Modifying Scheduled Flight Paths to Track the Motion of Regular Traffic

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
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
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
I	=	total number of main loop iterations
j	=	index 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
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

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

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

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

A. Existing Solutions

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.

B. Proposed Solution

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.

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

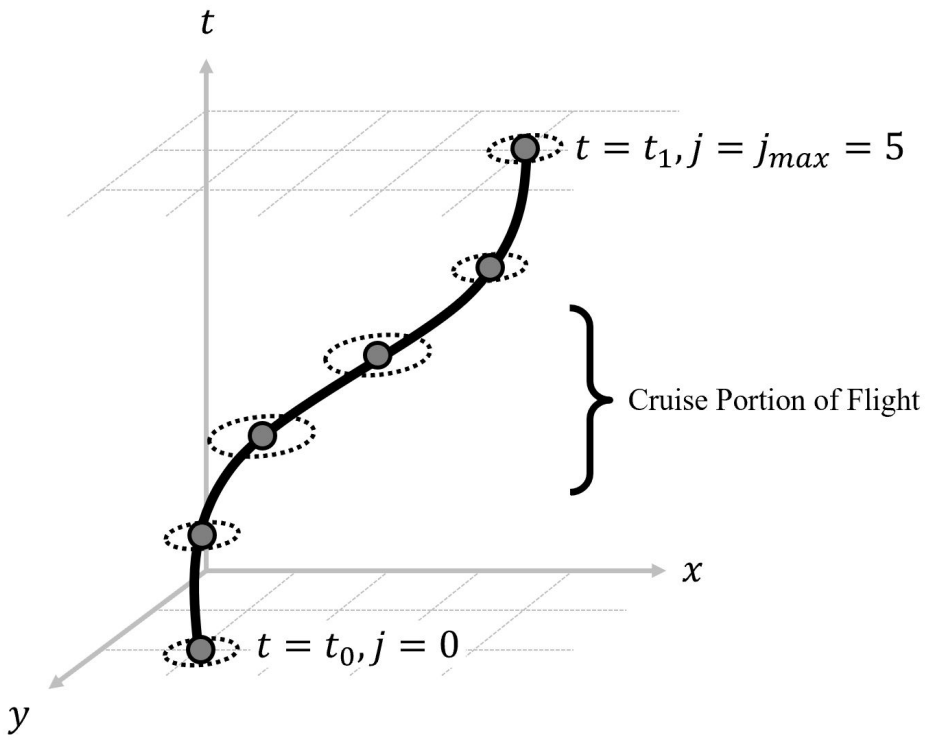

Fig. 1 Methodology's representation of a drone flying through space and time.

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).

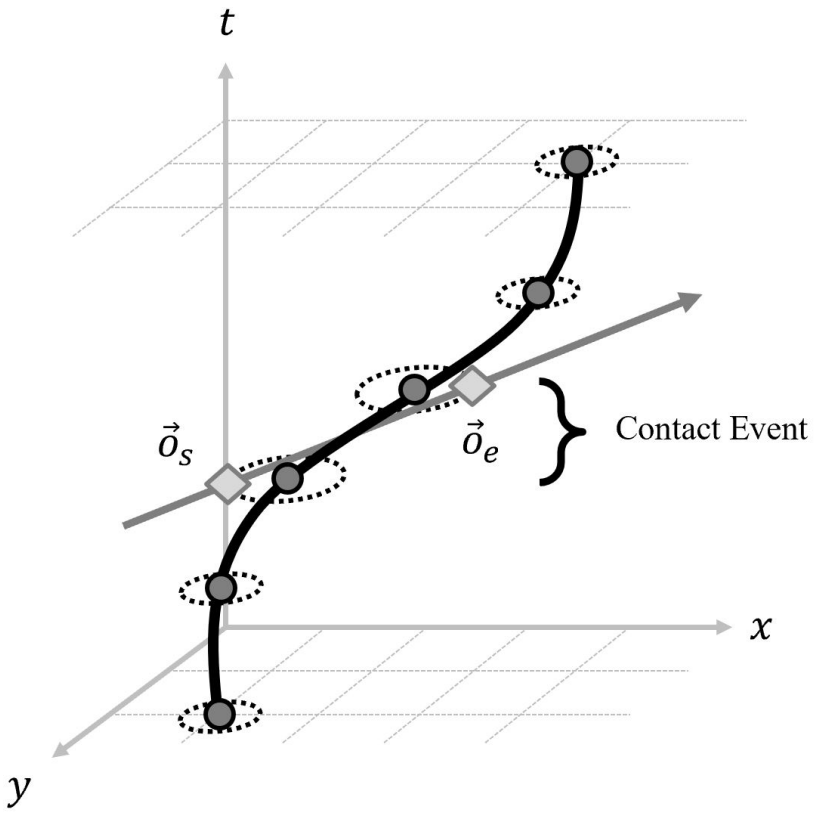

Fig. 2 Representation of contact event.

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 (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 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. 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 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.

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.

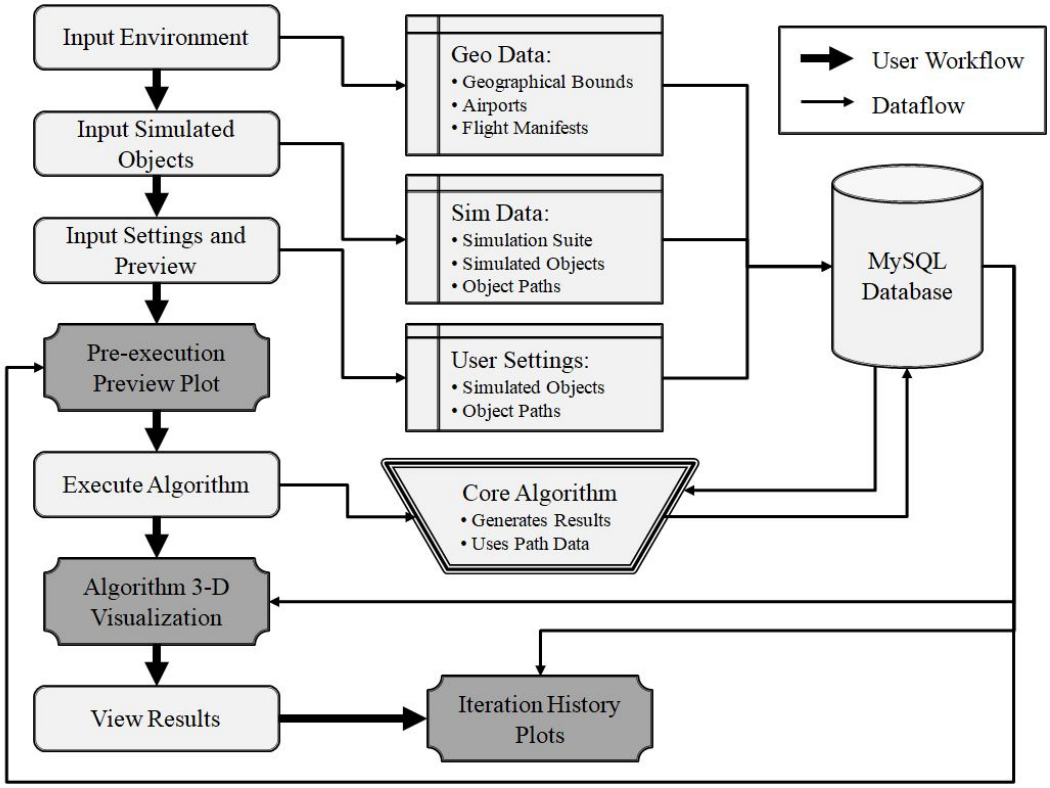

Fig. 3 Application architecture built to interface with core algorithm.

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 situated to mimic how it would operate in reality: the only information that 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. From here, the following sub-sections will explain what each component of the core algorithm in Fig. 4 does to help compute what the next iteration of the altered path will be.

C. Generating Alteration Vectors

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.

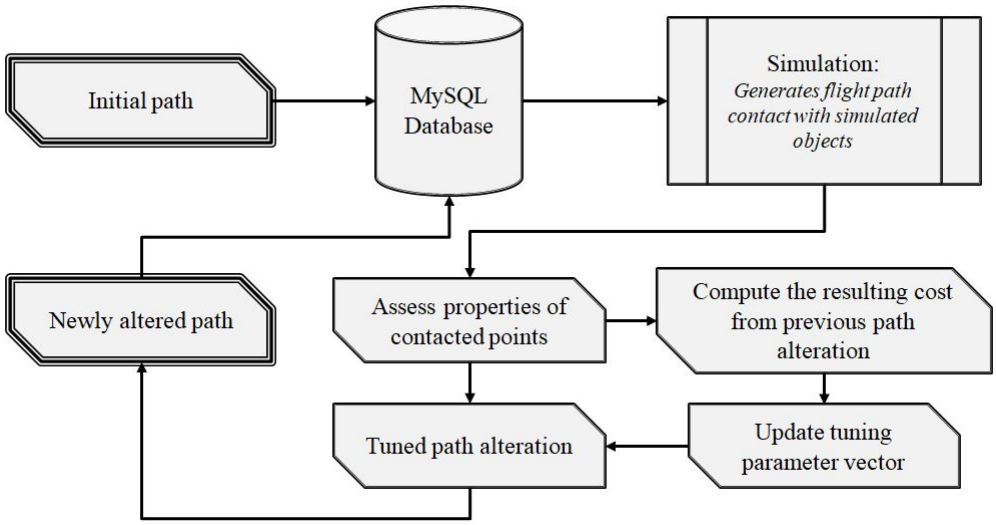

Fig. 4 Top-level architecture of core algorithm.

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 5 shows this relationship between the flight path and tracked object.

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

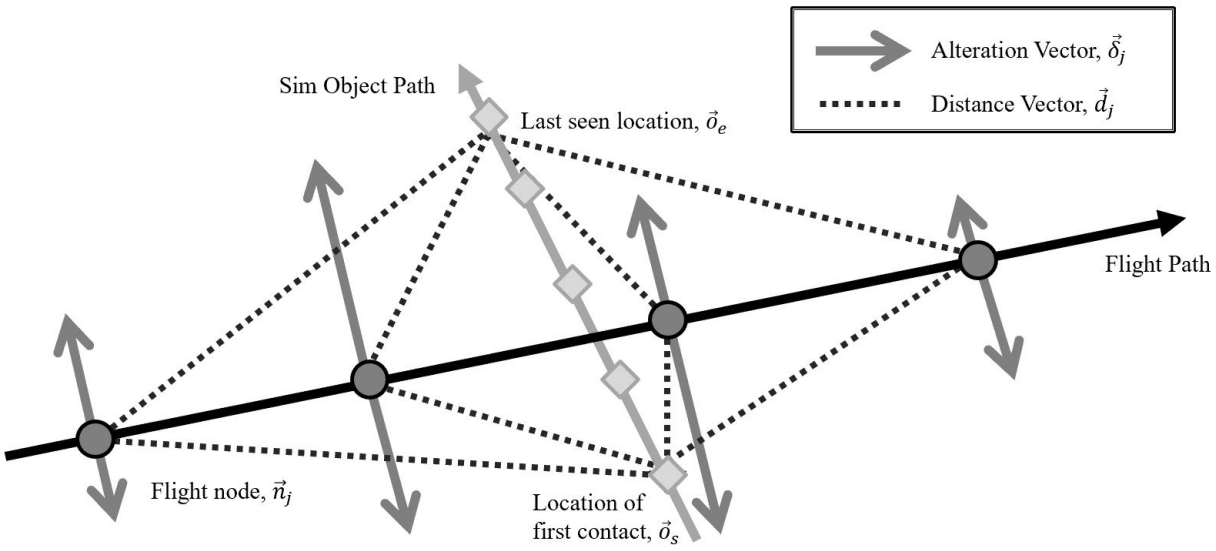

Fig. 5 Relationship between flight and object path.

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.

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

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

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.

Alteration vectors are generated along $\vec{u}_2$ for each node for both planes. This scaled $\vec{u}_2$ is determined by the path's shaping parameters per node index using Eq. (4).

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

The shape of each alteration set is independent within its respective plane. They take the form of the normal distribution to allow for a distinct, flexible peak deviation and gradual decline to zero deviation at both ends (the fixed flight path points). This form may have difficulty capturing two instances of contact, where a bi-modal shaping equation would likely perform better.

To initialize these parameters, educated guesses are made to approximate a good fit based on the initial contact using the heuristic Eqs. (5), (6), and (7).

$$\vec{\beta}_{\{s,e\}_{i=0}} = \frac{1}{2} \min(\vec{d}_{\{s,e\}}) \tag{5}$$

$$\vec{\sigma}_{\{s,e\}_{i=0}} = \frac{1}{16} j_{max} \tag{6}$$

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

Once $\delta_{j_{\{s,e\}}}$ is determined, the change to the path is computed per node using Eq. (8).

$$\vec{n}'_{j_{\{s,e\}}} = \vec{n}_{\{s,e\}} + \delta_{j_{\{s,e\}}} \vec{u}_{2_{\{s,e\}}} \tag{8}$$

These two altered paths based on $\{\vec{o}_s, \vec{o}_e\}$ must be combined into one $\vec{n}'$ in order to determine how the one path must be changed. The two sets of planes are blended per node according to a weighted average using Eq. (9).

$$\vec{n}'_j = \frac{\|\vec{d}_{j_e}\|}{\|\vec{d}_{j_s}\| + \|\vec{d}_{j_e}\|} \vec{n}'_{j_s} + \left(1 - \frac{\|\vec{d}_{j_e}\|}{\|\vec{d}_{j_s}\| + \|\vec{d}_{j_e}\|}\right) \vec{n}'_{j_e} \tag{9}$$

D. Assessing Cost

E. Updating Parameters with Genetic Algorithm

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. 5.

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. (10).

$$\vec{n}_{i+1} = \vec{n}_i + \vec{\Delta n}_i = \vec{n}_i + \vec{\delta}_i \circ \vec{d}_i = \vec{n}_i + \left(\vec{\delta}_{d_s} \circ \vec{\delta}_{d_e} \circ \vec{\delta}_{\theta_s} \circ \vec{\delta}_{\theta_e} \circ \vec{\delta}_j\right)_i^T \circ \vec{d}_i \tag{10}$$

Each of the Hadamard factors of δ is determined from Eqs. (11), (12), and (13). 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.

$$\vec{\delta}_{d_{\{s,e\}}} = \left[\frac{1}{\sqrt{2\pi\sigma^2_{d_{\{s,e\}}}}} \exp\left(-\beta_{d_{\{s,e\}}} \frac{\left(\|\vec{d}\| - \mu_d\right)^2}{2\sigma^2_{d_{\{s,e\}}}}\right)\right]^{\vec{\rho}} \tag{11}$$

$$\delta_{\theta_{\{s,e\}}} = \left[\frac{1}{\sqrt{2\pi\sigma^2_{\theta_{\{s,e\}}}}} \exp\left(-\beta_{\theta_{\{s,e\}}} \frac{(\theta - \mu_\theta)^2}{2\sigma^2_{\theta_{\{s,e\}}}}\right)\right]^{\vec{\rho}} \tag{12}$$

$$\delta_j = \left[\frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\beta_j \frac{\left(j - \mu_j\right)^2}{2\sigma_j^2}\right)\right]^{\vec{\rho}} \tag{13}$$

Eqs. (11), (12), and (13) 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. 5. 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. (14), which would like to be minimized.

$$C_{deviation_i} = \frac{1}{j_{max} - j_{min}} \sum_{j=j_{min}}^{j_{max}} \frac{\|\vec{n}_{i_j} - \vec{n}_{0_j}\|}{\|\vec{n}_{0_{j_{min}}} - \vec{n}_{0_{j_{max}}}\|} \tag{14}$$

The clustering is minimized by assigning a cost for inter-node distance having a standard deviation above zero, as determined by Eq. (15).

$$C_{internode_i} = \sqrt{\frac{1}{j_{max} - j_{min}} \sum_{j=j_{min}}^{j_{max}-1} \left(\|\vec{n}_{i_j} - \vec{n}_{i_{j+1}}\| - \overline{\|\vec{n}_{i_j} - \vec{n}_{i_{j+1}}\|}\right)^2} \tag{15}$$

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. (16).

$$C_{contact_i} = 1 - \frac{\vec{o}[t]_{i_e} - \vec{o}[t]_{i_s}}{\vec{n}[t]_{i_{j_{max}}} - \vec{n}[t]_{i_{j_{min}}}} \tag{16}$$

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. (17).

$$C_i = \lambda_{deviation} C_{deviation_i} + \lambda_{internode} C_{internode_i} + \lambda_{contact} C_{contact_i} \tag{17}$$

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. (18), a well-known equation in the subject of optimization.

$$\{\beta, \sigma, \mu, \vec{\rho}\}_{i+1} = \{\beta, \sigma, \mu, \vec{\rho}\}_i - \alpha \left(\frac{\partial C_{i+1}}{\partial \{\beta, \sigma, \mu, \vec{\rho}\}_i} \right) \tag{18}$$

4. Tuned Path Alteration

The main loop is continued by updating the node vector by the same Eq. (10), 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 main loop for updating node locations while actively tuning the parameters is summarized in Algorithm (1).

Algorithm 1 Main loop used to update flight path while tuning parameters

INPUT: Object point locations continuously discovered by sim $\vec{o}_i$; initial flight path nodes $\vec{n}_0$; number of iterations I
OUTPUT: Final path nodes $\vec{n}_I$; final tuned shaping parameters $\{\beta, \sigma, \mu, \vec{\rho}\}_I$.

```
Initialize variables to store history in database
Run simulation to create o⃗_0
Initialize dictionary of shaping parameters {β, σ, μ, ρ⃗}_0 per flight leg
for all i in (1, I) do
    Run simulation to update o⃗_i based on n⃗_i
    for all flight in fleet paths do
        for all path leg in flight do
            Extract contacted points of leg in o⃗_i
            if contacted points exist then
                Compute δ⃗_i, d⃗_i from n⃗_i, o⃗_i, {β, σ, μ, ρ⃗}_i
                Compute updated path n⃗_{i+1} from n⃗_i, δ⃗_i, d⃗_i
                Compute cost C_{i+1} from n⃗_{i+1}, o⃗_i, {β, σ, μ, ρ⃗}_i
                Compute gradients of the cost with respect to {β, σ, μ, ρ⃗}_i using PyTorch auto-grad
                Update shaping parameters {β, σ, μ, ρ⃗}_{i+1} from gradients
            end if
            Store n⃗_{i+1} keyed by path leg
            Store {β, σ, μ, ρ⃗}_{i+1} keyed by path leg
        end for
        Store path leg keyed by flight
    end for
    Store fleet paths keyed by i
end for
```

Eq. (19) defines the velocity of a given drone in terms of the two spatial axes and time axis. This connects the fleet requirement of minimum and maximum allowable vehicle velocities to the path shape.

$$\vec{v} = \sqrt{\left(\frac{\partial x}{\partial t}\right)^2 + \left(\frac{\partial y}{\partial t}\right)^2} \tag{19}$$

F. 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 interactive plot used to assess what the algorithm is processing per iteration.

VI. Experiments

A. Straight Perpendicular Encounter

B. Straight Parallel Encounter

VII. Future Work

VIII. Conclusion

Appendix

Acknowledgments

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.