parameters of each flight path that happened upon an object according to a performance metric. This metric is determined by the amount of time the drone remains in contact with the target object. This shape is manipulated inside a volume which represents a geographical region extruded over time. The parameters are initialized by a heuristic which relies on two anchor points: the instance of first contact and the last-seen instance of the target object. From here alteration planes are generated using the distance vectors and path segment as the basis vectors. In this way, the change vectors for each node are oriented optimally.

After presenting a two-dimensional example of results, the results for two contact events in the full three-dimensional simulation were presented. The two encounters occurred on different sides of the geography: in the west, there was a perpendicular encounter, with an 11% increase in the performance metric after 640 flights. As expected, the parallel encounter in the east fared better, with a 13% increase in the performance metric after only 24 flights, and a 58% improvement by flight 640 at the end of the simulation.

References

[1] Amorim, R. M. D., Mogensen, P. E., Sørensen, T. B., Kovács, I., and Wigard, J., "Pathloss Measurements and Modeling for UAVs Connected to Cellular Networks," *IEEE VTS Vehicular Technology Conference*, IEEE, 2017. doi:10.1109/VTCSpring.2017.8108204.

[2] Zorbas, D., Razafindralambo, T., Di Puglia Pugliese, L., and Guerriero, F., "Energy Efficient Mobile Target Tracking Using Flying Drones," *The 4th International Conference on Ambient Systems, Networks and Technologies*, Vol. 19, 2013, pp. 80, 87. doi:10.1016/j.procs.2013.06.016.

[3] Barth, D., "The bright side of sitting in traffic: Crowdsourcing road congestion data," `https://googleblog.blogspot.com/2009/08/bright-side-of-sitting-in-traffic.html`, 2009. Accessed: 2019-04-20.

[4] McEvoy, J. F., Hall, G. P., and McDonald, P. G., "Evaluation of unmanned aerial vehicle shape, flight path and camera type for waterfowl surveys: disturbance effects and species recognition," *The Journal of Life and Environmental Sciences*, 2016. doi:10.7717/peerj.1831.

[5] Kerr, J. T., and Ostrovsky, M., "From space to species: ecological applications for remote sensing," *Trends in Ecology & Evolution*, Vol. 18, No. 6, 2003, pp. 299–305. doi:10.1016/S0169-5347(03)00071-5.

[6] M, C. J., and et al., "Satellite monitoring of cyanobacterial harmful algal bloom frequency in recreational waters and drinking water sources," *Ecological Indicators*, Vol. 80, 2017, pp. 84–95. doi:10.1016/j.ecolind.2017.04.046.

[7] Khan, M., Heurtefeux, K., Mohamed, A., Harras, K. A., and Hassan, M. M., "Mobile Target Coverage and Tracking on Drone-Be-Gone UAV Cyber-Physical Testbed," *IEEE Systems Journal*, Vol. 12, No. 4, 2018, pp. 3485–3496. doi:10.1109/JSYST.2017.2777866.

[8] Sun, L., Baek, S., and Pack, D., "Distributed Probabilistic Search and Tracking of Agile Mobile Ground Targets Using a Network of Unmanned Aerial Vehicles," *Human Behavior Understanding in Networked Sensing: Theory and Applications of Networks of Sensors*, edited by P. Spagnol, P. Luigi Mazzeo, and C. Distante, Springer International Publishing, Switzerland, 2014, Chap. 14, 1st ed., pp. 301–319. doi:10.1007/978-3-319-10807-0_14.