Calculating Flight Time for Unmanned Aerial Vehicles in the Presence of Obstacles and the Incorporation of Flight Dynamics

David Myers, Dr. Rajan Batta, and Dr. Mark Karwan
Department of Industrial & Systems Engineering, University at Buffalo (State University of New York), Buffalo, NY, USA
djmyers2@buffalo.edu, batta@buffalo.edu, mkarwan@buffalo.edu

July 2010

Executive Summary

This paper considers the problem of calculating flight time for unmanned aerial vehicles (UAVs) while incorporating flight dynamics in an operational field that contains obstacles. These flight time calculations can be used as parameter inputs in mission planning and dynamic reassignment problems. A network generation procedure is developed and pseudonodes are added into the network to include flight dynamics in the edge weights. The network generation procedure includes a method for handling a pop-up target in a dynamic reassignment problem. A selective Dijkstra's algorithm was designed and implemented to ensure that the shortest path between two nodes of interest contains flight dynamics. A complex mission plan consisting of thirty tasks and three obstacles was 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 for a dynamic domain such as the use of mission planning algorithms and dynamic reassignment algorithms.

Application Area

Unmanned Systems

OR Methodology

Network Methods

INTRODUCTION AND PROBLEM DESCRIPTION

As the use of unmanned aerial vehicles grows in the military domain, important factors have and will emerge. In 2005, the United States Air Force (USAF) released "The U.S. Air Force Remotely Piloted Aircraft and Unmanned Aerial Vehicle Strategic Vision" (USAF 2005). In the strategic vision, an outline is given for what the USAF considered the future vision and recommendations should be for the use of UAVs. One of the major bullet points that was made in this document is that routing and path planning of these UAVs is an important factor in their future use in operations across the world.

UAV routing problems fall into one of two major categories. Mission planning algorithms are not dynamic and take place prior to the beginning of a mission. Dynamic routing algorithms are used when a change in the original mission plan is needed during flight. Non-dynamic mission planning algorithms can be seen vastly in literature (Kingston and Schumacher 2005; Mufalli et al. 2010; Shetty et al. 2008; Shima et al. 2006; Weinstein and Schumacher 2007; Wilde et al. 2007). Dynamic routing algorithms can have single objectives (Bellingham et al. 2001; Kinney et al. 2005; Schumacher et al. 2007) or multiple objectives (Berger et al. 2007; Murray and Karwan 2010; Tavana et al. 2008). They also may contain elements such as time windows for the tasks, payload capacity, fuel capacity, the ability to skip certain tasks, and precedence in targets among other features.

Problem Description

The purpose of this research is to devise a method by which the minimal flight time and associated travel route of a single UAV can be determined in a plane that consists of bases, tasks and polygonal obstacles. These obstacles can either be due to topography or be 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, tasks and obstacles on the plane. Many methods have been created for the routing of a UAV using obstacle avoidance within a plane from a single source to a single target. However, many of these methods use very 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 include flight dynamics and thus a shortest path; obstacle-free flight time that includes flight dynamics will be calculated for a UAV on the network 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, tasks, or a combination of the two. The shortest paths between all possible node pairs of interest are calculated. The goal is to include as large of a scenario as would be used in mission planning in a time-frame small enough for use in a dynamic domain. These flight time and travel route calculations are meant to provide more accuracy in flight time calculations to mission planning algorithms, as well as provide these algorithms with the ability to incorporate flight dynamics and obstacles.

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 in our network represent the distance between two nodes. There are four complications to our network generation procedure. First, the procedure is complicated by the inclusion of obstacles. Often used automatic generators cannot be used in this case because in the presence of obstacles they cannot create networks that guarantee that the optimal path between two nodes of interest is present in the network. The second complication exists when the assumption of a point-like object traversing throughout the network is removed and a model is used to include flight dynamics (e.g. a Dubins' model). This may require adding a large number of what we call pseudonodes to the network. This creates another problem because the shortest path algorithms tend to slow down when they are run on a very large network. Third, while the shortest path calculations seem simple most traditional shortest path algorithms, such as Dijkstra's algorithm (Dijkstra 1959) do not typically handle multiple types of nodes in a single graph, so modifications of this algorithm must be made in order to find the shortest path(s) that include these flight dynamics additions in our research. The biggest consideration of these complications is that 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.

PATH PLANNING WITH OBSTACLE AVOIDANCE

Path planning with obstacle avoidance can be defined as the maneuvering of a vehicle from one point to another point in an operational field while avoiding certain areas of that operational field. This problem exists for any type of vehicle in any appropriate operational environment. In our research, we explore UAV path planning with obstacle avoidance. These obstacles can be defined as topographical obstacles (e.g. mountains that extend into the flight altitude) or adversarial obstacles (e.g. stationary SAM sites).

Literature Review

The underlying problem of path planning with obstacle avoidance has been explored in operations research literature. Larson and Li (Larson and Li 1981) find the rectilinear solution to the problem of path planning in the presence of barriers. Larson and Sadiq (Larson and Sadiq 1983) apply this rectilinear solution to the problem of facility location in the presence of barriers for travel. Sarkar, Batta, and Nagi (Sarkar et al. 2004) have a commentary on previous work of Butt and Cavalier (Butt and Cavalier 1997) solving the inadequacies in their solution approach. The same authors (Sarkar et al. 2009) explore the rectilinear solution to the problem of path planning in the presence of convex polygonal congested regions. These all solve the rectilinear problem, which is not the same as the Euclidean shortest path problem in the presence of obstacles. Lee and Preparata (Lee and Preparata 1984) solve the Euclidean shortest path problem in the presence of rectilinear barriers. The barriers in our research are not only not solely rectilinear, but also our shortest path must contain flight dynamics, which is not the focus of any of these works.

Another of the fields that has studied this problem is the field of computational geometry. There are several methods that are used in this field for the shortest path problem. Mitchell (Mitchell 1996) and Storer and Reif (Storer and Reif 1994) utilize the continuous Dijkstra's paradigm to solve the obstacle avoidance problem in a plane. The continuous Dijkstra's paradigm has a fairly slow solution time in comparison to what is needed for our research, as well as they are only considering a single source and single vertex. Lee (Lee 1978) and Welzl (Welzl 1985) perform work in the area of using the visibility graph for the solution of the underlying shortest path problem. The creation of the visibility graph is complex in nature and this is also only concerned with a single source and vertex pair. Mitchell (Mitchell et al. 1987) explores the shortest path problem from the discrete geodesic problem point-of-view. Herschberger and Suri (Herschberger and Suri 1997) propose an optimal-time algorithm for the shortest path computation between two points in the presence of polygonal obstacles. Mitchell (Mitchell 1998) performs an extensive review of the geometric shortest path problem. These methods are also designed for a scenario that only includes a single source and destination pair.

The problem of path planning with obstacle avoidance has had many different perspectives studied and explored from several points-of-view that do not fall into one of the above stated fields of mathematics. Zabarankin et al. (Zabarankin et al. 2006) develop a method for three-dimensional risk minimization of aircraft detection. Goldman (Goldman 1994) presents a method for exploring the three-dimensional path

planning problem with circular danger regions. Our problem is considering a single UAV at a constant altitude, as well as does not have a risk consideration. Bertuccelli et al. (Bertuccelli et al. 2009) present a method by which they analyze task assignment of multiple UAVs in an uncertain environment. Our work is to provide flight time data for these task assignment problems. Divelbiss and Wen (Divelbiss and Wen 1997) present an algorithm for finding a feasible path using nonholonomic motion planning in the presence of obstacles. Their work is focused on ground based vehicles such as tractor trailers. Royset et al. (Royset et al. 2009) takes the approach of solving this problem using a constrained shortest path model that discretizes the relevant airspace into a grid. Discretization of airspace does not guarantee that the optimal path is included in the network. Lewis and Ross (Lewis and Ross 2007) write of a pseudo spectral method for real-time motion planning and obstacle avoidance. Their method is not in fact real-time and has a very slow solution time. Kunchev et al.(Kunchev et al. 2006) review methods and algorithms by which mobile robots and UAVs can perform obstacle-free paths. Their division of the work-area method would be very large for airborne operations and was meant for smaller ground-based work areas.

NETWORK DEVELOPMENT

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

Model Assumptions

Our model makes a few reasonable assumptions for its use. The first assumption is that the data for all of the vertices must be known prior to the network generation process. This means that the location of all of the obstacles should be known or approximated and defined. While this assumption is not necessarily always a true condition, with today's imagery and reconnaissance capabilities, this condition can often exist. For instance, if all of the obstacles in an operational field are topological, today's geographic information systems (GIS) software could easily be able to determine if a land feature obstructs the motion of a UAV at a certain altitude. This would create a situation where this condition is most certainly met and the data would be available for such a condition.

Another assumption that is needed is all bases, tasks, and obstacles must remain static throughout the process. Movement of vertices and obstacles is not handled by this model. Once again, if the obstacles are purely topological then they will obviously remain static. If the obstacles are caused by adversaries, they too often remain static during a given period of time and would be handled under the current condition of the model.

The assumption of treating a UAV as a point-like object is made in this model. The operational field that we are examining can easily be several to hundreds of square miles. This large of an operational field can easily make a UAV be reasonably approximated as a point-like object. Another assumption about the UAV itself is that it maintains a constant velocity throughout its travel time. The assumption of the UAV flying at a constant elevation is also used in this model. UAV control systems are often designed and operated to hold a constant velocity and can even attempt to maintain constant elevation during a mission, making this assumption reasonable.

Another assumption that is made in this model is that all of the paths that are developed by the model are feasible. This is satisfied by having a requirement that the obstacles are located a reasonable distance away from one another. With this requirement being held, all of the paths that will be generated should be close to if not perfectly feasible. This is a result of the obstacles are often defined by vague edges. For example, a stationary 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, it is likely allowed to enter the obstacle edge by a slight margin. As long as the provided information for the model has a slight exaggeration of the obstacles, any path that may enter the obstacle would still be feasible. In our experience, not only was it difficult to force a path to enter an obstacle, but the obstacles must be located closely to one another for this to occur.

A major assumption of this model is that when considering a pair of interest in the network, it is assumed that the UAV will always be facing on the rhumb-line towards the first vertex that it is traveling to. 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.

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 edges were added through the center of no fly zones or edges that where far away from these no fly zones were not added. The consistent results of the flat Earth approximation lead to its use.

Literature Review

There has been much work in the use of a network generation procedure for the purpose of path planning for not only UAVs but also mobile robots. Latombe's (Latombe 2001) work in robot motion planning uses a method of dividing the plane into cells for the network generation. Chen et al. (Chen et al. 1997) use a proposed algorithm that is developed by combining what they consider the advantages of the grid based methodology that has been presented in other papers and the quad tree method. Casas et al. (Casas et al. 2007) present a method whose objective is to create an automated generalized network generation procedure within a geographical information system (GIS) environment. Any of these above methods use a discretization method for the relevant airspace or operational field. This does ont guarantee that the optimally shortest path is included in the network. Helgason et al. (Helgason et al. 2001) develop 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. This heuristic is used in place of a 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.

Network Generation

Our network generation procedure consists of the following steps:

- Model Input
- Vertex Addition
- Circle Estimation
- Obstacle Defining Edge Addition
- Other Edge Addition

Model Input

Input for the model is a .csv (comma separated values) file where each row contains a vertex name, an obstacle number, a vertex number, an obstacle shape, a latitude, a longitude, and a radius. The vertex name must be a unique string of characters for the entire network. 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 is 0 for a base or target vertex, 1 if the shape of the obstacle is defined by points only, and 2 if the obstacle is circular. 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.

Vertex Addition

The vertices in the network represent a base, a target, or an obstacle. These vertices are represented by the latitude and longitude that is provided by the model input.

Circle Estimation

If a vertex has an obstacle shape value of 2, then it represents a circular obstacle. If this is found in the model input, 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 which is the apothem or radius of the inscribed circular obstacle as provided by the model input to r_o which is the radius of 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 gives:

$$r_0 = \frac{s}{2 * sin(\frac{\pi}{n})} \tag{1}$$

Pythagorean theorem then yields:

$$r_0^2 = \frac{s}{2}^2 + r_c^2 \tag{2}$$

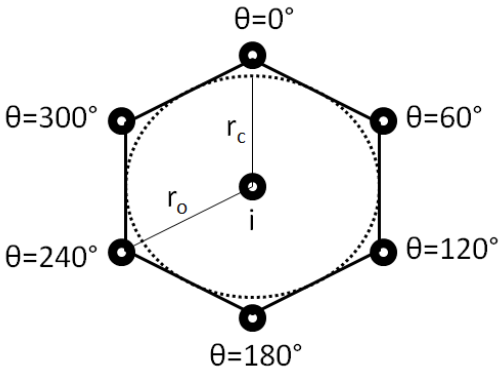


Figure 1: Circle Estimation Using Hexagonal Approximation

Substituting using equation 1 and solving for r_o in terms of r_c gives:

$$r_0 = \frac{r_c}{cos(\frac{\pi}{n})} \tag{3}$$

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 θ, the latitude of the destination point would be the result of equation 4 (Movable Type Scripts). The longitude of the destination point would be the result of equation 5 (Movable Type Scripts).

$$asin(sin(lat_1) * cos(\frac{d}{R}) + cos(lat_1) * sin(\frac{d}{R}) * cos(\theta)) \tag{4}$$

$$long_1 + atan2[sin(\theta) * sin(\frac{d}{R}) * cos(lat_1), cos(\frac{d}{R}) - sin(lat_1) * sin(lat_2)] \tag{5}$$

Using these formulas at evenly spaced headings around the circular obstacle allows us to estimate the circular obstacle with any number of points. Using the above equations, and spacing these vertices at $\theta = \frac{m}{360}$ for $m = 0, 1, 2, ..., n-1$, with n being the number of sides chosen to estimate the circular obstacle. We determined that $n = 6$ was a good representation of the circular obstacle without slowing the network generation procedure down. An example of taking a single point and estimating a circular obstacle around that single point using a hexagonal approximation ($n = 6$) is shown in Figure 1.

Obstacle Defining Edge Addition

Obstacle defining edges are added during the iteration through the model input. The obstacle defining vertices are listed first and together in the input and if the iterator finds that a vertex is next in line and belongs on the same obstacle, a directed edge is added between the first and second vertex in the list. When a dummy vertex (a vertex with a 0 value for vertex number on the obstacle) is reached, a directed edge is added from the last vertex in the list on the obstacle to the first vertex on the obstacle.

The edge weights for these edges is set to the Haversine distance between the two end points of the edge. This Haversine distance is the great circle distance across the Earth's surface (Movable Type Scripts). The formula for the Haversine distance between two points on the Earth's surface is done by the following:

$$\nabla lat = lat_2 - lat_1 \tag{6}$$

$$\nabla long = long_2 - long_1 \tag{7}$$

$$a = sin^2((\nabla lat)/2 + cos(lat_1) * cos(lat_2) * sin^2((\nabla long)/2)) \tag{8}$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a}) \tag{9}$$

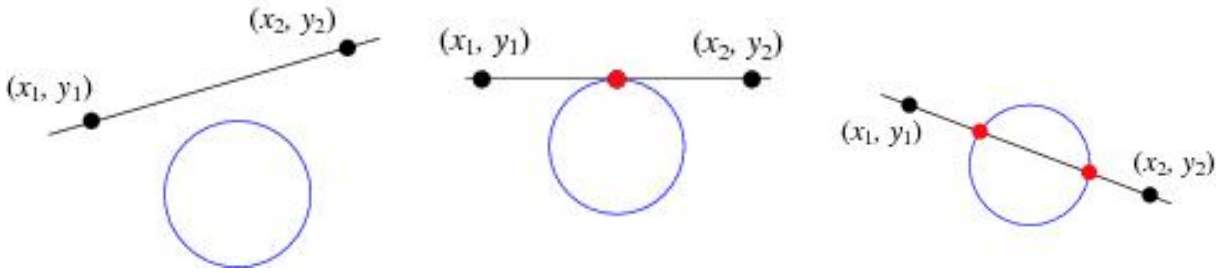
$$d = R * c \tag{10}$$

Other Edge Addition

The next step in the network generation procedure is the addition of edges that are not obstacle defining edges. All possible edges are considered for addition into the network at this step in the procedure. In order for a possible edge to be added into the network, it must not intersect with circular obstacles or 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. These are described in Figure 2 (Weisstein).

The method for checking which case the possible edge falls into is completed by:

$$d_x = x_2 - x_1 \tag{11}$$



(a) Case 1: no intersection (b) Case 2: one intersection (c) Case 3: two intersections

Figure 2: Circle-Line Intersection Cases

$$d_y = y_2 - y_1 \tag{12}$$

$$d_r = \sqrt{d_x^2 + d_y^2} \tag{13}$$

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

The points of intersection are:

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

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

The function $sgn^*(x)$ is defined by:

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

The discriminant is then:

$$\Delta = r^2 d_r^2 - D^2 \tag{18}$$

The summary of the existence of intersection point(s) is shown in Table 1.

Table 1: Summary of Existence of Intersection Point(s) of Circle and Line

Δ	**incidence**
$\Delta < 0$	no intersection
$\Delta = 0$	tangent intersection
$\Delta > 0$	intersection

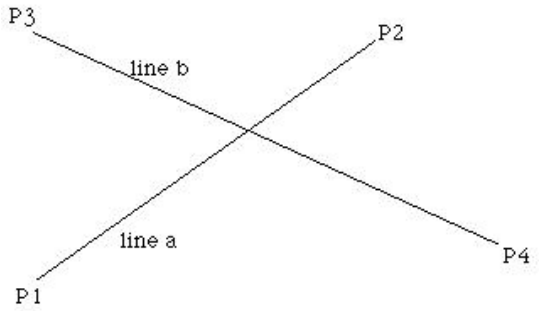


Figure 3: Line Segment Intersection

If the line created by the vertex pair of interest has a Δ 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. This last step is to determine if the intersection points occur within the line segment that is defined by each vertex of interest on the line. This is completed by determining if the intersection points from equations 15 and 16 fall between the vertices of interest in the pair. If either of the intersection points falls along the line segment being considered, 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 non-circular obstacle defining edges is done using the geometric formulas for line segment intersection (Bourke 1989). Figure 3 shows two lines for the purpose of an example of line segment intersection and to be considered in the formulas for line segment intersection.

The equations of the line segments in Figure 3 are:

$$P_a = P1 + u_a(P2 - P1) \tag{19}$$

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

$$P_b = P3 + u_b(P4 - P3) \tag{20}$$

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.

$$x_1 + u_a(x_2 - x_1) = x_3 + u_b(x_4 - x_3) \tag{21}$$

$$y_1 + u_a(y_2 - y_1) = y_3 + u_b(y_4 - y_3) \tag{22}$$

Solving these equations for u_a and u_b results in the following equations.

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

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

By substituting either of these values into the equation of the lines located above, the intersection point of the line segments is given by the following equation.

$$x = x_1 + u_a(x_2 - x_1) \tag{25}$$

$$y = y_1 + u_a(y_2 - y_1) \tag{26}$$

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

If the line segments defined by the possible edge pass the line segment intersection test for all obstacle edges that define non-circular obstacles in the operational field then it has passed the line segment

intersection check.

When a possible edge (i, j) passes the circular obstacle intersection check and the non circular obstacle intersection check, that edge is added to the network. The edge's 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 now contains half of the edges that a directed network without flight dynamics should have. At this point, the model takes every edge that exists in the graph and creates the directed edge that is opposite to that. That is for each edge (i, j) this part of the model creates an edge (j, i) and sets the edge weight to the weight of (i, j). This complete network G_{NFD} is the network on which any non-flight dynamics tests or analyses were completed.

Flight Dynamics

Once G_{NFD} is completed, the next step in the network generation procedure is to include flight dynamics. The contribution of this research is in the inclusion of flight dynamics in the network generation procedure and in the calculations of flight time in the presence of obstacles.

The use of pseudonodes is the method that we developed to include flight dynamics into the calculation of flight time. A pseudonode is a node that 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 will be called the previous node, and the node that the UAV is heading towards after the pseudonode will be called the next node. For example, if a 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 called a node triplet. This addition of pseudonodes allows for the inclusion of flight dynamics into the network. The reason that flight dynamics are needed in the network is that the actual flight time for a UAV to traverse a specific edge in the network might not just be the straight line distance along that edge. An example of the actual path of a UAV along a triplet i, j, k is shown in Figure 4. As can be seen from the figure, the flight time along the edge (j, k) will not be equal to the straight line distance along edge (j, k) if the UAV is coming from node i.

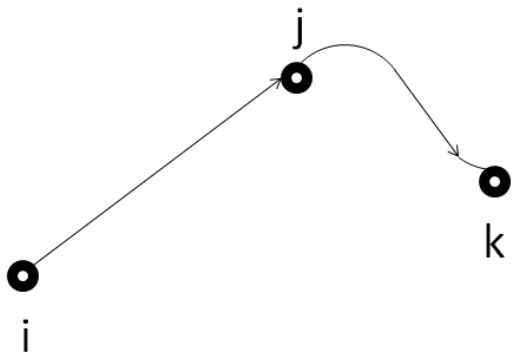


Figure 4: Flight Path Along a Vertex Triplet

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 though j on it's way to k, there must be edges that exist for this edge sequence in G_{NFD}.

The next condition is used in order to limit the number of pseudonodes that are added to the network. 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 exists between task i and task k (no obstacle is in the way), then the shortest path between these two tasks 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. If this condition is not met, a lot of unnecessary pseudonodes will be created between node triplets that only contain tasks. These pseudonodes will never be used in the shortest path calculations between an individual pair of tasks. The shortest path between two tasks in an operational field will begin and end at a node that designates a task, and any nodes in between will be nodes that define obstacles. Thus, any pseudonodes that would be created in a triplet that only contains tasks would not be used in any of the shortest paths that are desired.

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 edge weight of this edge is the Haversine distance that was set as the edge weights in the edges that were originally added in G_{NFD}. Next, a directed edge from node j_{ik} to node k is added and the edge weight of this edge is set to what is referred to as a Dubins' factor for distance. This Dubins' factor comes from a Dubins' Curve Path Model for approximating a minimum distance path for a Dubins flight model (Henchey 2010). Any flight dynamics model can be used in order to determine this edge weight. Dubins' original work (Dubins 1957) lead to work by Skhel and Lumelsky (Shkel and Lumelsky 2001) that limited the number of calculations needed to determine the minimum distance path for a Dubins problem. This work lead to the speed of the Dubins' Curve Path Model. The accuracy of the Dubins' Curve Path Model that is used is shown in Grymin and Crassidis (Grymin and Crassidis 2009). Our research chooses the Dubins' Curve Path Model for approximating the actual flight time between point j and point k originally coming from point i due to this speed of computation and good accuracy.

Since the Dubins' Curve Path Model uses a flat Earth approximation, we needed to scale the approximated flight time using the Haversine distance that we used for the other edge weights. This can easily be scaled as shown in equation 27, where DD_{ijk} is the new Dubins' factor that is used for the edge weights here 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.

$$DD_{ijk} = HD_{jk} * \frac{DCP_{ijk}}{SLD_{jk}} \tag{27}$$

An example of the addition of a pseudonode into a triplet (i, j, k) is shown in Figure 5. This figure shows the addition of a vertex and two directed edges and the setting of these edge weights on the triplet. The vertices and edges that are black would have been added during the process of developing G_{NFD}. The vertices and edges that are dashed are added during the process of including flight dynamics.

The next step is to connect all of the pseudonodes that should be connected. There is only one condition that must be met in order to determine if two pseudonodes should be connected. If the next node of one pseudonode corresponds to the node that the second pseudonode relates to in G_{NFD}, then the two pseudonodes should be connected. In other words, if pseudonode j_{ik} and k_{jl} both exist in the network, then the future node of the first pseudonode is node k and the current node of the second pseudonode is

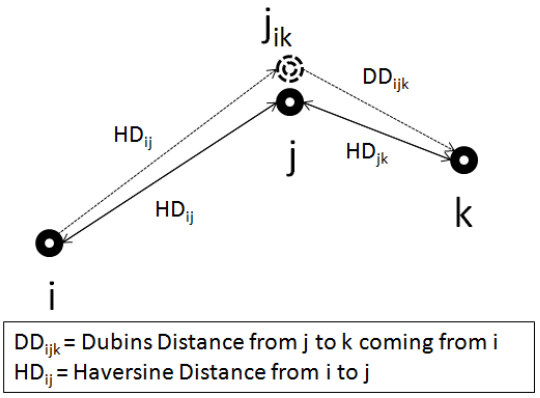


Figure 5: Addition of a Pseudonode to a Vertex Triplet

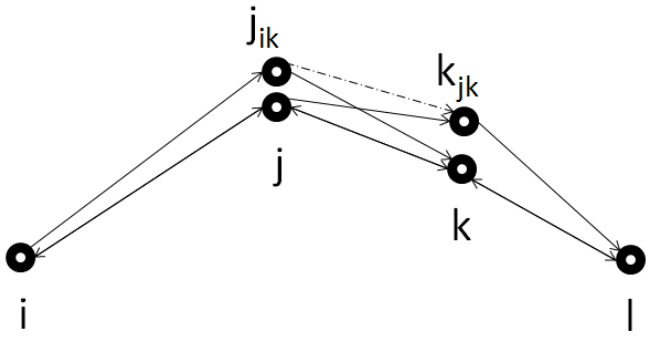


Figure 6: Connection of Pseudonodes

also node k. With this condition being met, a directed edge from j_{ik} to k_{jl} is added. The edge weight of this edge is set to be the same Dubins' factor distance that was shown previously. An example of this connection between pseudonodes is shown in Figure 6. The vertices and edges that are black would have been added previously in the procedure. The dashed edge indicates this edge that is added 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_{NFD} is now G_{FD}. G_{FD} will represent the network on which any flight dynamics tests or analyses were completed. This is the complete network for this research.=

SELECTIVE DIJKSTRA'S ALGORITHM

The last step in calculating the flight time between desired nodes on the network is to calculate the shortest path between these desired nodes. Given a scenario of bases, tasks, and some obstacles existing in the operational field, mission planning will need accurate flight time calculations. Let $G = (V, E)$ be the directed network that was developed with the previously mentioned procedure. In order to calculate the shortest flight time from any source vertex S to any target vertex T, the shortest path from S to T on G must be calculated.

While all pairs of tasks and bases are analyzed, each pair will fall into one of two cases. Case One is the case in which there exists an edge in the network that connects the origin of the pair to the destination of the pair. The shortest path in this case is obviously that single link between the origin and destination of the pair. Case Two is the case where there does not exist an edge in the network that connects the origin and destination of the pair. The shortest path in this case must be determined by 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.

Traditional Dijkstra's Algorithm (TDA) can be described as 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 network is easily developed over roads or other methods of travel. In this research, the network was generated over the operational airspace of the UAV.

Let distance[X] be the distance from the source vertex S to the vertex X along it's shortest path between V and X. Let parent[X] be the parent vertex of vertex X along it's shortest path between V and X. The steps below represent the steps of the TDA.

1: **for** $V \in G$ **do**
2: distance[V] = ∞
3: parent[V] = undefined
4: **end for**
5: distance[S] = 0
6: Q = set of all nodes in G
7: **while** Q is not empty **do**

```
    u = vertex in Q with smallest distance[]
    if distance[u] = ∞ then
        break
    end if
    remove u from Q
    for (neighbor v of u) ∈ Q  do
        tempDistance = distance[u] + edgeWeight[u,v]
        if tempDistance < distance[v] then
            distance[v] = tempValue
            parent[v] = u
        end if
    end for
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. It u is the target, then let S be an empty list. In order to obtain the edges that develop the shortest path:

```
while parent[u] is defined do
    insert u at beginning of S
end while
```

After this procedure, the list S will be the list of nodes in the shortest path, and a list of edges E_{path} can easily be determined from this. E_{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) was necessitated by the need to modify 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, s and t are initialized into Q. 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 in order to determine the true shortest path within the network that will include flight dynamics during the full length of the path. 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.

The above selectivity can also be completed on G_{NFD} if desired. The only change to the above

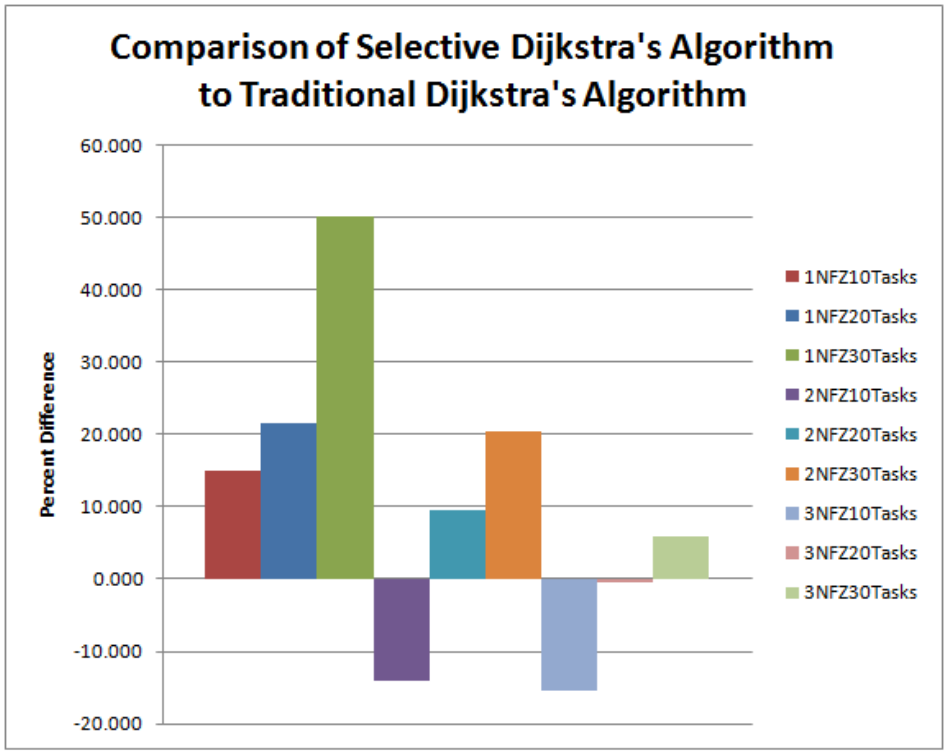


Figure 7: Selective Dijkstra's Algorithm vs. Traditional Dijkstra's Algorithm on G_{NFD}

algorithm would be to initialize the obstacle defining vertices into Q instead of the pseudonodes. This would allow for analysis on G_{NFD} using this algorithm.

In running tests of the SDA, it was noticed that some scenarios ran faster on G_{NFD} than using the TDA. So a comparison of the percent difference of running the SDA and TDA on the G_{NFD} was completed. This comparison is shown in Figure 7.

This figure shows that while some scenarios have a faster run time with SDA in comparison to the TDA, as the scenario grows, the time difference becomes minimal. However, the SDA is needed on G_{FD} and therefore still needed to be implemented in the model.

One possible explanation for why the SDA performed better in the scenarios only where it was needed can be seen in Table 3. 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 non flight dynamics. This table clearly shows that when the network goes from G_{NFD} to G_{FD}, the density of the network drops drastically. This is because for every pseudonode that is added to the network, two edges are added, and thus the density would decrease in the network. As can be seen from this table, the density of the G_{NFD} is much more varied than the density of the G_{FD} in both general edges per vertex and for obstacle edges per vertex.

Table 3: Network Density Comparisons

										Avg
Obstacles	1	1	1	2	2	2	3	3	3	
Tasks	10	20	30	10	20	30	10	20	30	
Bases	2	2	2	2	2	2	2	2	2	
NFD V	16	26	36	20	30	40	24	34	44	
NFD E	134	434	856	244	518	932	310	618	968	
NFD V to OE	36	68	93	100	147	199	146	209	266	
NFD E per V	8.4	16.7	12.8	12.2	17.3	23.3	12.9	18.2	22.4	17.2
NFD OE per V	2.3	2.6	2.6	5.0	4.9	5.0	6.1	6.1	6.0	4.5
FD V	108	296	488	388	838	1496	590	1270	2062	
FD E	470	1244	2352	1552	3658	6696	2420	5396	9276	
FD V to PE	244	540	1044	940	2332	4308	1544	3542	6272	
FD E per V	4.4	4.2	4.8	4.0	4.4	4.5	4.1	4.2	4.5	4.3
FD OE per V	2.3	1.8	2.1	2.4	2.8	2.9	2.6	2.8	3.0	2.5

IMPLEMENTATION

The calculation of these flight times using the method described in the previous sections can be utilized in two main areas of research for the routing of vehicles. For the purpose of describing the implementation of our work, the two areas of application that would benefit most is mission planning and dynamic reassignment of UAVs. General results from our research are given, and specific implementation in mission planning and dynamic reassignment will be described.

Results

All test cases, computations, and results come from running the Java model on a Dell Latitude D830 Laptop with an Intel Core 2 Duo T9300 CPU at 2.5GHz with 4GB RAM. The first analysis that was performed on the model was to determine if generating one full network was faster than generating a separate network for each vertex pair of interest. The network generation time and shortest path calculation time using TDA on the non-flight dynamics networks were added together for the total time for each pair. Then, five pairs of interest were chosen and their times were summed. These times that are shown are the average of five runs for each vertex pair of interest. The results of this analysis are shown in Table 4.

Table 4: Network per Pair vs. Network per Scenario

Pair Number	**Run-Time (sec)**
1	.1353
2	.1183
3	.1143
4	.1330
5	.1193
Total	.6203
Single Large Network	.5487

Table 5: 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
Tasks	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)	.0698	.2093	.2762	.1756	.3181	.4669	.2151	.5436	.5935
SPF	132	132	132	462	462	462	992	992	992
SPT (sec)	.1199	.2814	.3599	.3267	.5278	.6661	.4329	.8157	.9909

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 sample scenarios that were used to test the model. The sample scenarios, their network size, and their associated network generation time with flight dynamics included, and time to calculate all shortest paths of interest can be found in Table 5. In this table, NGT stands for the network generation time of the scenario. SPF stands for the shortest paths found while computing the scenario. 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 that are shown here are the average of ten runs of each scenario.

The fast computation time of the largest scenario shows that this model can be used in algorithms in substitution for straight line flight paths and will not slow down the algorithm. We judged that the

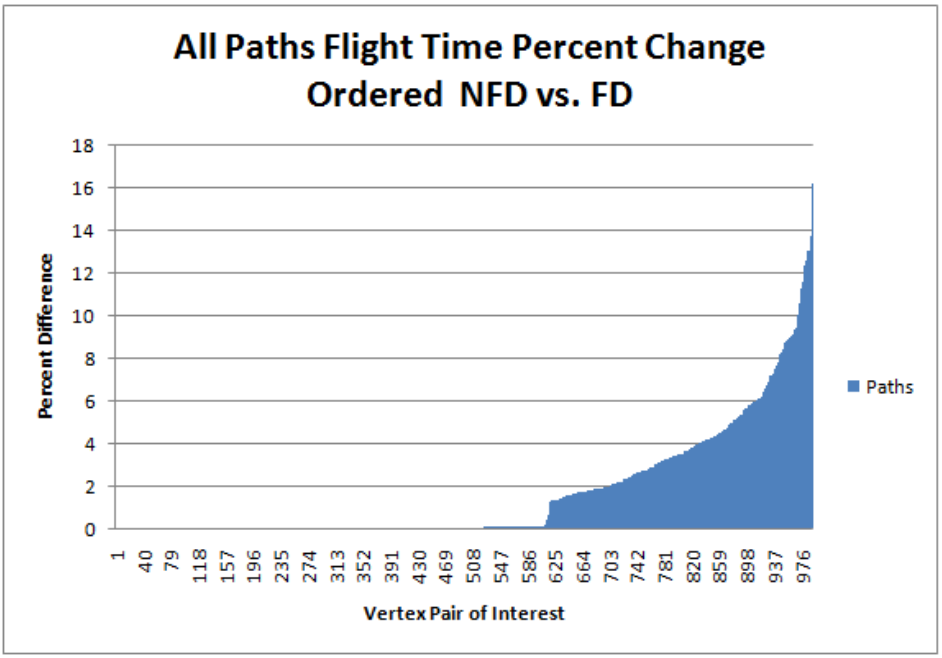


Figure 8: Flight Time Percent Change for All Paths Ordered

dynamic domain of this model would not be hindered by using our method of including flight dynamics in its calculations.

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. Percent difference is defined in equation 28.

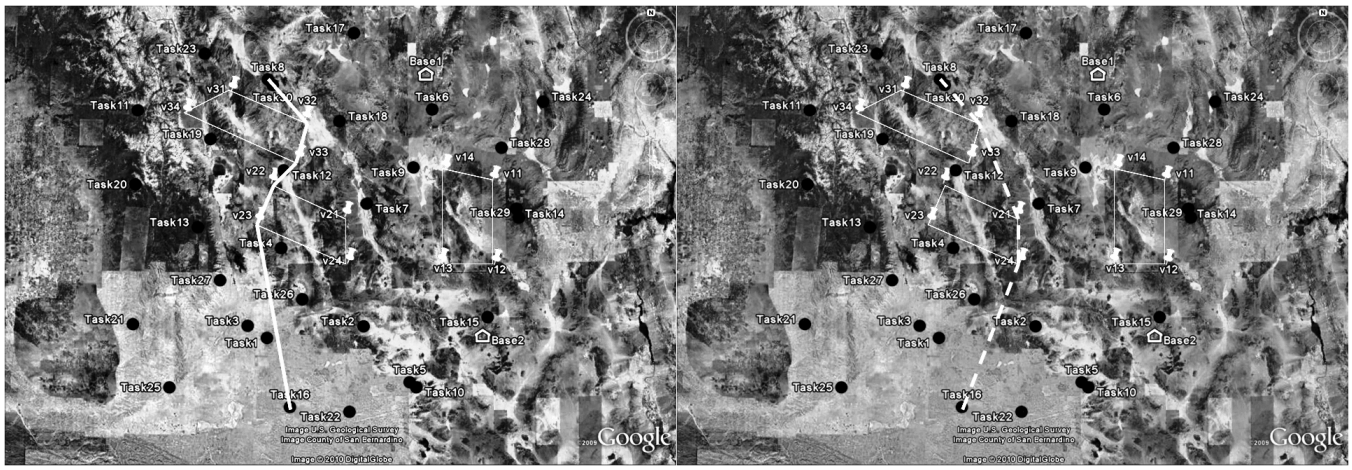
$$PercentDifference = \frac{|(FD_{time} - NFD_{time})|}{\frac{(FD_{time}+NFD_{time})}{2}} * 100 \tag{28}$$

We chose to complete this analysis on the largest scenario that we tested the model on. This scenario consisting of thirty tasks, two bases, and three obstacles was the scenario that was chosen. Figure 8 shows the percent difference of all the flight times for all 992 paths that were calculated by the model for this scenario. This figure has them ordered from smallest percent change to largest percent change.

An important step in determining if this model will create a more accurate flight time calculation that is currently used is determining if any flight paths will change as the result of including flight dynamics.

Table 6: Paths in Largest Sample Scenario

Total Paths	992
Single Edge Paths	470
Long Paths	522
Changed Paths	37
Percent Paths Changed	7.0881



(a) G_{NFD} Task 8 to Task 16 (b) G_{FD} Task 8 to Task 16

Figure 9: Example of Changed Path

Table 6 shows the breakdown of the 992 paths that existed in the largest sample scenario. A long path is defined as a path that has more than a single edge contained in it. A changed path is defined as a path that contained different vertices when computed using G_{FD} than when it was computed using G_{NFD}. With the results of this table, it confirms that this model will in fact make a difference and change paths from G_{NFD} to G_{FD}. An example of a path that was changed is shown in Figure 9. The solid line in Figure 9a represents the non-flight dynamics shortest path between 2 tasks in the sample scenario. The dashed line in Figure 9b represents the flight dynamics shortest path between the same 2 tasks in the same sample scenario. This shows that in fact adding flight dynamics and recalculating the shortest path can change the path taken.

In order to gain more insight into how these changed paths line up with the total paths percent change in flight time, Table 7 was developed. It shows six categories for percent change of the flight times and how the total number of paths breaks down into these categories, as well as where the changed paths fall in these cases. Using the results in this table it was determined that the changed paths do not fall into a

Table 7: Comparison of Total Paths to Changed Paths with Percent Difference

Case	% Diff. Range	Paths	% Paths	Changed Paths	% Paths
1	0	522	52.6210	0	0
2	0 to 1	95	9.5766	7	18.9189
3	1 to 2	85	8.5685	6	16.2162
4	2 to 5	176	17.7419	17	45.9459
5	5 to 10	92	9.2742	6	16.2162
6	10+	22	2.2177	1	2.7027
Total		992		37	

single case for determining the flight time difference. These percent differences are based on the percent difference of ten runs of the model for all nine of the test scenarios. For this comparison we separated the network generation time from the total computation time, allowing for the percent difference to be computed for just the time it takes to calculate the shortest path on the network.

With the analyses described above, it is reasonable to assume that the model that was developed in our research 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, as well as difficult to estimate which paths would change in the scenario.

Mission Planning

Mission planning is the first area of application that will benefit from a more accurate calculation of flight time as provided by our research. Mission planning is often times treated as an instance of the traditional operations research problem of the Traveling Salesman Problem (TSP). Since this is an NP-hard problem, heuristics must be used in order to generate a feasible solution for a moderately to large sized problem in reasonable time. Mufalli et al. (Mufalli et al. 2010) is an example of such a method used to determine a route selection for complex missions. In this paper they use column generation as the basis for developing these mission plans. Implementing our model into algorithms such as this one by Mufalli et al. can easily be done. Running our model on the scenarios being analyzed by the algorithms will provide the distance or flight time data that is needed. This data can be used in the Mufalli model as well as other mission planning algorithms wherever it is required. Most mission planning algorithms that are seen in the literature utilize flat Earth approximated distances, straight line flight paths, and consist of an obstacle

free operational field for the UAV. These three assumptions can be removed from the mission planning algorithms with implementation of our model.

The first assumption that can be removed is the use of a flat Earth approximated distances. Since our model calculates the edge weights either as the Haversine distance or a scaled version of the Haversine distance including flight dynamics, the model includes no distances that are calculated using the flat Earth approximation for distance.

The second assumption that can be removed is that of the straight line flight paths. Since the pseudonode method is included in our network generation procedure, flight dynamics is automatically included in the flight time for any path that has pseudonodes in it. This method and the setting of the edge weights as previously described allows for our research to guarantee that flight dynamics is included in the solution.

The third assumption that can be removed is the obstacle-free operational field. The testing of the model with both circular and non-circular obstacles located in the operational field shows that this methodology works with operational fields that have obstacles located in them. This will allow for mission planning to be completed in operational fields that contain obstacles.

The most important factor in terms of implementing this model is computational speed. Mufalli et al. determine the routing of a UAV for a complex mission plan of around thirty tasks. Therefore, our largest test for our model that consisted of thirty tasks, two bases, and three obstacles is a realistic test for the implementation of our model in mission planning algorithms. Our computation time of this large of a scenario in roughly one second shows that our model can be implemented even with the heuristic solution methods of mission planning and not slow them down in their computations.

Dynamic Reassignment

The next area of UAV routing application that would benefit from use of our research is dynamic reassignment. The underlying operations research problem is the dynamic resource management problem. This problem consist of the reassignment and rerouting of the UAVs to the new tasks based on time-sensitivity or other priority requirements. In Murray and Karwan (Murray and Karwan 2010), the framework for a mathematical model for multi-objective dynamic resource management problem is developed and utilized. Their model is easily modified (Murray 2010) to include our results as data for their optimization models.

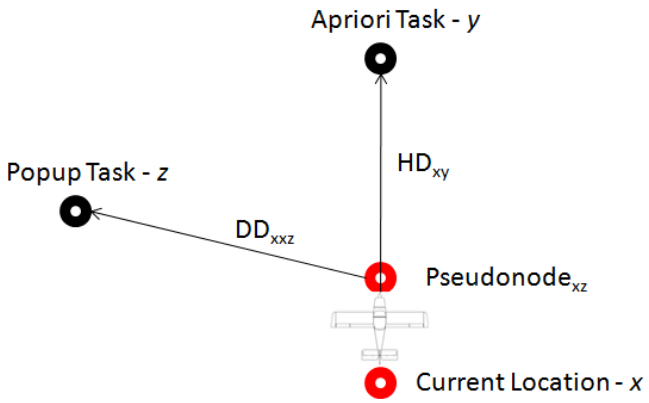


Figure 10: Implementation of Pseudonodes in Dynamic Reassignment

The same three previously mentioned assumptions are made in this model. The same reasons hold for why our model would improve the accuracy of the flight time calculations associated with this model and others used to solve the same problem. With these three assumptions being helped, there is one more stage of implementing our research that needs to be addressed. While the same results listed for the implementation of our model for mission planning can be used for the above assumptions, the addition of popup targets into the model results in another method being needed for the addition of the pseudonode for incorporation of flight dynamics.

Consider a single UAV heading along a predetermined mission plan path from a task toward an a priori task y. If the current location is considered vertex x, the distance from the current location until reaching the a priori task y is the Haversine distance between x and y. If a popup task is located at point z, then a pseudonode can be generated at current location x, storing the UAVs heading. The edge connecting $Pseudonode_{xz}$ to pop-up task z will be the Dubins' factor from x going through x and continuing toward point z. An example scenario can be found in Figure 10. This shows a single UAV at a current location x on a heading towards a priori task y, when a pop-up occurs at point z. This figure shows how our research can be used in dynamic resource management problems involving UAVs.

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 that could be completed to improve computation time and accuracy of the calculations. These extensions are centered around either making the model more accurate for all scenarios for mission planning and dynamic reassignment algorithms or just geared towards reducing the assumptions that were made.

One aspect of our model that is not currently used is the checking of feasible paths. If obstacles are placed 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 location of the UAV must be known 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 placed far enough apart for the UAV to most likely have feasible paths allows for the model to run with this assumption. If needed, this could be completed. It likely slow the calculations down to the point of not being usable for its intended purpose.

Removing the assumption that was made 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. If this extension was completed, new vertices will need to be added for all entrance headings into a vertex prior to finding the edge weight towards the next vertex in the flight path. Each of these vertices that would be added will need a new flight dynamics generator calculation 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 $i_{degreeHeading}$. This means that for the degree separation θ that is chosen, there needs to be $\frac{360^\circ}{\theta}$ new i vertices that has 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

θ would obviously increase this computation time more than larger values of θ. 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 θ in the small time frame given for these calculations and 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 $(i, i_{degreeHeading}, j, k)$. 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, removal of the constant altitude assumption also creates an interesting addition to the model. This changing of altitude would mean the most difference for obstacles. Topographical obstacles would be able to now maintain their true shape. Three-dimensional adversarial obstacles as well would be used. 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 that need to be run. The same SDA would be able to compute these shortest paths on the network. This method would be similar then to the method used in Casas et al. (Casas et al. 2007). 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. This is why this method could not be used directly for our work.

There are likely other extensions that could be completed on our model to enhance its ability to be used. 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 as described by the USAF in 2005. Their continued growth in use is an important aspect of the military domain, as well as by local authorities at the city and state level.

References

J. Bellingham, M. Tillerson, A. Richards, and J. How. Multi-task allocation and path planning for cooperating UAVs. In *Cooperative Control: Models, Applications and Alorithms*, pages 1–19, 2001.

J. Berger, M. Barkaoui, and A. Boukhtouta. A hybrid genetic approach for airborne sensor vehicle routing in real-time reconnaissance missions. *Aerospace Science and Technology*, 11:31–326, 2007.

L. Bertuccelli, H. Choi, P. Cho, and J. How. Real-time multi-UAV task assignment in dynamic and uncertain environments. In *AIAA Guidance, Navigation, and Control Conference*, 2009.

P. Bourke. Intersection point of two lines (2 dimensions) . http://local.wasp.uwa.edu.au/ pbourke/geometry/lineline2d/, 1989.

S. Butt and T. Cavalier. Facility location in the presence of congested regions with the rectilinear distance metric. *Socio-Economic Planning Sciences*, 31:103–113, 1997.

I. Casas, A. Malik, E. M. Delmelle, M. H. Karwan, and R. Batta. An automated network generation procedure for routing of unmanned aerial vehicles (UAVs) in a GIS environment. *Networks and Spatial Economics*, 7(2):153–176, 2007.

D. Chen, R. Szczerba, and J. Uhran. A framed quadtree approach for determining the euclidean shortest paths in a 2-D environment. *IEEE Transactions on Robotics and Automation*, 13(5):668–681, 1997.

E. W. Dijkstra. A note on two problems in conexion with graphs. *Numerische Mathematik*, 1959.

A. W. Divelbiss and J. T. Wen. A path space approach to nonholonomic motion planning in the presence of obstacles. *Ieee Transactions on Robotics and Automation*, 13(3):443–451, 1997.

L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

J. Goldman. Path planning problems and solutions, 1994.

D. J. Grymin and A. L. Crassidis. Simplified model development and trajectory determination for a UAV using the dubins set, 2009.

R. V. Helgason, J. L. Kennington, and K. R. Lewis. Cruise missile mission planning: A heuristic algorithm for automatic path generation. *Journal of Heuristics*, 7(5):473–494, 2001.

M. J. Henchey. A flight time approximation model for unmanned aerial vehicles: Estimating the effects of flight dynamics and wind. Master's thesis, University at Buffalo, State University of New York, 2010.

J. Herschberger and S. Suri. An optimal algorithm for euclidean shortest paths in the plane, 1997.

D. Kingston and C. Schumacher. Time-dependent cooperative assignment. In *Proceedings of the 2005 American Control Conference*, pages 4084–4089, 2005.

G. Kinney, R. Hill, and J. Moore. Devising a quick-running heuristic for an unmanned aerial vehicle (UAV) routing system. *Journal of Operational Research Society*, 56:776–786, 2005.

V. Kunchev, L. Jain, V. Ivancevic, and A. Finn. Path planning and obstacle avoidance for autonomous mobile robots: A review. *Knowledge-Based Intelligent Information and Engineering Systems, Pt 2, Proceedings*, 4252:537–544, 2006.

R. C. Larson and V. O. Li. Finding minimum rectilinear distance paths in the presence of barriers. *Networks*, 11:285–304, 1981.

R. C. Larson and G. Sadiq. Facility locations with the manhattan metric in the presence of barriers to travel. *Operations Research*, 31(4):652–669, 1983.

J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 2001.

D. Lee and F. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14: 393–410, 1984.

D.-T. Lee. *Proximity and reachability in the plane.* PhD thesis, Champaign, IL, USA, 1978.

L. Lewis and I. Ross. A pseudospectral method for real-time motion planning and obstacle avoidance, 2007.

J. S. Mitchell. Geometric shortest paths and network optimization, 1998.

J. S. B. Mitchell. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry and Applications*, 6(3):309–332, 1996.

J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *Siam Journal on Computing*, 16(4):647–668, 1987.

Movable Type Scripts. http://www.movable-type.co.uk/scripts/latlong.html.

F. Mufalli, R. Batta, and R. Nagi. Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans, 2010.

C. Murray. *Dynamic Reassignment and Rerouting in Cooperative Airborne Operations*. PhD thesis, University at Buffalo, State University of New York, 2010.

C. Murray and M. Karwan. An extensible modeling framework for dynamic reassignment and rerouting in cooperative airborne operations. *To appear in Naval Research Logistics.*, 2010.

J. O. Royset, W. M. Carlyle, and R. K. Wood. Routing military aircraft with a constrained shortest-path algorithm. *Military Operations Research*, 14(3):31–52, 2009.

A. Sarkar, R. Batta, and R. Nagi. Commentary on facility location in the presence of congested regions with the rectilinear distance metric. *Socio-Economic Planning Sciences*, 38:291–306, 2004.

A. Sarkar, R. Batta, and R. Nagi. Finding rectilinear least cost paths in the presence of convex polygonal congested regions. *Computers & Operations Research*, 36:737–754, 2009.

C. Schumacher, P. Chandler, M. Pachter, and L. Pachter. Optimization of air vehicles operations using mixed-integer linear programming. *Journal of the Operational Research Society*, 58(4):516–527, 2007.

V. Shetty, M. Sudit, and R. Nagi. Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles. *Computers & Operations Research*, 35:1813–1828, 2008.

T. Shima, S. Rasmussen, A. Sparks, and K. Passino. Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. *Computers & Operations Research*, 33:3252–3269, 2006.

A. M. Shkel and V. Lumelsky. Classification of the dubins set. *Robotics and Autonomous Systems*, 34: 179–202, 2001.

J. A. Storer and J. H. Reif. Shortest paths in the plane with polygonal obstacles. *Journal of the Acm*, 41 (5):982–1012, 1994.

M. Tavana, M. Bailey, and T. Busch. A multi-criteria vehicle-target allocation assessment model for network-centric joint air operations. *International Journal of Operational Research*, 3(3):235–254, 2008.

USAF. The u.s. air force remotely piloted aircraft and unmanned aerial vehicle strategic vision, 2005.

A. Weinstein and C. Schumacher. UAV scheduling via the vehicle routing problem with time windows. technical report afrl-va-wp-tp-2007-306. Technical report, Air Force Research Laboratory, 2007.

E. W. Weisstein. Circle-line intersection. from mathworld–a wolfram web resource . http://mathworld.wolfram.com/Circle-LineIntersection.html.

E. Welzl. Constructing the visibility graph for n-line segments in O(N^2) time. *Information Processing Letters*, 20(4):167–171, 1985.

J. Wilde, D. Dibiaso, and M. Nervegna. Team planning for unmanned vehicles in the risk-aware mixed-initiative dynamic replanning system. In *Oceans 2007*, pages 1–8, 2007.

M. Zabarankin, S. Uryasev, and R. Murphey. Aircraft routing under the risk of detection. *Naval Research Logistics*, 53(8):728–747, 2006.