


# A real-time network approach for including obstacles and flight dynamics in UAV route planning

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology  
2016, Vol. 13(3) 291–306  
© The Author(s) 2016  
DOI: 10.1177/1548512916630183  
dms.sagepub.com  


David Myers<sup>1</sup>, Rajan Batta<sup>2</sup> and Mark Karwan<sup>2</sup>

## Abstract

The procedure presented within considers the problem of calculating flight time for unmanned aerial vehicles (UAVs) while incorporating a subset of important flight dynamics characteristics in an operational field that contains obstacles. These flight time calculations are parameter inputs in mission planning and dynamic reassignment problems. The addition of pseudonodes and the addition of penalties into the associated edge weights are the basis for how the network generation procedure includes flight dynamics. The procedure includes a method for handling pop-up targets or obstacles in a dynamic reassignment problem. To guarantee that the optimal path includes flight dynamics, a selective Dijkstra's algorithm computes the shortest path. A complex mission plan consisting of thirty targets and three obstacles is the largest test scenario of nine developed scenarios. Our network generation procedure, along with the shortest path calculations of all 992 node pairs of interest, solves in approximately one second. This procedure allows for the fast computation needed to generate parameters for use in a dynamic domain such as mission planning and dynamic reassignment algorithms.

## Keywords

Unmanned systems, network methods modeling obstacles, flight time calculation

## 1. Introduction

As the use of unmanned aerial vehicles grows in the military domain, important factors have and will continue to emerge. In 2005, the United States Air Force (USAF) released “The U.S. Air Force Remotely Piloted Aircraft and Unmanned Aerial Vehicle Strategic Vision.”<sup>1</sup> The strategic vision provides an outline for what the USAF considered the future vision and recommendations for the use of unmanned aerial vehicles (UAVs). One of the major bullet points in this document is that routing and path planning of these UAVs is an important factor in their future use in operations around the world.

UAV routing problems fall into one of two major categories: mission planning algorithms that are not dynamic and take place prior to the beginning of a mission; and dynamic routing algorithms that handle changes in the original mission plan during the execution of a mission. The literature contains vast amounts of nondynamic mission planning algorithms.<sup>2–7</sup> Dynamic routing algorithms

can have single objectives<sup>8–10</sup> or multiple objectives.<sup>11–14</sup> Similar to mission planning algorithms, they also may contain elements such as time windows for the targets, payload capacity, fuel capacity, the ability to skip certain targets, and precedence in targets.

A detailed problem description is in Section 1.1. A thorough review of the literature is in Section 2. Sections 3 and 4 contain the network development and selective

<sup>1</sup>US Air Force Research Laboratory, Rome, NY, USA

<sup>2</sup>Department of Industrial & Systems Engineering, University at Buffalo (State University of New York), Buffalo, NY, USA

### Corresponding author:

Rajan Batta, Department of Industrial and Systems Engineering, 410 Bell Hall, University at Buffalo (State University of New York), Buffalo, NY 14260, USA.

Email: [batta@buffalo.edu](mailto:batta@buffalo.edu)

Dijkstras algorithm used to find shortest paths in the network. Section 5 shows some implementation results.

### 1.1. Problem description

The purpose of this research is to devise a real-time computational method by which the minimal flight time between a large number of points and the associated travel route of a single UAV can be determined in a plane that consists of bases, targets and polygonal obstacles. These flight times and associated travel routes will be provided as vital parameter values to the mission planning and routing algorithms referenced in the introduction above. Ultimately, these algorithms develop mission plans and travel routes for UAVs. Providing accurate real-time parameter values to these algorithms allows for the incorporation of obstacles and flight dynamics into UAV route planning.

Obstacles in the operational field can either be due to topography or to no-fly zones (NFZ) caused by adversaries

or political policy. These obstacles or NFZs can be as simple as a mountain that extends into the flying elevation of the UAV or a threat that exists on the ground from an adversary. Possible scenarios will include a certain number of given bases, targets, and obstacles on the plane. Figure 1 shows our largest test case, which consists of two bases, thirty targets, and three obstacles. In this image, the house shapes represent bases, the circular shapes represent targets, and the push-pin shapes represent obstacle-defining vertices. We have shaded in the edges of the obstacle in this image.

Routing an UAV using obstacle avoidance within a plane from a single source to a single target is a well-studied problem in the literature, which has many solutions. However, many of these methods use complex algorithms and methods to find these trajectories. These methods are very time consuming and thus are not desirable for our purpose. A network generation procedure was developed and enhanced to obtain a shortest obstacle-free flight time path between all node pairs of interest in the operational field. A node pair of interest is defined as a pair of nodes  $(i, j)$  where both node  $i$  and node  $j$  are bases and/or targets. These flight time and travel route calculations provide more accuracy in flight time calculations and mission planning algorithms.

Let  $G = (V, E)$  be a network, where  $V$  is the set of nodes and  $E$  is the set of edges in the network. Edges have weights attached to them that represent the distance or time between two nodes. The first distinguishing feature of our work is the inclusion of obstacles. The process of discretizing airspace cannot guarantee the optimal path's inclusion in the network. Figure 2 is an example of this. Figure 2(a) shows an example of a source and target vertex with an obstacle en route. Figure 2(b) shows the optimal path between the two vertices. Figure 2(c) shows that using a grid, and adding a node at the centroid of the parts of the grid (if no obstacle exists at the centroid) would produce a path that is not optimal. It is possible to use a very fine grid so that close to optimal points are

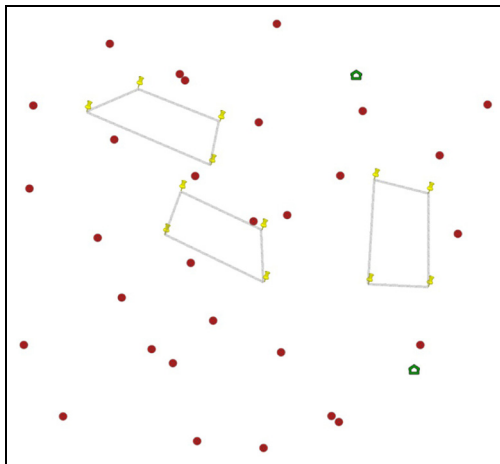


Figure 1. Largest test case.

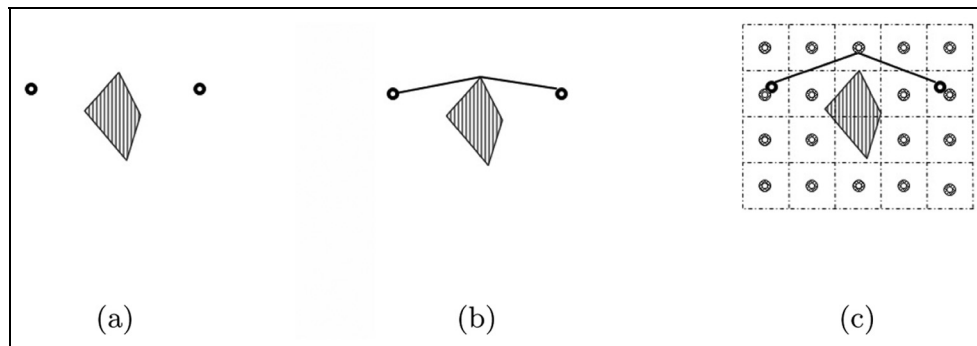


Figure 2. Discretization of airspace. (a) Example scenario. (b) Optimal path. (c) Grid path.

considered. This would significantly slow down the network generation and solution time of any procedure.

The second distinguishing feature of our work is that the assumption of a point-like object traversing throughout the network along straight-line paths is inappropriate. For this reason, we include consideration of flight dynamics in flight time calculations. We use the Dubins' model, which usually requires adding a large number of "pseudonodes" to the network. The third distinguishing feature of our work is that traditional shortest path algorithms (e.g. Dijkstra's algorithm)<sup>15</sup> need to be modified to handle multiple types of nodes. The fourth and most important distinguishing feature of our work is that we are interested in real-time results. The entire procedure from data input until the calculation of all possible origin-destination (OD) pair calculations should be done as fast as possible for use in a dynamic military environment where time is critical for mission success.

## 2. Literature review

We define path planning with obstacle avoidance as the maneuvering of a vehicle from one point to another point in an operational field while avoiding certain areas of that field. This problem exists for any type of vehicle in any appropriate operational environment.

The underlying problem of path planning with obstacle avoidance is well-explored in the operations research literature. The rectilinear solution to the problem of path planning in the presence of barriers has been reported in the literature.<sup>16</sup> This rectilinear solution has been applied to the problem of facility location in the presence of barriers for travel,<sup>17</sup> and Sakar et al. have a commentary on the previous work of Butt and Cavalier, solving the inadequacies in their solution approach.<sup>18,19</sup> The same authors explore the rectilinear solution to the problem of path planning in the presence of convex polygonal congested regions.<sup>20</sup> These all solve the rectilinear problem, which is not the same as the Euclidean shortest path problem in the presence of obstacles. The Euclidean shortest path problem in the presence of rectilinear barriers has been solved in the literature. The barriers in our research are not solely rectilinear, and our shortest path must contain flight dynamics, which is not the focus of any of these works.<sup>21</sup>

Another field that has studied this problem is that of computational geometry. Utilizing the continuous Dijkstra's paradigm to solve the obstacle avoidance problem in a plane has been reported in the literature.<sup>22,23</sup> The continuous Dijkstra's paradigm has a fairly slow solution time in comparison to what is needed for our applications, and in addition only a single source and single vertex were considered. Work has also been reported in the area of using the visibility graph for the solution of the underlying

shortest path problem.<sup>24,25</sup> The creation of the visibility graph is complex in nature and is also only concerned with a single source and vertex pair. The shortest path problem from the discrete geodesic problem point of view has also been explored,<sup>26</sup> along with an optimal-time algorithm for the shortest path computation between two points in the presence of polygonal obstacles.<sup>27</sup> An extensive review of the geometric shortest path problem has been published, however these methods are only capable of including a single source and destination pair.<sup>28</sup>

The problem of path planning with obstacle avoidance has been studied and explored from several points of view that do not fall into one of the above stated fields of mathematics. A method has been developed for three-dimensional risk minimization of aircraft detection,<sup>29</sup> and exploring a three-dimensional path-planning problem with circular danger regions has been presented in the literature.<sup>30</sup> Our problem is different, as we consider a single UAV at a constant altitude and do not have a risk consideration. A method has been reported by which target assignment of multiple UAVs in an uncertain environment has been analyzed.<sup>31</sup> Our work is to provide flight time data for these target assignment problems. An algorithm has been reported for finding a feasible path using nonholonomic motion planning in the presence of obstacles, which is focused on ground-based vehicles such as tractor trailers.<sup>32</sup> In the literature this problem has been solved using a constrained shortest path model that discretizes the relevant airspace into a grid, which as stated earlier does not guarantee that the optimal path is included in the network.<sup>33</sup> Lewis and Ross write about a pseudo spectral method for real-time motion planning and obstacle avoidance, however their method is computationally very slow.<sup>34</sup> The methods and algorithms for finding obstacle-free paths for mobile robots and UAVs have been reviewed in the literature, however this division of the work-area method would be very large for airborne operations and only works well with smaller ground-based work areas.<sup>35</sup>

There has been much work in the use of a network generation procedure for path planning for not only UAVs but also mobile robots. Previous work in robot motion planning divides the plane into cells for the network generation.<sup>36</sup> Chen et al. combine what they consider the advantages of the grid-based methodology and the quad tree method.<sup>37</sup> Casas et al. present a method whose objective is to create an automated generalized network generation procedure within a geographical information system (GIS) environment.<sup>38</sup> All of the above methods use a discretization method for the relevant airspace or operational field. This does not guarantee that the optimal shortest path is included in the network. The mathematics behind using a discrete approximation for the continuous shortest-path problem in the context of minimum-risk planning for

UAVs has been reported in the literature.<sup>39</sup> Despite the discrete approximations providing good solutions, their approximation approach has large solution times: over ten seconds for a single vertex pair. A model for the mission planning of a set of cruise missiles to avoid high population density areas as well as important areas of a city has been reported in the literature.<sup>40</sup> This heuristic replaces the flight dynamics generator. We utilize a flight dynamics generator in our network generation procedure and thus do not have the need for a heuristic-based approach. The original work on turn penalties in networks is given by Caldwell.<sup>41</sup> His work formulates a dual graph where nodes represent edges traveled. This is a concept that we leverage in our creation of pseudonodes.

### 3. Network development

We use a network generation procedure for the purpose of calculating the flight time of an UAV in the presence of obstacles while incorporating flight dynamics. We implement what we refer to as pseudonodes for including flight dynamics.

Our model makes a few reasonable assumptions in regards to the network development procedure. The first assumption is that the network generation process must know the data for all of the vertices a priori. This means that the location of all of the obstacles should be known or approximated and defined. Any loss in fidelity introduced in modeling because of this assumption is minimal because of today's imagery and reconnaissance capabilities. While this assumption is not necessarily always true, our model has a method for handling this scenario. Adversarial obstacles can appear in the operational field during flight. The pop-up target framework to be described later affords a natural means of also allowing new obstacles to pop-up during flight, allowing for new nodes and pseudo-nodes to be inserted into the network so that dynamic reassignment is possible.

Another required assumption is that all bases, targets, and obstacles must remain static throughout the process. This model does not handle movement of vertices and obstacles. Once again, if the obstacles are purely topological then they will obviously remain static. If adversaries cause the obstacles, they too often remain static during a given period.

Our network generation procedure consists of the following steps:

1. Model input
2. Vertex addition
3. Circle estimation
4. Obstacle defining edge addition
5. Other edge addition

**3.0.1. Model input.** Input for the model is a file where each row contains a vertex name, an obstacle number, a vertex number, an obstacle shape, latitude, longitude, and a radius. The vertex name must be a unique string of characters. The obstacle number is an integer value for each obstacle in the operational field. The vertex number is unique for each obstacle number. The obstacle shape value defines whether the point represents a circular obstacle or not. The latitude and longitude values correspond to the actual latitude and longitude values of the vertex on the surface of the Earth. The radius is a value in kilometers of the radius of a circular obstacle if the vertex represents the center of a circular obstacle.

We utilize a flat Earth approximation for the vertices from latitude and longitude down to Cartesian coordinates for mathematical purposes. We use this method because the use of multiple projection systems (Gauss-Kruger and Transverse Mercator) and a Java-based GIS method (GeoTools Envelope Method) at locations within the United States provided very inconsistent networks. These were determined to be inconsistent because the use of these projection methods created edges through the center of no fly zones (or edges that were far away from no fly zones were not in the network). The consistent results of the flat Earth approximation lead to its use.

**3.0.2. Vertex addition.** The vertices in the network represent a base, a target, or an obstacle. The latitude and longitude data for the input represent these vertices.

**3.0.3. Circle estimation.** If a vertex represents a circular obstacle, then the circular obstacle is estimated by using an inscription method within an  $n$ -sided regular polygon. For this to occur, we must first translate from  $r_c$  (apothem or radius of the inscribed circular obstacle) to  $r_o$  (distance from center of circular obstacle to the obstacle-defining vertices to be found). Let  $s$  represent the length of one of the  $n$  sides in the polygon. Using the law of sines and solving  $r_o$  in terms of  $s$  yields Equation (1). The use of the Pythagorean theorem results in Equation (2). Substituting using Equation (1) and solving for  $r_o$  in terms of  $r_c$  gives Equation (3):

$$r_o = \frac{s}{2 \times \sin(\frac{\pi}{n})} \quad (1)$$

$$r_o^2 = \frac{s^2}{2} + r_c^2 \quad (2)$$

$$r_o = \frac{r_c}{\cos(\frac{\pi}{n})} \quad (3)$$

Once a value for  $r_o$  is found, and given that  $lat_1$  and  $long_1$  are the latitude and longitude of the starting point respectively, and  $d$  is the distance to be traveled at heading  $\theta$ , the

latitude and longitude of the destination point would be given by Equations (4) and (5), respectively.

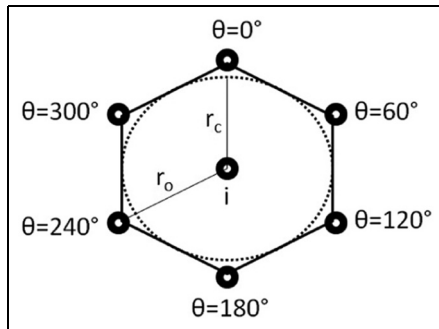
$$\begin{aligned} lat_{\text{destination}} = & a \sin \left( \sin(lat_1) * \cos\left(\frac{d}{R}\right) \right. \\ & \left. + \cos(lat_1) * \sin\left(\frac{d}{R}\right) * \cos(\theta) \right) \end{aligned} \quad (4)$$

$$\begin{aligned} long_{\text{destination}} = & long_1 + a \tan 2 \\ & [\sin(\theta) * \sin\left(\frac{d}{R}\right) * \cos(lat_1), \\ & \cos\left(\frac{d}{R}\right) - \sin(lat_1) * \sin(lat_2)] \end{aligned} \quad (5)$$

Using these formulas at evenly spaced headings around the circular obstacle allows us to estimate the circular obstacle with a regular  $n$ -sided polygon with vertices spaced at  $\theta = \frac{m}{360}$  for  $m = 0, 1, 2, \dots, n - 1$ . We determined that  $n = 6$  was a good representation of the circular obstacle without slowing down the network generation procedure. Figure 3 shows an example of taking a single point and estimating a circular obstacle around that single point using a hexagonal approximation ( $n = 6$ ). These vertices in Figure 3 are entered into the network as obstacle-defining vertices.

**3.0.4. Obstacle defining edge addition.** The goal of this procedure is to add the edges to the network that define the exterior of obstacles that exist in the operational field. The input data lists the obstacle-defining vertices in order first, and if the iterator finds that a vertex is next in line and belongs on the same obstacle, the procedure adds a directed edge between the first and second vertex in the list. When the iterator finds a dummy vertex (a vertex with a zero value for vertex number on the obstacle), the procedure adds a directed edge from the previous vertex to the first vertex on the obstacle.

The edge weights for these obstacle-defining edges are equal to the Haversine distance between the two end points



**Figure 3.** Circle estimation using the hexagonal approximation.

of the edge. This Haversine distance is the great circle distance across the Earth's surface.<sup>42</sup> The following shows the formula for the Haversine distance between two points on the Earth's surface:

$$\nabla lat = lat_2 - lat_1 \quad (6)$$

$$\nabla long = long_2 - long_1 \quad (7)$$

$$\begin{aligned} a = & \sin^2((\nabla lat)/2) + \cos(lat_1) \\ & \times \cos(lat_2) \times \sin^2((\nabla long)/2) \end{aligned} \quad (8)$$

$$c = 2 \times a \tan 2(\sqrt{a}, \sqrt{1-a}) \quad (9)$$

$$d = R \times c \quad (10)$$

**3.0.5. Open space edge addition.** The next step in the network generation procedure is the addition of edges that are not obstacle-defining edges. In the network, these edges are equivalent to the obstacle-defining edges. The addition procedure considers all possible edges for addition into the network. In order for a possible edge to gain addition into the network, it must not intersect with either circular obstacles or with any of the edges of the obstacles defined solely by points anywhere in the operational field. There are three cases that a possible edge will fall into when considering intersection with a circular obstacle. Figure 4 describes these.<sup>43</sup>

The following method determines in which case the possible edge fits:

$$d_x = x_2 - x_1 \quad (11)$$

$$d_y = y_2 - y_1 \quad (12)$$

$$d_r = \sqrt{d_x^2 + d_y^2} \quad (13)$$

$$D = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1 \quad (14)$$

The points of intersection are as follows:

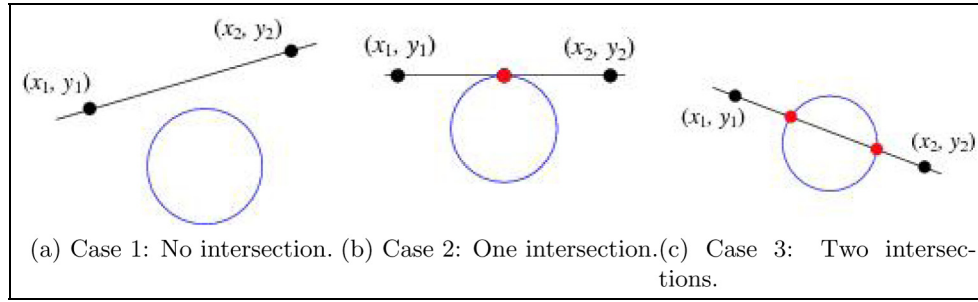
$$x = \frac{D d_y \pm \text{sgn} * (d_y) d_x \sqrt{r^2 d_r^2 - D^2}}{d_r^2} \quad (15)$$

$$y = \frac{-D d_x \pm |d_y| \sqrt{r^2 d_r^2 - D^2}}{d_r^2} \quad (16)$$

$$\text{sgn} * (x) = \begin{cases} -1 & \text{for } x < 0 \\ 1 & \text{otherwise} \end{cases} \quad (17)$$

$$\Delta = r^2 d_r^2 - D^2 \quad (18)$$

Table 1 shows the summary of the existence of intersection point(s).



**Figure 4.** Circle-line intersection cases.

**Table 1.** Summary of existence of intersection point(s) of circle and line.

$\Delta$	Incidence
$\Delta < 0$	no intersection
$\Delta = 0$	tangent intersection
$\Delta > 0$	intersection

If the line created by the vertex pair of interest has a  $\Delta$  value that puts it in case 1 or case 2, then that edge passes the circle intersection test for that circular obstacle. If it falls into case 3 then there is one more step to complete in order to determine if the edge passes the circle intersection test or not. In these equations,  $r$  is the radius of the circle. This last step is to determine if the intersection points occur within the line segment defined by each vertex of interest on the line. Equations (15) and (16) produce the intersection points. If these intersection points fall between the vertices of interest along the line segment, then the possible edge fails the circle intersection check. If the intersection points fall outside of this line segment then the possible edge will pass the circle intersection check for that circular obstacle. The method for checking the intersection with the noncircular obstacle-defining edges uses the geometric formulas for line segment intersection.<sup>44</sup> Figure 5 shows two example line segments for an example intersection check.

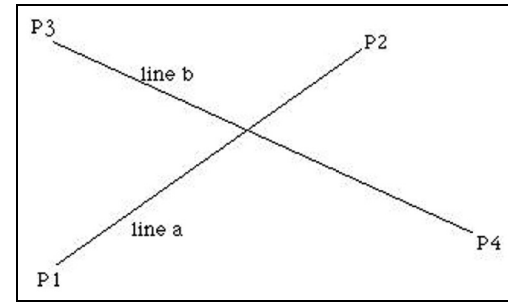
The equations of the line segments in Figure 5 are

$$P_a = P_1 + u_a(P_2 - P_1) \quad (19)$$

$$P_b = P_3 + u_b(P_4 - P_3) \quad (20)$$

Solving these equations for the possible intersection point of these line segments ( $P_a = P_b$ ) results in the following equations, where  $u_a$  and  $u_b$  are unknown values:

$$x_1 + u_a(x_2 - x_1) = x_3 + u_b(x_4 - x_3) \quad (21)$$



**Figure 5.** Line segment intersection.

$$y_1 + u_a(y_2 - y_1) = y_3 + u_b(y_4 - y_3) \quad (22)$$

Solving these equations for  $u_a$  and  $u_b$  results in the following equations:

$$u_a = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (23)$$

$$u_b = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (24)$$

By substituting either of these values into the equation of the lines located above, the following equation provides the intersection point of the line segments:

$$x = x_1 + u_a(x_2 - x_1) \quad (25)$$

$$y = y_1 + u_a(y_2 - y_1) \quad (26)$$

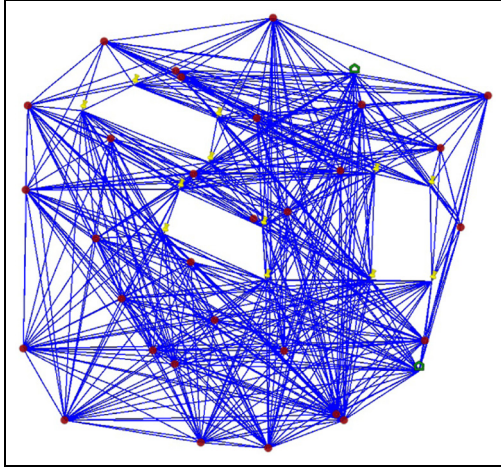
The results of these equations put the line segments into one of many cases, only three of which are interesting. Table 2 describes the three cases.

If the line segments defined by the possible edge pass the intersection test for all obstacle edges that define noncircular obstacles in the operational field then it has passed the intersection check.

When a possible edge ( $i, j$ ) passes the circular obstacle intersection check and the noncircular obstacle intersection check, that edge is added to the network. The edge's

**Table 2.** Summary of existence of intersection point(s) of two line segments.

$u_a$	$u_b$	Intersection
denom. = 0	denom. = 0	none
denom. and num. = 0	denom. and num. = 0	coincident line segments
$0 < u_a < 1$	$0 < u_b < 1$	between endpoints

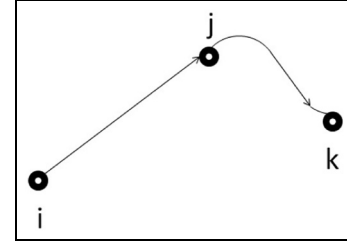
**Figure 6.** Test case network.

weight is set to be the Haversine distance between the two end points  $i$  and  $j$  for the edge.

At this point in the network generation procedure, the network contains half of the edges that a directed network without flight dynamics should have. We then take every edge that exists in the graph and create the directed edge that is opposite to that. That is for each edge  $(i, j)$  we create an edge  $(j, i)$  and set the edge weight to the weight of  $(i, j)$ . This complete network  $G_{NFD}$  is the network on which any nonflight dynamics tests or analyses were completed. The network for our largest test case can be seen in Figure 6.

### 3.1. Flight dynamics

Flight dynamics is the science of air vehicle orientation and control in three dimensions. The next step in the network generation procedure is to include the consideration of flight dynamics. The reason for the requirement of the inclusion of flight dynamics in the network is that the actual flight time for an UAV to traverse a specific edge in the network might not just be the straight-line distance along that edge. Figure 7 shows an example path of an UAV along triplet  $i, j, k$ . As can be seen from the figure, the flight time along the edge  $(j, k)$  will not be equal to the

**Figure 7.** Flight path along a vertex triplet.

straight-line distance along edge  $(j, k)$  if the UAV is coming from node  $i$ .

One model assumption is the treatment of an UAV as a point-like object. This does not refer to the point-like object traveling along straight-line paths, but rather a point-like object traveling along paths that consider flight dynamics. The operational field that we are examining can easily be several to hundreds of square miles. This large operational field means an UAV can be reasonably approximated as a point-like object. Other assumptions about the UAV itself are that it travels at a constant velocity and a constant altitude during flight. UAV control systems typically hold an UAV at a constant velocity and can even attempt to maintain constant elevation during a mission, making this assumption reasonable.

Another assumption of this model is that when considering a pair of interest in the network, the UAV will always be on the rhumb line facing towards the first vertex in the path. The rhumb line between two points is the straight line that connects the two points. An example of this assumption is if we are considering a vertex triplet  $i, j, k$  starting a shortest path out of node  $i$ . The UAV is considered to start traveling from  $i$  along the rhumb line between  $i$  and  $j$ . In the same shortest path consideration, the same node  $i$  is considered in another starting triplet of  $i, p, k$ , then the UAV is considered to start along the rhumb line from  $i$  to  $p$ .

To include flight dynamics into the calculation of flight time, we define a pseudonode which is located at the exact geographical location as one of the original nodes, yet allows for some other functionality in the model. A pseudonode not only contains the geographical data that the original node contains, but also contains the node that the UAV would previously be located at, as well as the node that the UAV would be heading towards after it goes through the pseudonode. The node that the UAV would previously be flying from into the pseudonode is the previous node, and the node that the UAV is heading towards after the pseudonode is the next node. For example, if an UAV is located at node  $i$  and is going to fly from node  $i$  through node  $j$  to node  $k$ , the pseudonode would be located at node  $j$  and would be called  $j_{ik}$ . This node set of  $i, j, k$  is



a node triplet. This addition of pseudonodes allows for the inclusion of flight dynamics into the network.

There are a couple of conditions that must be met in order for the model to determine if the pseudonode  $j_{ik}$  should be added to the network. These conditions not only determine if a pseudonode should be located at  $j_{ik}$ , but also if the pseudonode could possibly be used. The first condition is that in  $G_{NFD}$ , an edge must exist between node  $i$  and node  $j$ , as well as an edge between node  $j$  and node  $k$ . This is required because in order for the UAV to fly from  $i$  through  $j$  on its way to  $k$ , there must be edges that exist for this edge sequence in  $G_{NFD}$ .

The next condition exists to limit the number of pseudonodes that are generated. This condition is that if an edge exists between node  $i$  and node  $k$ , then the model will not add the pseudonode  $j_{ik}$  into the network. This condition is implemented because if an edge already exists between target  $i$  and target  $k$  (no obstacle is in the way), the shortest path between these two targets will just be that edge and there would be no need to fly from  $i$  through  $j$  to  $k$  as the inclusion of the pseudonode would entail. Another condition that is added is the condition that node  $j$  must be an obstacle-defining node. Failure to meet this condition will result in many unnecessary pseudonodes being created between node triplets that only contain targets.

After the previously mentioned conditions are determined to be met and a pseudonode  $j_{ik}$  is created, the next step is to connect these pseudonodes to the rest of the network. A directed edge is added from node  $i$  to node  $j_{ik}$ . The weight of this edge is the Haversine distance. Next, a directed edge from node  $j_{ik}$  to node  $k$  is added and its weight is set equal to a Dubins' factor for distance, which comes from approximating a minimum distance path for a Dubins' flight model.<sup>45</sup> Dubins' original work led to work by Schkel and Lumelsky,<sup>46,47</sup> which limited the number of calculations needed to determine the minimum distance path for a Dubins' problem, and resulted in an efficient implementation of the Dubins' curve path model. The accuracy of the Dubins' curve path model has been demonstrated in the literature;<sup>48</sup> its accuracy and ease of implementation is why we choose the Dubins' curve path model for approximating the actual flight time between point  $j$  and point  $k$  originally coming from point  $i$ .

The following six equations are used to define the Dubins' curve path model required calculations.<sup>45</sup> Table 3 defines the nomenclature in the equations:

$$x_{ij} = \frac{Long_j - Long_i}{\text{feetToDegrees} \times \frac{1}{\cos(Lat_i) * (\frac{180}{\pi})}} \quad (27)$$

$$y_{ij} = \frac{Long_j - Long_i}{\text{feetToDegrees}} \quad (28)$$

**Table 3.** Dubins' nomenclature.

$Lat_a$	latitude centered at waypoint $a$ (decimal degrees)
$Long_a$	longitude centered at waypoint $a$ (decimal degrees)
$psi0$	assumed heading at the initial waypoint (degrees)
$psiF$	desired heading at the final waypoint (degrees)
$x_{ab}$	horizontal Cartesian coordinate distance between points $a$ and $b$ (feet)
$y_{ab}$	vertical Cartesian coordinate distance between points $a$ and $b$ (feet)
$\text{feetToDegrees}$	$2.742980561538962 \times 10^{-6}$ (constant)

$$psi0 = \text{mod}(a \tan 2(x_{ij}, y_{ij}), 2 * \pi) \times (\frac{180}{\pi}) \quad (29)$$

$$x_{jk} = \frac{Long_k - Long_j}{\text{feetToDegrees} \times \frac{1}{\cos(Lat_k) * (\frac{180}{\pi})}} \quad (30)$$

$$y_{jk} = \frac{Long_k - Long_j}{\text{feetToDegrees}} \quad (31)$$

$$psiF = \text{mod}(a \tan 2(y_{jk}, x_{jk}), 2 * \pi) \times (\frac{180}{\pi}) \quad (32)$$

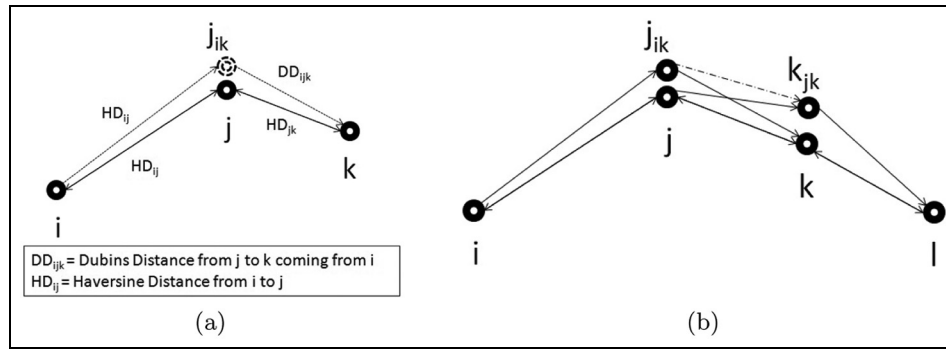
Since the Dubins' curve path model uses a flat Earth approximation, we need to scale the approximated flight time using the Haversine distance. This is achieved by Equation (33), where  $DD_{ijk}$  is the new Dubins' factor that is used for the edge weights along the edge from the pseudonode  $j_{ik}$  to vertex  $k$ .  $HD_{jk}$  is the Haversine distance from vertex  $j$  to vertex  $k$ .  $DCP_{ijk}$  is the Dubins' curve path model's flight distance flying from  $j$  to  $k$  (originally coming from  $i$ ), and  $SLD_{jk}$  is the straight-line distance that the Dubins' curve path model calculates from vertex  $j$  to vertex  $k$ :

$$DD_{ijk} = HD_{jk} \times \frac{DCP_{ijk}}{SLD_{jk}} \quad (33)$$

Figure 8(a) shows an example of the addition of a pseudonode into a triplet  $(i, j, k)$ . In Figure 8, all nodes labeled  $j_{ik}$  represent the flight from node  $j$  coming from node  $i$  and heading to node  $k$  next. This figure shows the addition of a vertex, two directed edges, and the setting of these edge weights on the triplet. The vertices and edges that are solid would have been added during the process of developing  $G_{NFD}$ . The process of including flight dynamics adds the dashed vertices and edges in the figure.

The next step is to connect all of the correct pseudonodes. There is only one necessary condition for connecting two pseudonodes. If the next node of one pseudonode corresponds to the node that the second pseudonode relates





**Figure 8.** Psuedonode addition and connection. (a) Addition of a pseudonode. (b) Connection of pseudonodes.

**Table 4.** Computation times for nine sample scenarios.

Obstacles	1	2	3	1	2	3	1	2	3
Bases	2	2	2	2	2	2	2	2	2
Targets	10	10	10	20	20	20	30	30	30
Vertices	108	296	488	388	838	1496	590	1270	2062
Edges	470	1244	2352	1552	3658	6696	2420	5396	9276
NGT (sec)	0.0698	0.2093	0.2762	0.1756	0.3181	0.4669	0.2151	0.5436	0.5935
SPF	132	132	132	462	462	462	992	992	992
SPT (sec)	0.1199	0.2814	0.3599	0.3267	0.5278	0.6661	0.4329	0.8157	0.9909

to in  $G_{NFD}$ , then the two pseudonodes should be connected. The edge weight of this added edge is set to be the same previously shown Dubins' factor distance. Figure 8(b) shows an example of this connection of pseudonodes. The solid black vertices and edges exist because of the earlier portion of the procedure. The dashed edge indicates the added edge during this part of the network generation procedure for including flight dynamics.

After the correct pseudonodes are created, the correct edges are added, and the edge weights are set properly,  $G_{FD}$  is now completed.  $G_{FD}$  will represent the network on which any flight dynamics tests or analyses were completed. This is our complete network.

#### 4. Selective Dijkstra's algorithm

The last step in calculating the flight time between desired nodes on the network is to calculate the shortest paths between each pair. While considering all pairs of targets and bases, each pair will fall into one of two cases. For a pair to fall into case 1 there must be an edge that directly connects the origin and destination. The shortest path in this case is obviously that single link between the origin and destination of the pair. Case 2 is the case where no edge in the network exists that connects the origin and destination of the pair. This case requires the use of a shortest path algorithm. In order to ensure that the resulting path incorporates flight dynamics, the UAV must fly from the

desired origin node in the pair through any number of connected pseudonodes in the network and ultimately end up at the desired destination node. After comparing multiple methods, it was determined that the use of Dijkstra's algorithm would be the best way to determine these shortest paths.

This problem could be considered an all-pairs problem, in  $G_{NFD}$ . However, in  $G_{FD}$  it is not an all-pairs problem. Using the information from Table 4, one can see that in the largest test scenario there are 31 nodes of interest for each of 32 source nodes considered. This gives us 992 paths to be determined since the entire network has 2061 potential vertices of interest for each source node; our problem corresponds better to a one-to-one approach to finding the shortest paths. Another consideration is that common algorithms such as the Floyd–Warshall algorithm, which would consider over 2000 vertices, do not store the path. Floyd–Warshall can store the path, but its consideration as an all-pairs shortest path would slow down the procedure because of the pseudonodes. Since we are finding not only the flight time, but also the corresponding flight path, it was determined that the use of Dijkstra's algorithm would be the best way to determine these shortest paths.

Traditional Dijkstra's algorithm (TDA) is a single-source graph-search algorithm that finds the geodesic or shortest path between two vertices within the graph. The algorithm consists of a label-setting operation for the

vertices. This algorithm is popular in routing research, as a generated network exists over roads or other methods of travel. In this research, the generated network exists over the operational airspace of the UAV.

Let  $\text{distance}[X]$  be the distance from the source vertex  $S$  to the vertex  $X$  along its shortest path between  $V$  and  $X$ . Let  $\text{parent}[X]$  be the parent vertex of vertex  $X$  along its shortest path between  $V$  and  $X$ . The steps below represent the steps of the TDA.

---

```

1: for  $V \in G$  do
2:    $\text{distance}[V] = \infty$ 
3:    $\text{parent}[V] = \text{undefined}$ 
4: end for
5:  $\text{distance}[S] = 0$ 
6:  $Q = \text{set of all nodes in } G$ 
7: while  $Q$  is not empty do
8:    $u = \text{vertex in } Q \text{ with smallest distance}[]$ 
9:   if  $\text{distance}[u] = \infty$  then
10:    break
11:   end if
12:   remove  $u$  from  $Q$ 
13:   for (neighbor  $v$  of  $u$ )  $\in Q$  do
14:      $\text{tempDistance} = \text{distance}[u] + \text{edgeWeight}[u,v]$ 
15:     if  $\text{tempDistance} < \text{distance}[v]$  then
16:        $\text{distance}[v] = \text{tempValue}$ 
17:        $\text{parent}[v] = u$ 
18:     end if
19:   end for
20: end while

```

---

At line 12, if the  $u$  to be removed from  $Q$  is the target, then the procedure can be terminated and the shortest path is found. If  $u$  is the target, then let  $S$  be an empty list. In order to obtain the list of edges that develops the shortest path:

---

```

1: while  $\text{parent}[u]$  is defined do
2:   insert  $u$  at beginning of  $S$ 
3: end while

```

---

After this procedure,  $S$  will be the list of nodes in the shortest path, and a list of edges  $E_{\text{path}}$  can easily be determined.  $E_{\text{path}}$  must be traversed in order to obtain the shortest path along the network from source  $s$  to target  $t$ .

One modification for a “selective” Dijkstra’s algorithm (SDA) is the modification of  $Q$ . Instead of initializing  $Q$  to be the set of all vertices in the network, our SDA needed to determine which vertices could possibly be in a shortest path that includes flight dynamics. If a vertex is either the source vertex  $s$  or the target vertex  $t$ , then it is clear that the shortest path must contain these vertices in order to have the actual shortest path between them. Therefore,  $Q$  is initialized to  $s$  and  $t$ . If a shortest path between  $s$  and  $t$  is

to contain flight dynamics, then any vertex that would be reached along the shortest path must be a pseudonode. Therefore, all pseudonodes in the network must be initialized into  $Q$ . With  $Q$  initialized properly, the rest of the SDA is the same as the TDA, yet includes the selectivity that is needed in order to provide paths that include flight dynamics between all of the vertex pairs in the scenario.

One assumption that our model makes about these computed paths using flight dynamics is that they are all feasible; requiring that the obstacles remain a reasonable distance away from one another satisfies this condition. With this requirement, all of the generated paths should be close to (if not perfectly) feasible. This is a result of the fact that obstacles are often defined by vague edges. For example, a stationary surface-to-air missile (SAM) site’s range might be set at a certain distance, however this range could have a tolerance and as long as the given range is at the outer edge of that tolerance, an UAV can enter the obstacle edge by a slight margin. This condition is only caused when considering flight dynamics because the equations only calculate flight time, and do not capture the complete location of the vehicle during flight. As long as the provided information for the model has a slight exaggeration of the obstacles, any path that may marginally enter the obstacle would still be feasible. In our experience, not only is it difficult to force a path to enter an obstacle, but also the obstacles must be located very close to one another for this to occur.

Table 5 describes some of the information about the networks generated for our test cases. In this table, V stands for vertices, E stands for edges, OE stands for obstacle edges, FD stands for flight dynamics network, and NFD stands for the nonflight dynamics network. This table shows that when the network goes from  $G_{NFD}$  to  $G_{FD}$ , the density of the network drops drastically. This is because for every added pseudonode in the network, there exists two required edges, and thus the density would decrease in the network.

## 5. Implementation

The calculation of these flight times using the method described in the previous sections can benefit the routing of vehicles for mission planning and the dynamic reassignment of UAVs. This section will provide general results from our research and specific implementation descriptions for these two applications.

### 5.1. Results

All test cases, computations, and results come from running a Java implementation on a Dell Latitude D830 Laptop with an Intel Core 2 Duo T9300 CPU at 2.5GHz with 4GB RAM. The first analysis performed on the model was to determine if generating one full network

**Table 5.** Network density comparisons.

	Avg								
<b>Obstacles</b>	1	1	1	2	2	2	3	3	3
<b>Targets</b>	10	20	30	10	20	30	10	20	30
<b>Bases</b>	2	2	2	2	2	2	2	2	2
<b>NFD V</b>	16	26	36	20	30	40	24	34	44
<b>NFD E</b>	134	434	856	244	518	932	310	618	968
<b>NFD V to OE</b>	36	68	93	100	147	199	146	209	266
<b>NFD E per V</b>	8.4	16.7	12.8	12.2	17.3	23.3	12.9	18.2	22.4
<b>NFD OE per V</b>	2.3	2.6	2.6	5.0	4.9	5.0	6.1	6.1	6.0
<b>FD V</b>	108	296	488	388	838	1496	590	1270	2062
<b>FD E</b>	470	1244	2352	1552	3658	6696	2420	5396	9276
<b>FD V to PE</b>	244	540	1044	940	2332	4308	1544	3542	6272
<b>FD E per V</b>	4.4	4.2	4.8	4.0	4.4	4.5	4.1	4.2	4.5
<b>FD OE per V</b>	2.3	1.8	2.1	2.4	2.8	2.9	2.6	2.8	3.0
									2.5

**Table 6.** Network per pair versus network per scenario.

Pair number	Run-time (sec)
1	0.1353
2	0.1183
3	0.1143
4	0.1330
5	0.1193
Total	0.6203
Single large network	0.5487

was faster than generating a separate network for each vertex pair of interest. The summation of the network generation time and shortest path calculation time using TDA on the nonflight dynamics networks provided the total time for each pair. Then, summation of five chosen pairs of interest gave us a comparison point. These times shown are the average of five runs for each vertex pair of interest. Table 6 shows the results of this analysis.

These results show that even without flight dynamics included, the generation of a single large network results in faster computation time than generating a separate network for each vertex pair of interest. This result confirms our belief that this was the fastest method for computation.

The next important result is the computation time of the nine tested sample scenarios. Table 4 shows the sample scenarios, their network size, their associated network generation time with flight dynamics included, and the time taken to calculate all shortest paths of interest. In this table, NGT stands for the network generation time of the scenario, SPF stands for the shortest paths found while computing the scenario, and SPT stands for the shortest path time for computing the shortest paths in the scenario. This time also represents the total run time for that scenario in the model. These times are the average of ten runs of each scenario.

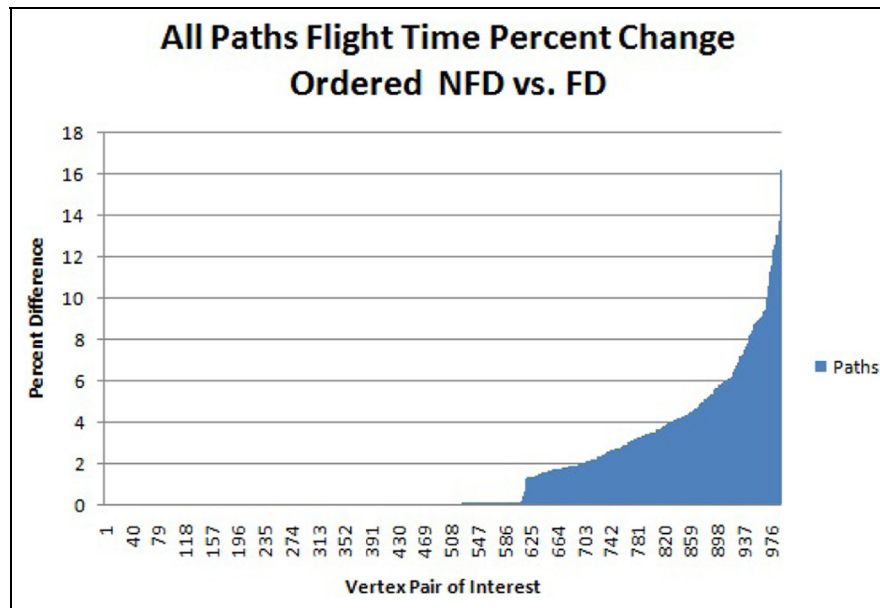
The fast computation time of the largest scenario shows that this model allows for the substitution of this method for the straight-line method in flight planning algorithms without slowing down the algorithm. We judged that using our method (including flight dynamics in its calculations) would not hinder the dynamic domain of this model.

The next analysis that was completed was based on comparing the flight times that were calculated using TDA on the  $G_{NFD}$  and the flight times that were calculated using the SDA on  $G_{FD}$ . This percent difference is between the flight time of the shortest path found in the  $G_{NFD}$  and the flight time of the shortest path found in  $G_{FD}$ . We calculated the percent difference between the flight times for every vertex pair of interest in these two networks. Equation (34) defines percent difference:

$$\text{Percent Difference} = \frac{|(FD_{\text{time}} - NFD_{\text{time}})|}{FD_{\text{time}}} \times 100 \quad (34)$$

We chose to complete this analysis on the largest scenario on which we tested the model. This scenario (consisting of thirty targets, two bases, and three obstacles) was the selected scenario. Figure 9 shows the percent difference of all the flight times for all 992 calculated paths in this scenario. This figure has them ordered from smallest percent change to largest percent change.

An important step is determining if any flight paths will change as the result of including flight dynamics. Table 7 shows the breakdown of the 992 paths that existed in the largest sample scenario. We define a long path as a path that has more than a single edge contained in it. We define a changed path as a path that contained different vertices when computed using  $G_{FD}$  than when it was computed using  $G_{NFD}$ . The results of this table confirm that this model will change paths from  $G_{NFD}$  to  $G_{FD}$ . Figure 10 shows an example of a path that was changed. The solid line in Figure 10(a) represents the nonflight dynamics



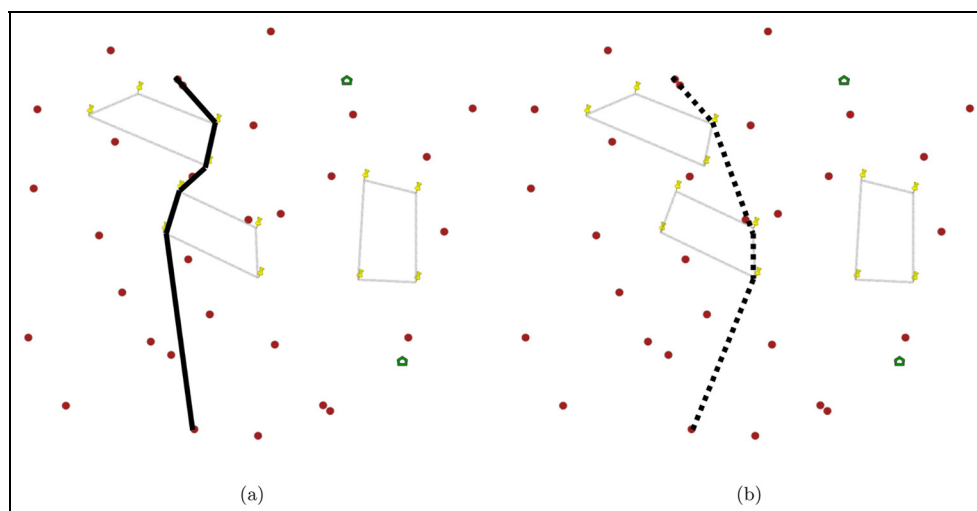
**Figure 9.** Flight time percent change for all paths ordered.

**Table 7.** Paths in largest sample scenario.

<b>Total paths</b>	992
<b>Single edge paths</b>	470
<b>Long paths</b>	522
<b>Changed paths</b>	37
<b>Percent paths changed</b>	7.0881

shortest path between two targets in the sample scenario. The dashed line in Figure 10(b) represents the flight dynamics shortest path between the same two targets.

Table 8 provides more insight into how these changed paths correlate with the total paths' percent change in flight time. Column one is the range of percent difference being considered. Column two is the number of total paths that



**Figure 10.** Example of changed path. (a)  $G_{NFD}$  target 8 to target 16. (b)  $G_{FD}$  target 8 to target 16.

**Table 8.** Comparison of total paths to changed paths with percent difference.

% Diff. range	Total paths	% of total paths	Changed paths	% of changed paths
[0]	522	52.6210	0	0
(0, 1]	95	9.5766	7	18.9189
(1, 2]	85	8.5685	6	16.2162
(2, 5]	176	17.7419	17	45.9459
(5, 10]	92	9.2742	6	16.2162
(10, $\infty$ ]	22	2.2177	1	2.7027
Total	992		37	

fell into each range. Column three is the associated percentage of total paths that fall into each range. Column four is the number of changed paths in each range when considering flight dynamics. Column five is the associated percentage of changed paths from the total number of paths that changed when considering flight dynamics.

With the analyses described above, it is reasonable to assume that our developed model is implementable and useful for mission planning and dynamic reassignment. It would be difficult to estimate the path changes (as a percentage increase on a path) or which paths would change in the scenario.

### 5.2. Mission planning

Mission planning is often treated as an instance of the traditional operations research problem of the traveling salesman problem (TSP). Since this is an NP-hard problem, heuristics often generate a feasible solution for a moderate-to large-sized problem in reasonable time. An example of such a method used to determine a route selection for complex missions is given in the literature.<sup>3</sup> Column generation is used as the basis for developing these mission plans. Implementation of our model into algorithms such as this one would benefit the mission plans they produce. Most mission planning algorithms that exist in the literature utilize flat Earth approximated distances, straight-line flight paths, and consist of an obstacle-free operational field for the UAV. This allows for the removal of these three assumptions from the mission planning algorithms with the implementation of our model.

The most important factor in terms of implementing this model is computational speed. Mufalli et al. determined the routing of an UAV for a complex mission plan of approximately 30 targets. Therefore, the largest test for our model (i.e. 30 targets, two bases, and three obstacles) is a realistic test for the implementation of our model in mission planning algorithms. Our computation time of roughly one second for this large scenario shows that our model will not hinder the heuristic solution methods of mission planning or slow them down in their computations.

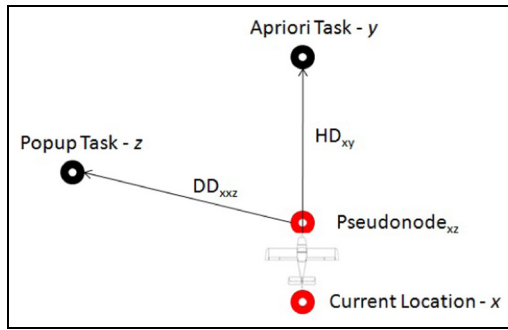
### 5.3. Dynamic reassignment

The dynamic resource management problem consists of the reassignment and rerouting of the UAVs to the new targets based on time-sensitivity or other priority requirements. The framework for a mathematical model for the multi-objective dynamic resource management problem has been developed and utilized in the literature.<sup>12</sup> This model is easily modified to include our method as data for their optimization models.<sup>49</sup>

The model includes the same three previously mentioned assumptions. The same reasons hold as to why our methods would improve the accuracy of the flight time calculations associated with this model and others used to solve the same problem. The same results listed for the implementation of our model for mission planning allow for the elimination of the three basic assumptions in dynamic rerouting. With the elimination of these three assumptions, our model must handle pop-up targets in order for dynamic rerouting to incorporate our model.

Consider a single UAV heading along a predetermined mission plan path from a target toward an a priori target  $y$ . If the current location is considered vertex  $x$ , the distance from the current location to the a priori target  $y$  is the Haversine distance between  $x$  and  $y$ . If a pop-up target is located at point  $z$ , then a pseudonode can be generated at current location  $x$ , storing the UAVs heading. The edge connecting Pseudonode<sub>xz</sub> to pop-up target  $z$  will be the Dubins' factor from  $x$  going through  $x$  and continuing toward point  $z$ . Figure 11 shows an example scenario. This shows a single UAV at a current location  $x$  on a heading towards a priori target  $y$ , when a pop-up occurs at point  $z$ . This figure shows how dynamic resource management problems involving UAVs can incorporate our model.

Since our model already has computed the large number of shortest paths for the scenario, the inclusion of a pop-up target only creates a small number of additional shortest paths. Considering our largest test case, one pop-up target only requires the shortest path from the current location to the pop-up target, as well as the shortest path from the pop-up target to every other potential target. This requires only an additional thirty-three shortest path calculations.



**Figure 11.** Implementation of pseudonodes in dynamic reassignment.

Since our model has previously computed the entire network and 992 shortest paths in roughly one second, our model can add to the network and compute thirty-three shortest paths in a very small amount of time.

It is shown by the framework of the network generation procedure, as well as the description of the implementation of this framework for pop-up targets in a dynamic reassignment problem, that a pop-up obstacle could also be handled. Our model can handle adversarial obstacles that arise during flight. This network generation procedure could capture those pop-up obstacles and recalculate the network in a short amount of time. This quick turnaround from the obstacle's addition into the operational field to the calculation of these flight times and paths would allow a dynamic reassignment algorithm to recalculate the mission plan.

## 6. Closing remarks

With the expanding role of autonomous vehicles in applications that range from the military domain to manufacturing facilities, an accurate use of routing methods is needed and desired. In the military domain specifically, UAVs are commonly used and their expanding use is highly beneficial. With a domain as dynamic as the military domain, use of UAVs in operational fields that consist of obstacles will be a common occurrence. The ability of mission planning algorithms and dynamic reassignment algorithms to deal with these obstacles and still provide accurate routing plans is an important aspect in the continued growth of UAV use.

While our research provides a more accurate method of incorporating flight dynamics in an operational field that contains obstacles, there are extensions to improve computation time and accuracy of the calculations. Some extensions will make the model more accurate for mission planning and dynamic reassignment algorithms, while others eliminate assumptions.

One aspect of our model that is not currently used is the checking of feasible paths. If obstacles are close enough together in the operational field, a path generated by our model could enter an obstacle and not be feasible. In order to determine if a path is feasible, the model must know the location of the UAV at all times. Avoiding the need to know the exact location of the UAV at all times is one of the main reasons that we developed the network generation procedure. The assumption that the obstacles are far enough apart for the UAV to most likely have feasible paths allows the model to run with this assumption. If required, this is feasible. It likely slows the calculations down to the point of not being usable for its intended purpose.

Removing the assumption about the UAV being on the rhumb line to start all flight paths chosen by the shortest path algorithm could be another extension for our model. This would increase the computation time of the model. In order to complete this extension, all entrance headings into a vertex would have their own vertex generated. Each of the new vertices needs a new calculation from the flight dynamics generator to determine the flight time for the associated edges. For example, in an  $(i, j, k)$  vertex triplet being traversed, the  $i$  vertex will need to be replaced with multiple  $i_{\text{degreeHeading}}$ . This means that for the degree separation  $\theta$  that is chosen, there needs to be  $\frac{360^\circ}{\theta}$  new  $i$  vertices that have associated edges to  $j$  with flight dynamics edge weights included. In addition to increasing the size of the network, this extension would also mean that for each  $i$  being considered as the source vertex in a vertex pair of interest, there would need to be the same number of shortest paths added for that node. Thus, in a large mission plan (such as scenario nine of our test scenarios), the original 992 shortest paths found would need to be replaced with  $992 * \frac{360^\circ}{\theta}$  shortest paths. This would cause the most drastic increase in computation time for the model. Smaller values of  $\theta$  would obviously increase this computation time more than larger values of  $\theta$ . Since our current model runs a complex mission plan in approximately one second, this extension might be able to be completed for larger values of  $\theta$ , and could improve the ability of mission planning algorithms to use our model for more accurate calculations of flight time. However, the mission planning algorithms that we have reviewed do not currently model the problem in a way that would allow them to handle this extension of changing vertex triplets  $(i, j, k)$  into quadruplets  $(i, i_{\text{degreeHeading}}, j, k)$ . The handling of this extension would be difficult with the current formulations used in mission planning algorithms that we have reviewed.

Along with the removal of the rhumb-line assumption, the removal of the constant altitude assumption also creates an interesting addition to the model. Changing the altitude would imply the biggest difference for obstacles.

Topographical obstacles would now maintain their true shape. Three-dimensional adversarial obstacles would also exist. This would most certainly increase the network size by a large amount. Unlike the previous extension, this would not increase the number of shortest path calculations required. The same SDA would be able to compute these shortest paths on the network. This method would be similar to the method used by Casas et al.<sup>38</sup> Their method of developing the network over airspace using a GIS creates a network that would avoid obstacles in the operational field while incorporating a change in altitude. Therefore, vertices in the network will be available at certain altitudes and not at others. This method cannot guarantee that the optimal path is located within the network, which is why we could not utilize this method directly for our work.

Another extension would be the removal of complete obstacle avoidance. Currently our model does not handle the minimum-risk planning associated with the chosen routes. The pseudonode structure and associated path selection method could lead to investigation of this area.

There are probably other possible extensions to enhance our model. Our model's ability to provide flight time calculations in an operational field that contains obstacles, while incorporating flight dynamics for complex mission plans in a short amount of time makes it implementable and usable in mission planning algorithms. Improving aspects of mission planning will likely improve the routing of UAVs. Their continued growth in use is an important aspect of the military domain, and they are also used by local authorities at the city and state level.

## Funding

Funding for this research was provided by the Office of Naval Research Optimization Planning and Tactical Intelligent Management of Aerial Sensors (OPTIMAS) grant N00173-08-C-4009 and Air Force Research Laboratories Platform and Data Fusion Technologies for Cooperative ISR, through IAVO, grant FA9550-09-C-0066, PO#30,40.

## References

1. USAF. The U.S. Air Force remotely piloted aircraft and unmanned aerial vehicle strategic vision, 2005.
2. Kingston D and Schumacher C. Time-dependent cooperative assignment. In: *Proceedings of the 2005 American control conference*, Portland, OR, USA, 8–10 June 2005, pp.4084–4089.
3. Mufalli F, Batta R and Nagi R. Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans. *Comput Oper Res* 2012; 39(11): 2787–2799.
4. Shetty V, Sudit M and Nagi R. Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles. *Comput Oper Res* 2008; 35: 1813–1828.
5. Shima T, Rasmussen S, Sparks A, et al. Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. *Comput Oper Res* 2006; 33: 3252–3269.
6. Weinstein A and Schumacher C. UAV scheduling via the vehicle routing problem with time windows. Technical report afrl-wa-wp-tp-2007-306, Air Force Research Laboratory, 2007.
7. Wilde J, Dibiaso D and Nervegna M. Team planning for unmanned vehicles in the risk-aware mixed-initiative dynamic replanning system. In: *Oceans 2007*, Vancouver, BC, Canada, 29 September–4 October 2007, pp.1–8.
8. Bellingham J, Tillerson M, Richards A, et al. Multi-task allocation and path planning for cooperating UAVs. In: *Cooperative control: Models, applications and algorithms*. Boston, MA, USA: Kluwer Academic Publishers, 2003, pp.1–19.
9. Kinney G, Hill R and Moore J. Devising a quick-running heuristic for an unmanned aerial vehicle (UAV) routing system. *J Oper Res Soc* 2005; 56: 776–786.
10. Schumacher C, Chandler P, Pachter M, et al. Optimization of air vehicles operations using mixed-integer linear programming. *J Oper Res Soc* 2007; 58(4): 516–527.
11. Berger J, Barkaoui M and Boukhtouta A. A hybrid genetic approach for airborne sensor vehicle routing in real-time reconnaissance missions. *Aerosp Sci Technol* 2007; 11: 31–326.
12. Murray C and Karwan M. An extensible modeling framework for dynamic reassignment and rerouting in cooperative airborne operations. *Nav Res Log* 2010; 57(7): 634–652.
13. Murray C and Karwan M. A branch-and-bound-based solution approach for dynamic rerouting of airborne platforms. *Nav Res Log* 2013; 60(2): 141–159.
14. Tavana M, Bailey M and Busch T. A multi-criteria vehicle-target allocation assessment model for network-centric joint air operations. *Int J Oper Res* 2008; 3(3): 235–254.
15. Dijkstra EW. A note on two problems in connexion with graphs. *Numer Math* 1959; 1(1): 269–271.
16. Larson RC and Li VO. Finding minimum rectilinear distance paths in the presence of barriers. *Networks* 1981; 11: 285–304.
17. Larson RC and Sadiq G. Facility locations with the Manhattan metric in the presence of barriers to travel. *Oper Res* 1983; 31(4): 652–669.
18. Sarkar A, Batta R and Nagi R. Commentary on facility location in the presence of congested regions with the rectilinear distance metric. *Socio Econ Plan Sci* 2004; 38: 291–306.
19. Butt S and Cavalier T. Facility location in the presence of congested regions with the rectilinear distance metric. *Socio Econ Plan Sci* 1997; 31: 103–113.
20. Sarkar A, Batta R and Nagi R. Finding rectilinear least cost paths in the presence of convex polygonal congested regions. *Comput Oper Res* 2009; 36: 737–754.
21. Lee D and Preparata F. Euclidean shortest paths in the presence of rectilinear barriers. *Networks* 1984; 14: 393–410.
22. Storer JA and Reif JH. Shortest paths in the plane with polygonal obstacles. *J ACM* 1994; 41(5): 982–1012.



23. Mitchell JSB. Shortest paths among obstacles in the plane. *Int J Comput Geom Ap* 1996; 6(3): 309–332.
24. Lee DT. *Proximity and reachability in the plane*. PhD Thesis, Champaign, IL, USA, 1978.
25. Welzl E. Constructing the visibility graph for n-line segments in  $O(N^2)$  time. *Inform Process Lett* 1985; 20(4): 167–171.
26. Mitchell JSB, Mount DM and Papadimitriou CH. The discrete geodesic problem. *SIAM J Comput* 1987; 16(4): 647–668.
27. Herschberger J and Suri S. An optimal algorithm for Euclidean shortest paths in the plane *SIAM J Comput* 1997; 28: 2215–2256.
28. Mitchell JS. Geometric shortest paths and network optimization, 1998. *Handbook of computational geometry*. North-Holland: Elsevier Science Publishers BV, 1998, pp.633–701.
29. Zabrankin M, Uryasev S and Murphey R. Aircraft routing under the risk of detection. *Nav Res Log* 2006; 53(8): 728–747.
30. Goldman J. Path planning problems and solutions. In: *Proceedings of the IEEE National Aerospace and Electronics Conference*, Dayton, OH, USA, 23–27 May 1994, pp.105–108.
31. Bertuccelli L, Choi H, Cho P, et al. Real-time multi-UAV task assignment in dynamic and uncertain environments. In: *AIAA guidance, navigation, and control conference*, Chicago, IL, USA, 10–13 August 2009.
32. Divelbiss AW and Wen JT. A path space approach to nonholonomic motion planning in the presence of obstacles. *IEEE T Robotic Autom* 1997; 13(3): 443–451.
33. Royset JO, Carlyle WM and Wood RK. Routing military aircraft with a constrained shortest-path algorithm. *Mil Oper Res* 2009; 14(3): 31–52.
34. Lewis L and Ross I. A pseudospectral method for real-time motion planning and obstacle avoidance. In: *Platform innovations and system integration for unmanned air, land, and sea vehicles (AVT-SCI Joint Symposium)*, Neuilly-sur-Seine, France 2007, pp.10–1 to 10–22.
35. Kunchev V, Jain L, Ivancevic V, et al. Path planning and obstacle avoidance for autonomous mobile robots: A review. *Knowledge-Based Intelligent Information and Engineering Systems, Pt 2, Proceedings* 2006; 4252: 537–544.
36. Latombe J. *Robot Motion Planning*. Boston, MA, USA: Kluwer Academic Publishers, 2001.
37. Chen D, Szczerba R and Uhran J. A framed quadtree approach for determining the Euclidean shortest paths in a 2-D environment. *IEEE T Robotic Autom* 1997; 13(5): 668–681.
38. Casas I, Malik A, Delmelle EM, et al. An automated network generation procedure for routing of unmanned aerial vehicles (UAVs) in a GIS environment. *Netw Spat Econ* 2007; 7(2): 153–176.
39. Kim J and Hespanha J. Discrete approximations to continuous shortest-path: Application to minimum risk path planning for groups of UAVs. In: *42nd IEEE conference on decision and control*, Maui, HI, USA, 9–12 December 2003, pp.1734–1740.
40. Helgason RV, Kennington JL and Lewis KR. Cruise missile mission planning: A heuristic algorithm for automatic path generation. *J Heuristics* 2001; 7(5): 473–494.
41. Caldwell T. On finding minimum routes in a network with turn penalties. *Commun ACM* 1961; 4: 107–108.
42. Movable Type Scripts. <http://www.movable-type.co.uk/scripts/latlong.html> (2011; accessed 7 July 2009).
43. Weisstein EW. Circle-line intersection. from mathworld – a Wolfram web resource. <http://mathworld.wolfram.com/Circle-LineIntersection.html> (2011; accessed 7 July 2009).
44. Bourke P. Intersection point of two lines (2 dimensions). <http://local.wasp.uwa.edu.au/pbourke/geometry/lineline2d/> (1989; accessed 7 July 2009).
45. Henchey M, Batta R, Karwan M, et al. A flight time approximation model for unmanned aerial vehicles: Estimating the effects of path variations and wind. *Mil Oper Res* 2014; 19(1): 51–68.
46. Dubins L. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *Am J Math* 1957; 79(3): 497–516.
47. Shkel A and Lumelsky V. Classification of the Dubins set. *Robot Auton Syst* 2001; 34: 179–202.
48. Grymin D and Crassidis A. Simplified model development and trajectory determination for a UAV using the Dubins set. Presented at the AIAA Atmospheric Flight Mechanics Conference, Chicago, IL, USA, 10–13 August 2009.
49. Murray C. *Dynamic reassignment and rerouting in cooperative airborne operations*. PhD Thesis, University at Buffalo, State University of New York, 2010.

### Author biographies

**David Myers** is a research engineer with the United States Air Force Research Laboratory in Rome, New York, USA. David completed his PhD at the University at Buffalo (SUNY) in 2013. He was selected as a Seth Bonder Scholar for Military Operations Research, NSF-IGERT Fellow in Geographic Information Science, 2012 Cohort of the SMART Scholarship for Service, and University at Buffalo Presidential Fellow.

**Rajan Batta** is a SUNY Distinguished Professor at the University at Buffalo (SUNY), Department of Industrial and Systems Engineering, Buffalo, New York, USA. Major awards include the David F. Baker Distinguished Research Award (2008) and the Albert G. Holzman Distinguished Educator Award (2015), both from the Institute of Industrial Engineers.

**Mark H. Karwan**, is the Praxair Professor in Operations Research and a SUNY Distinguished Teaching Professor at the University at Buffalo (SUNY), Department of Industrial and Systems Engineering, Buffalo, New York, USA.