Modifying Operational Fleet Paths to Better Track Moving Objects

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

Insert Abstract Here. It should be one paragraph long (not an introduction) and complete in itself (no reference numbers). It should indicate subjects dealt with in the paper and state the objectives of the investigation. Newly observed facts and conclusions of the experiment or argument discussed in the paper must be stated in summary form; readers should not have to read the paper to understand the abstract. The abstract should be bold, indented 3 picas (1/2") on each side, and separated from the rest of the document by blank lines above and below the abstract text..

I. Nomenclature

α = hyperparameter to determine tuning update
β = tuned scaling parameter
δ = scaling vector for alteration vector
λ = hyperparameter for determining total cost
θ = angle between distance vector and simulated object path
σ = tuned standard deviation parameter of activation function
C = cost metric
$\vec{d}$ = distance vector from node to anchor point
e = denotes last "known" point location for simulated object point
i = iteration of an initial or altered path
j = identifier of a node for a given path leg
$\vec{n}$ = node location
$\vec{o}$ = target object location
s = denotes first "known" point location for simulated object path
t = location along time axis
x = location along East/West axis
y = location along North/South axis

II. Introduction

This work establishes a methodology for tracking moving objects with an existing fleet of aircraft. 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 for their given schedules, or air pollution for a given a week. In real-world applications of this methodology, a fleet of unmanned aerial vehicles is the intended platform for the following reasons:

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

Regardless, the algorithm is agnostic to whatever vehicle platform is used so long as its path can be freely changed. However, to meet this criteria, this will almost certainly be an aerial platform (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 the unknown guidance variables accompanied by navigating through non-flat terrain.

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

The platform's minimum and maximum velocity is defined to place limits on how a path can be altered as a function of time. Additionally, a constant "sight" parameter is used to establish the spatial range of sight for the platform aircraft. In reality, this sight parameter is not constant throughout the flight, since low-altitude flight profiles are sensitive to large ground features such as terrain or tall buildings obstructing the full view. It is also important to note that if the altitude is non-constant, the cone of sight increases with increased altitude.[1]

III. Background

This methodology intends to approach the problem of object tracking by visualizing the entire system as a 3-dimensional volume: the x-y plane is the physical region over which the fleet is to operate, and the z-axis is the time-domain. The intent for this setup is for basic principles of calculus and linear algebra to guide a heuristic designed to better match and track objects moving in the same geographical space and time domain. This representation can intuitively visualize flight paths and incidences of contact with the target object, as illustrated in Fig. (1). With this 3-dimensional definition, "location" vectors will not simply refer to spatial location, but rather the spatial location in time, in the form of $\vec{n} = [x, y, t]$.

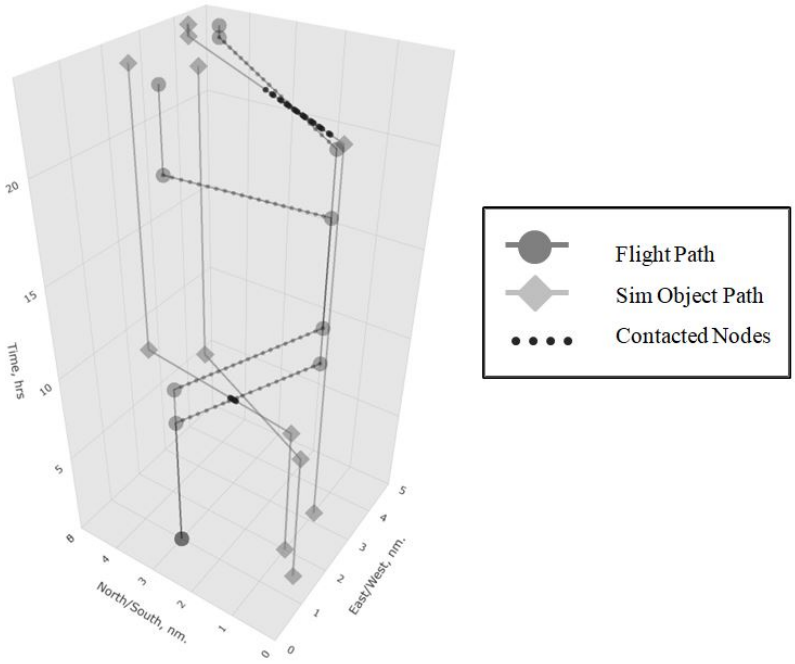

Fig. 1 Methodology's introduced representation of two flights and three objects.

Physical flight parameters can bound the shape an altered flight may take. For example, introductory calculus gives Eq. (1), which defines the velocity of a given object in terms of the three spatial axes. This is connects the fleet requirement of minimum and maximum allowable vehicle velocities.

$$\vec{v} = \sqrt{\left(\frac{\partial x}{\partial t}\right)^2 + \left(\frac{\partial y}{\partial t}\right)^2} \tag{1}$$

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. This is unrealistic, since the vehicle cannot peer backwards or forwards in time, but is considered a numerical artifact of the finite steps for this simulation. Any simulated target object within this cylindrical volume is considered to have made "contact" with the manipulable path node.

IV. Problem Statement

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 at which 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 [2]. 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. Atmospheric Survey

While the previous use case is used to validate existing data, vehicles 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.

C. Migration Patterns

Another potential use case for this

V. Methodology

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

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. 2, 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 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 (structured query language database).

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.

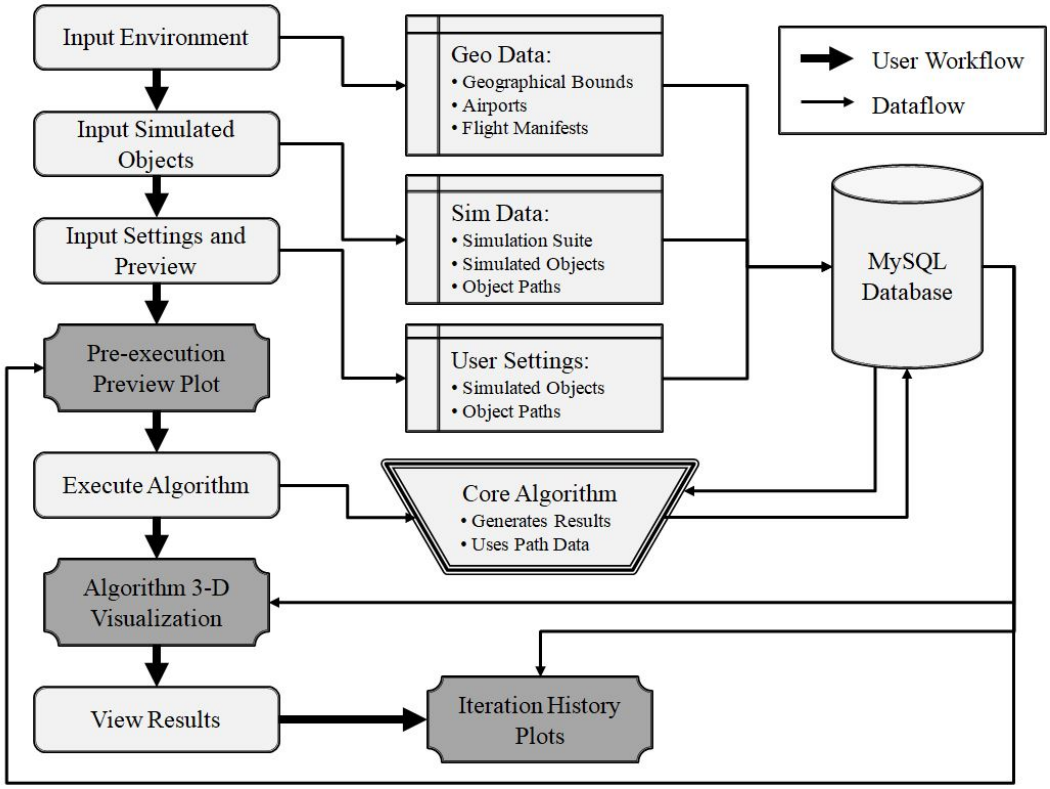

Fig. 2 Application architecture built to interface with core algorithm.

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

B. Algorithm Architecture

The algorithm is designed as it would operate in reality: the only information that is passed into it 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. From here, the following sub-sections will explain what each component of the core algorithm in Fig. 3 does to help compute what the next iteration of the altered path will be.

1. Assessing Contact Points

Upon receiving information about the instances of contact with the target object, the first operation is to assess some basic properties about this event. These are inputs required to inform a devised heuristic. For each flight leg, the first and last seen location of the sim object is recorded. These two points serve as anchoring points for distance vectors created between these two locations and the locations of each path node. The alteration vector for each node will be determined from the components of each node's two respective distance vectors, as shown in Fig. 4.

For each node, the weighted sum of the two anchored vectors will drive its new location for the next iteration of the path. Said weighting will be determined by tuned parameters. For each iteration i, the determination of this alteration vector is of size n and is defined in Eq. (2).

$$\vec{\Delta n}_i = \delta_i \circ \vec{d}_i = \left(\delta_{d_s} \circ \delta_{d_e} \circ \delta_{\theta_s} \circ \delta_{\theta_e} \circ \delta_n\right)_i \circ \vec{d}_i \tag{2}$$

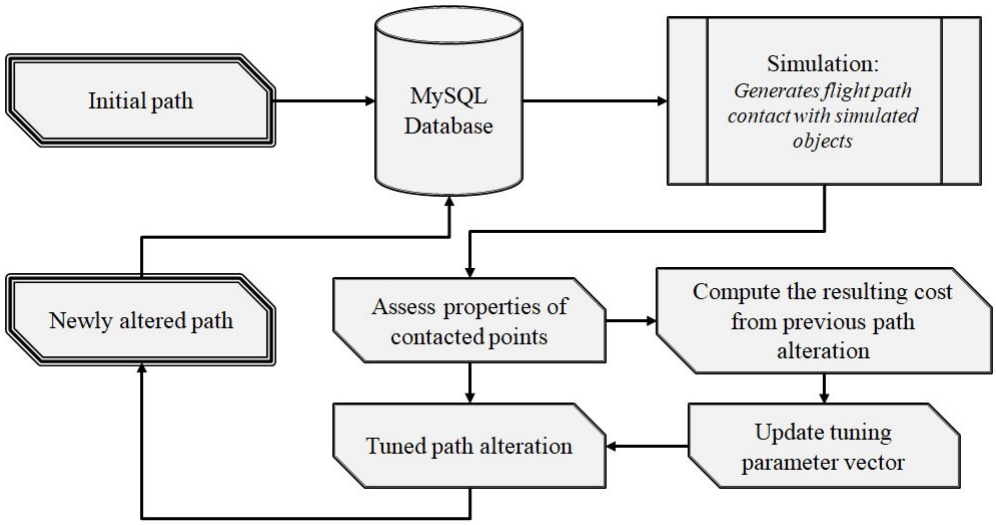

Fig. 3 Top-level architecture of core algorithm.

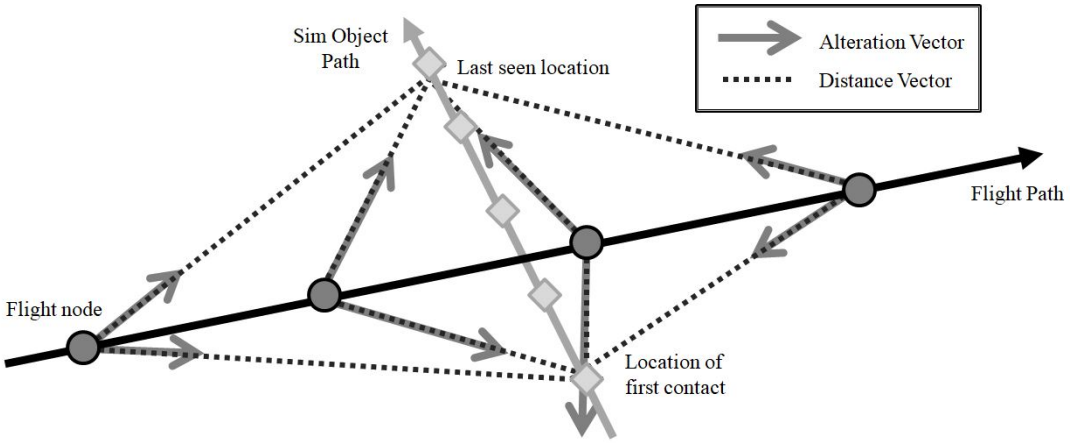

Fig. 4 Visualization of heuristic for determining alteration vector per node.

Each of the Hadamard factors of δ is determined from Eqs. (3), (4), and (5). The parameters d, θ, and j for each node are based on the contact event with the sim object and are described in the nomenclature.

$$\delta_{d_{\{s,e\}}} = \frac{1}{\sqrt{2\pi\sigma_{d^2_{\{s,e\}}}}} \exp\left(-\beta_{d_{\{s,e\}}} \frac{\left\|\vec{d}\right\|^2}{2\sigma^2_{d_{\{s,e\}}}}\right) \tag{3}$$

$$\delta_{\theta_{\{s,e\}}} = \frac{1}{\sqrt{2\pi\sigma^2_{\theta_{\{s,e\}}}}} \exp\left(-\beta_{\theta_{\{s,e\}}} \frac{\left(\theta - \frac{\pi}{2}\right)^2}{2\sigma^2_{\theta_{\{s,e\}}}}\right) \tag{4}$$

$$\delta_n = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\beta_n \frac{j^2}{2\sigma_n^2}\right) \tag{5}$$

Eqs. (3), (4), and (5) are all shaping functions for their respective property parameters, taking the form of a normal distribution about 0 or $\frac{\pi}{2}$. They are also scaled by a tuned β and σ. Through several iterations, optimally tuning the

relationships between these driving parameters is what allows $\vec{\Delta n}_i$ to be intelligently determined.

It is important to note that the δ weights can be asymmetric with respect to which sim object endpoint $\{s, e\}$ they anchor to. This allows for the algorithm to create optimal "S"-shapes in order to better track an object moving perpendicular to the flight path, as illustrated in Fig. 4. Otherwise, the nodes would cluster near the center, where its two alteration vectors would cancel from each anchor point, drawing the node to the midpoint of the two anchor points. This does not matter for a contact event where the object is moving parallel to one side of the flight, since the starting conditions are asymmetric and both anchor points draw the path towards whatever side of the flight path they happen to be on.

2. Compute Resulting Cost

To optimally tune these parameters, an evaluation function is required. There are two undesirable behaviors regarding alterations: deviation from the original path, which was known since defining the problem, and the tendency for nodes to cluster at one point (as if the alteration vectors create a sort of "gravity" about some resulting center point). The deviation is determined as the average distance from the original scheduled path using Eq. (6), which would like to be minimized.

$$C_{deviation_i} = \frac{1}{j_{max} - j_{min}} \sum_{j=j_{min}}^{j_{max}} \frac{\left\| \vec{n}_{i_j} - \vec{n}_{0_j} \right\|}{\left\| \vec{n}_{0_{j_{min}}} - \vec{n}_{0_{j_{max}}} \right\|} \tag{6}$$

The clustering is minimized by assigning a cost for inter-node distance having a standard deviation above zero, as determined by Eq. (7).

$$C_{internode_i} = \sqrt{\frac{1}{j_{max} - j_{min}} \sum_{j=j_{min}}^{j_{max}-1} \left(\left\| \vec{n}_{i_j} - \vec{n}_{i_{j+1}} \right\| - \overline{\left\| \vec{n}_{i_j} - \vec{n}_{i_{j+1}} \right\|} \right)^2} \tag{7}$$

A desirable behavior resulting from an alteration would be an increase in the length of time a simulated object can be observed by the new path. Since this is being used to determine a cost function to *minimize*, the percentage of flight time during which the object is unseen factors into the cost using Eq. (8).

$$C_{contact_i} = \frac{\vec{o}[t]_{i_e} - \vec{o}[t]_{i_s}}{\vec{n}[t]_{i_{j_{max}}} - \vec{n}[t]_{i_{j_{min}}}} \tag{8}$$

To determine the total cost of this iteration, three positive hyper-parameters are required. These are hand-tuned according the relative importance of each undesirable behavior and determine the total cost using Eq. (9).

$$C_i = \lambda_{deviation_i} C_{deviation_i} + \lambda_{internode_i} C_{internode_i} + \lambda_{contact_i} C_{contact_i} \tag{9}$$

3. Update Tuning Parameter Vector

Upon determining the cost of the new contact event as a result of the β and σ parameters, the algorithm parameters can now be optimized according to Eq. (10), a well-known equation in the subject of optimization.

$$\{\beta_{i+1}, \sigma_{i+1}\} = \{\beta_i, \sigma_i\} - \alpha \left(\frac{\partial C}{\partial \{\beta, \sigma\}} \right)_i \tag{10}$$

4. Tuned Path Alteration

The main loop is continued by updating the node vector by the same Eq. (2), but instead using the updated parameters. After the nodes are updated, each alteration is saturated, if applicable, by any physical velocity limitations before committed. In this way there are no large jumps in position within a small timeframe. Higher-order acceleration and jerk saturation limiters may also be considered, but the time steps taken by the nodes seem to be large enough to reasonably assume that such criteria are met.

After the node locations are updated, this becomes the starting point for the next iteration. The algorithm loop is summarized here.

C. Visualization

Two plotting libraries were used to visualize the algorithm results of the simulation, and the overall aesthetics of the application consisted of imported Bootstrap templates. A large part of developing this algorithm involved the visualizations, since it provided feedback on whether a path was changing in the correct way. For example, in one trial, the path points were actually moving *away* from the tracked object. From that observation, the problem was found to be a simple mistake in the calculation of the distance vector, which caused it to be pointing in the opposite direction. This mistake would have been difficult to detect so early on without a previously developed visualization.

All visualization data are extracted from the database so that the individual points do not need to be stored within the application's memory usage. Additionally, the MySQL workbench GUI allows for easy manipulation of data to test different cases. Before being processed by a plotting library, the data are stored in hash tables keyed by flight number and departure time. In this way, each flight leg has a unique set of data points and parameters to manipulate. This is done to allow the system to flexibly adapt to unique contact events, such as a perpendicular crossing, a crossing from the front left, or a parallel path on the right.

1. Bokeh 2-D Plots

The first plot displays the simulation setup before execution, showing a map of the paths taken by the target objects and fleet aircraft, as well as the location of airports (note that "airports" do not have to be airports, but can instead be fixed points in space where a vehicle must arrive at or depart from by a fixed time).

The second plot is the unpacked plotted result for each iteration of paths generated by the algorithm.

2. Plotly 3-D Plots

This is the 3-dimensional iteractive plot used to assess what the algorithm is processing per iteration. An example of this visualization is shown in Fig. 1.

VI. Experiments

A. Straight Perpendicular Encounter

B. Straight Parallel Encounter

VII. Future Work

VIII. Conclusion

Appendix

Acknowledgments

References

[1] Zorbas, D., and Razafindralambo T, G. F., Di Puglia Pugliese L, "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.

[2] 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.