Received July 18, 2021, accepted September 2, 2021, date of publication October 8, 2021, date of current version October 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3118954

Interference Identification for Time-Varying Polyhedra

ADAM BIENKOWSKI[1], (Member, IEEE), DAVID SIDOTI[2], AND KRISHNA R. PATTIPATI[1], (Life Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269, USA
[2]Naval Research Laboratory, Monterey, CA 93943, USA

Corresponding author: Adam Bienkowski (adam.bienkowski@uconn.edu)

This work was supported in part by the University of Connecticut (UConn) through the U.S. Office of Naval Research; and in part by the U.S. Naval Research Laboratory under Grant N00014-18-1-1238, Grant N00014-21-1-2187, and Grant N00173-16-1-G905.

ABSTRACT Identification of when and where moving areas intersect is an important problem in maritime operations and air traffic control. This problem can become particularly complicated when considering large numbers of objects, and when taking into account the curvature of the earth. In this paper, we present an approach to conflict identification as a series of stages where the earlier stages are fast, but may result in a false detection of a conflict. These early stages are used to reduce the number of potential conflict pairs for the later stages, which are slower, but more precise. Our approach is generally applicable to objects moving in piece-wise straight lines on a 2D plane, and we present a specific case where the Mercator Projection is used to transform objects moving along rhumb lines on the earth into straight lines to fit in our approach. We present several examples to demonstrate our methods, as well as to quantify the empirical time complexity by using randomly generated areas.

INDEX TERMS Water space planning, collision avoidance, collision detection, interference identification, nonlinear programming, R-trees, decision support system.

I. INTRODUCTION

Identifying conflicting operating areas on the ocean or in the air is a complex and important problem in both maritime operations and air traffic control. With thousands of ships and aircraft travelling around the world every day, it is vital to ensure that these vehicles do not operate too close to each other, risking a collision. In the maritime domain, in 2009, the USS Hartford and the USS New Orleans collided in the strait of Hormuz, needing $120 million worth of repairs and resulting in a loss of 21 months of operations [18]. The USS Hartford, a submarine, was unaware of the USS New Orleans when the vessels collided as the USS Hartford ascended to the surface. The Federal Aviation Administration listed midair collisions as the eighth leading cause of fatal aviation accidents between 2001 and 2016 [7]. When planning routes in these domains, a tool to rapidly identify spatio-temporal conflicts can guarantee safe maritime and aircraft operations.

One of the challenges in identifying conflicts for aircraft and ships is that the assets are not traveling in a straight line in the Euclidean space. The curvature of the earth must be taken into account for accurate conflict identification. In our approach, we assume that the assets are travelling along rhumb lines, which are lines with a constant angle relative to the north pole. We then use the Mercator Projection to transform the space into a 2D plane where rhumb lines result in straight lines, and linear methods can be used. This approach could be used for other transformations as well. For example, for tracks that follow the shortest path between two points on the surface of the earth, i.e., great circle tracks, the gnomonic projection would result in a 2D plane where great circles are straight lines.

The associate editor coordinating the review of this manuscript and approving it for publication was Haluk Eren.

II. RELATED RESEARCH

There has been considerable work done on interference identification in the area of air traffic control. Much of this research focuses on conflict resolution, or finding a route that eliminates conflicts, but detection of conflicts is a prerequisite to solve the conflict resolution problem. Chiang *et al.* [5] presented a method to detect conflicts and use this information to route aircraft without conflicts using Delaunay and Voronoi diagrams. Wollkind *et al.* [23] propose a distributed multi-agent negotiation technique for resolving air traffic conflicts. Alligier *et al.* [2] investigated the problem of conflict detection with uncertain trajectories. Jardin [15] uses a grid-based approach to determine if multiple aircraft occupy the same discrete location simultaneously. There has also been work done in detecting when collisions will happen between two ships for the purposes of autonomous ship routing [16], [21].

A common approach to determine if two regions overlap involves a two-phase identification process [19]. The first phase, called the broad phase, uses bounding boxes to quickly identify potentially conflicting pairs of polyhedra. However, broad phase methods can result in false alarms, that is, an interference is deemed to exist when, in fact, none does. The second phase, called the narrow phase, invokes pairwise collision detection algorithms that determine whether interference indeed exists between the pairs identified by the broad phase.

A. BROAD PHASE

The computational efficiency of broad phase methods stems from the use of bounding boxes. For example, the method of axis-aligned bounding boxes (AABBs) draws a rectangular prism that completely contains a region. If the projections of the bounding boxes onto the global axis overlap, then the regions may overlap; otherwise, the regions are deemed disjoint. Oriented bounding boxes (OBBs) provide a tighter fit than AABBs, since the bounding volume is oriented to minimize the total volume [13]. The AABB and OBB volumes can be embedded as hierarchical tree structures to represent assigned regions. Examples of hierarchical tree data structures include octrees [14], k-d trees [3], bucket trees [10], and R-trees [17]. Octrees recursively divide a cube into eight octants, while quadtrees [9], the 2-dimensional analogs, divide a plane into quadrants. An octree is formed with child elements that are the octants generated by recursively splitting the parent elements. Each child element contains an indicator of whether a region is contained in the child element or not. The k-d tree is a generalization of the octree, where instead of binary splits, as used in generating the octree, the volume is split into k sub-trees. The analysis of k-d trees applies to octrees and quadtrees. The resolution (i.e., size of the space partitions) of the k-d tree represents a trade-off between the false positive rate and the query time. A finer resolution results in fewer false positives at the expense of an increased query time. Therefore, the choice of resolution is critical to the success of the k-d trees and the related space partitioning methods (octrees, quadtrees, etc).

B. NARROW PHASE

Examples of existing narrow phase algorithms for convex polygon and polytope regions include GJK (Gilbert, Johnson, Keerthi), Voronoi Marching (Closest Feature Pair), and linear programming algorithms. The GJK algorithm successively minimizes the distance between two convex polygons, or more generally, polytopes [4], [11]. Voronoi Marching finds the closest feature pair, where features are Voronoi partitions of the exterior space of a convex polygon (polytope) [8]. A linear program [1] is solvable by an interior point method to detect intersections of polytopes. The GJK algorithm is a simplex-based method, and thus inferior to an interior point method for polygons (polytopes) with a large number of vertices, since the latter can handle models with a large number of vertices faster. The Voronoi Marching method has robustness issues and can exhibit cycling [20]. If the polygons are on the surface of the Earth, the line between two points along the surface of the Earth is not uniquely defined. We will consider the case in which both the edges of the polygon, as well as the movement of the polygon are described as rhumb lines, a line with constant angle, or bearing, relative to either the true or magnetic north pole. In this case, the problem can no longer be solved as a linear program.

TABLE 1. Summary of notation.

$\underline{p}^i(k,t)$	Vertex k of polygon i at time t
N_i	Number of vertices of polygon i
t_0^i, t_f^i	Initial and final times of polygon i
x, y	x and y coordinates a point
$\underline{c}^i(t)$	Reference point of polygon i at time t
L^i, R^i, B^i, T^i	Left, right, bottom, and top bounds of polygon i
Γ^i	Convex hull of polygon i
A^{ij}	Intersection of convex hulls of polygons i and j
$a^{ij}(k)$	Vertex k of intersection of convex hulls of polygons i and j
N_a^{ij}	Number of vertices of intersection of convex hulls of polygons i and j
$\rho^{ij}(k)$	Projection of $a_ij(k)$ onto the line that the reference point travels
$\rho_{min}^{ij}, \rho_{max}^{ij}$	Minimum and maximum values of $\rho^{ij}(k)$ over k
ρ_f^i, ρ_b^i	Projection of the front and back vertices of polygon i onto the line that the reference point travels
t_{start}, t_{end}	Start and end time of conflict
α_k, β_k	Coefficients for convex combination of polygon vertices
v^i	Velocity of vertices for polygon i in the constant velocity case
$\underline{p}_s^i(k,t)$	Vertex k of polygon i at time t in spherical coordinates (i.e., Latitude, Longitude)
λ, ϕ	Longitude, latitude coordinates of a point
$\underline{c}_s^i(t)$	Reference point of polygon i at time t in spherical coordinates (i.e., Latitude, Longitude)
s^i, θ^i	Speed and bearing of reference point of polygon i
R	Radius of Earth
$\psi_m^i(k), d_m^i(k)$	Bearing and distance step m to get to vertex k of polygon i from reference point
$\underline{q}_m$	Point along the reference point line corresponding to a projection fraction ρ in Mercator coordinates
$\underline{q}_s$	Point along the reference point line corresponding to a projection fraction ρ in spherical coordinates
$\Delta\psi$	Projected latitude difference for rhumb line distance calculation
w	Longitude distortion factor for rhumb line distance calculation
d_f^i, d_b^i, d_s^i	Distance to the front, back, and sides of a rectangular dynamic polygon

III. PROBLEM FORMULATION

Consider N polygons each with N_i vertices with the i^{th} polygon evolving over time as

$$\underline{p}^i(k,t) = \begin{bmatrix} p_x^i(k,t) \\ p_y^i(k,t) \end{bmatrix}, \quad k \in \{1 \dots N_i\} \tag{1}$$

This includes both dynamic polygons whose vertices vary with time, as well as static polygons whose vertices do not change with time. Both static and dynamic polygons have initial times t_0^i, and final times t_f^i, upon which they appear and disappear, respectively. We assume the polygons are convex and that the edges of the polygons are linear, but we assume the vertices are moving in a straight line, with possibly *time-varying* velocity. To model more complex paths of dynamic polygons, the path can be decomposed into piece-wise straight line segments. We will also make use of a "center" reference point that does not need to be at the geometric center of the polygon, but needs to be at a constant location within the moving polygon. We define this reference point as

$$\underline{c}^i(t) = \begin{bmatrix} c_x^i(t) \\ c_y^i(t) \end{bmatrix} \tag{2}$$

We also assume that, given a known velocity and a location $\underline{c}^i(t)$, we can determine the time t when the center point is at that location. The method for doing this is discussed for a specific application in Section V.

IV. SOLUTION APPROACH

A. SOLUTION OUTLINE

Our method to find the set of intersecting objects is to use successively more complicated stages to narrow down the set of potentially conflicting objects. The earlier stages (e.g., broad phase purges undeniably non-intersecting objects) will be faster, but not as selective in removing pairs of objects from the set of potential conflicts in that there may be false alarms. Later stages will be slower, but are invoked upon a smaller, more manageable input set due to the screening work from the earlier stages. The first stage entails the use of axis-aligned bounding boxes in an R-Tree. The second stage comprises polygon intersection with a sweep-line algorithm. The third stage, if required, subsequently outputs the time of intersection via the projection of the intersection's vertices. The final and the fourth stage is required when both objects are moving, and comprises our proposed novel nonlinear program to determine the time of intersection. This overall process is shown in Figure 1, and is described in algorithm 1.

B. STAGE 1: R-TREE

The first step is to insert the Axis-Aligned Bounding Boxes (AABBs) of all objects into an R-tree data structure to query for potential conflicts. The AABB is simply the minimum and maximum longitude and latitude of the polygon. For moving polygons, a bounding box that spans the entire object's trajectory is used. For a given AABB, the left, right, bottom and top is defined as

$$L^i = \min_{k,t} p_x^i(k,t) \tag{3}$$

$$R^i = \max_{k,t} p_x^i(k,t) \tag{4}$$

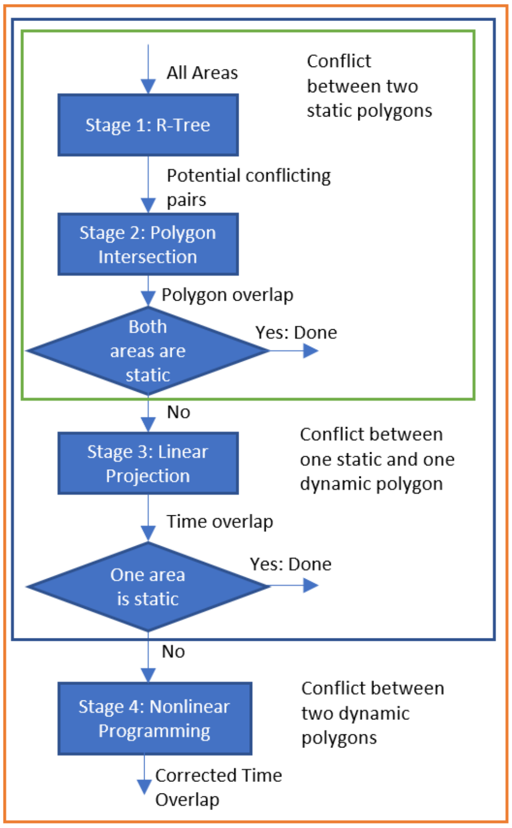

All Areas
Conflict between two static polygons
Stage 1: R-Tree
Potential conflicting pairs
Stage 2: Polygon Intersection
Polygon overlap
Both areas are static
Yes: Done
No
Stage 3: Linear Projection
Conflict between one static and one dynamic polygon
Time overlap
One area is static
Yes: Done
No
Stage 4: Nonlinear Programming
Conflict between two dynamic polygons
Corrected Time Overlap

FIGURE 1. Solution approach outline.

$$B^i = \min_{k,t} p_y^i(k,t) \tag{5}$$

$$T^i = \max_{k,t} p_y^i(k,t) \tag{6}$$

An alternative would be to use Time Parameterized R-Trees (TPR Trees) [22]; however, such structures have the limitation that the objects must be added in chronological order, which would render this step slower. More significantly, they are only valid for objects moving linearly and with constant velocity. R-trees group the AABBs of objects into larger AABBs in a balanced tree structure, which can then be quickly queried to determine if a point is within one of the AABBs. Once all the objects are added to the R-tree, each object is queried to determine which other object(s) the AABB conflicts with. The output of this step is a set of pairs of potential conflicts. Because the polygons are represented by AABBs, there is a possibility of false detection of conflicts, but there will not be any missed detections. Because each query for an object returns all the other potential conflicts with that object, this method is very fast at uncovering potential conflicts, and can be used to remove unquestionably non-conflicting objects. This is useful because the dataset that stems from waterspace planning or air traffic planning is likely to be such that there are many non-conflicting objects and a small fraction of objects with conflicts. Algorithm 2 describes this step.

Algorithm 1: Overall Solution Approach

```
Input: A: All areas

/* Stage 1, use R-Trees to find
   potential conflicts                         */
potential_conflicts = R_Tree_Check(A)
for x in potential_conflicts
    /* Stage 2, For all potential
       conflict pairs, use Shapely to
       find intersection                       */
    intersection = shapely_check(x)
    if intersection is empty
        /* No conflict, continue to next
           potential conflict                  */
    end if
    if either object in x is a track
        /* Stage 3, for tracks, use
           linear projection, using
           intersection from Shapely           */
        intersection_time =
         linear_projection(x,
         intersection)
        if intersection_time is empty
            /* No conflict, continue to
               next potential conflict         */
        end if
    end if
    if both objects are tracks
        /* Stage 4, If both objects are
           tracks, use nonlinear
           programming, using
           intersection time range
           calculated by linear
           projection                          */
        intersection_time =
         NLP(x,intersection_time)
        if intersection_time is empty
            /* No conflict, continue to
               next potential conflict         */
        end if
    end if
end for
```

Algorithm 2: Stage 1: R-Tree

```
Function R_Tree_Check(A):
    Input: A: All areas
    Output: List of pairs of potential conflicts
    /* This function adds the AABB of
       each area to an R-Tree and
       checks after each addition for
       potential conflicts with the
       latest added area                         */
    for $area_i$ in A
        /* Make Axis-Aligned Bounding
           Box for the area, as defined
           in Equations 3-6                      */
        $AABB_i$ = make_AABB($area_i$)
        ▷ Add $AABB_i$ to R-tree
        for $area_j$ in R-tree within $AABB_i$
            /* redCheck if time ranges of
               objects overlap                   */
            if $max(t_0^i, t_0^j) \leq min(t_f^i, t_f^j)$
                ▷ Add objects $i, j$ as potentially
                  conflicting
            end if
        end for
    end for
```

C. STAGE 2: AREA OF INTERSECTION

Once we have the subset of objects that are potentially conflicting, we can begin checking each pair (in parallel, if necessary) to eliminate false detections. If both areas are static polygons, this is simply checking for the intersection of the two polygons.

If one or both of the objects are dynamic polygons, this intersection operation is not adequate to definitively determine that there is a conflict, but it can provide useful information and there is no possibility of a missed detection of a conflict. For these objects, the vertices of the swept area of the track are calculated and used in the intersection algorithm. The swept area is defined as the convex hull of the polygon at the initial and final times for each piece-wise linear segment, which is subsequently aggregated to account for the entirety of the track. The convex hull of a set of points is defined as the smallest convex set containing all points. As shown in Section V, the convex hull operation is trivial for some shapes, such as rectangles. Using the swept area gives us the spatial information of where the conflict could potentially be, but not the temporal information of whether the objects are in the same place at the same time. The swept area is defined by the set operation

$$\Gamma^i = convex\_hull\left(\left\{\underline{p}^i(k, t_0^i)\forall k\right\} \cup \left\{\underline{p}^i(k, t_f^i)\forall k\right\}\right) \quad (7)$$

If the polygon is static, the convex hull operation does not need to be performed, and the set of points for the polygon can be defined as

$$\Gamma^i = \left\{\underline{x} : x = \sum_{k=1}^{N_i} \alpha_k \underline{p}^i(k), \sum_{k=1}^{N_i} \alpha_k = 1, \alpha_k >= 0\right\} \quad (8)$$

The intersecting area determined by Shapely, a package for set-theoretic analysis and manipulation of planar features [12], is given by

$$A^{ij} = \Gamma^i \cap \Gamma^j \quad (9)$$

The vertices (extreme points) of this set are defined as

$$\underline{a}^{ij}(k), k \in \{1, \ldots, N_a^{ij}\} \tag{10}$$

To calculate the intersection of two polygons, we used the Python library Shapely, which is based on the C++ library GEOS and the Java Topology Suite (JTS). The algorithm used by Shapely is described in Chapter 2.3 of [6] This algorithm will provide the vertices of the intersection of two polygons. If this intersection is empty, there is no conflict. If it is not empty, we not only learn that there is definitely a conflict, but we also obtain the additional spatial information of where the overlapping area is. If both polygons are static, this is the final step for determining if a conflict exists. If one or both of the conflicts are dynamic polygons, further computational steps are needed to ensure that a conflict does in fact exists, and the information from the Shapely analysis is used in the next stage.

D. STAGE 3: PROJECTED TIME OF INTERSECTION

At this point in our proposed multistage approach, all remaining pairs of polygons to be checked contain at least one dynamic polygon, and at some point in space, the paths of the polygons intersect. What remains to be determined is whether there is still a collision when time is taken into account. To do this, we can project the vertices of the conflicting area that were calculated in the previous stage onto the line between the reference point of the polygon at the start and end times.

$$\rho^{ij}(k) = \frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{a}^{ij}(k) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2} \tag{11}$$

If $\rho^{ij}(k)$ is 0, it means that the vertex was projected to the reference point at the start time. On the other hand, if $\rho^{ij}(k)$ is 1, it means the vertex was projected to the reference point at the end time. We are interested in the first time of conflict and the last time of conflict, so we only need the largest and smallest values of $\rho^{ij}(k)$, i.e., ρ^{ij}_{max} and ρ^{ij}_{min}. In order to determine the time of conflict, we use the distance from the reference point to the vertex that is furthest forward and backward in the direction of travel, defined as

$$\rho_f^i = \max_k \left[\frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{p}^i(k, t_0^i) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2}\right] \tag{12}$$

$$\rho_b^i = \min_k \left[\frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{p}^i(k, t_0^i) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2}\right] \tag{13}$$

If $\rho_b^i \leq \rho_{min}^{ij} \leq \rho_f^i$ then the conflict is within the polygon at time t_0^i, meaning this is our first time of conflict. If $1 - \rho_b^i \leq \rho_{max}^{ij} \leq 1 + \rho_f^i$, then the conflict is present at time t_f^i and this is our final time of conflict. If these conditions are not true, then the fractional position of the reference point can be found by $\rho_{max}^{ij} - \rho_f^i$ for the final time of conflict and $\rho_{min}^{ij} + \rho_b^i$ for the first time of conflict. From these values, the reference point can be determined, and from that, the time of conflict. These last steps are application specific, and are discussed for a specific application in Section V. Algorithm 3 and Algorithm 4 in Section V describe this step. If both polygons in a potential conflict are dynamic, then the times calculated from this section are not necessarily accurate, because the polygons need not be present at the conflict vertices $\underline{a}^{ij}(k)$ at the projection times. In this case, however, the times can be calculated by projecting the conflict vertices onto the tracks of both polygons, and the most restrictive times of conflict can be used as limits, i.e. the true time of conflict must be within that range. If, in this step, one polygon is static and the other is dynamic, then the times calculated will be the true range of the time of conflict, and no further computational steps

Algorithm 3: Stage 3: Linear Projection

```
Function linear_projection(i,j,intersection):
    Input: i,j: indices of potential conflict pair
    intersection: intersection of potential conflicting
    areas, calculated by shapely, consisting of vertices
    $\underline{a}^{ij}(k)$
    Output: $t_{start}$, $t_{end}$: start and end time of conflict,
            or none if no conflict exists
    /* This function uses linear
       projection to calculate the
       earliest and latest times of
       intersection between two objects
       */
    for each vertex $\underline{a}^{ij}(k)$ in intersection
        $\rho^{ij}(k) = \frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{a}^{ij}(k) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2}$
        $\rho_f^i = \max_k \left[\frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{p}^i(k, t_0^i) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2}\right]$
        $\rho_b^i = \min_k \left[\frac{(\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i))^T(\underline{p}^i(k, t_0^i) - \underline{c}^i(t_0^i))}{\left\|\underline{c}^i(t_f^i) - \underline{c}^i(t_0^i)\right\|_2^2}\right]$
    end for
    if $\rho_b^i \leq \rho_{min}^{ij} \leq \rho_f^i$ then
        $t_{start} = t_0^i$
    else
        $t_{start} = track\_time(\rho_{min}^{ij} + \rho_b^i)$
    end if
    if $1 - \rho_b^i \leq \rho_{max}^{ij} \leq 1 + \rho_f^i$ then
        $t_{end} = t_f^i$
    else
        $t_{end} = track\_time(\rho_{max}^{ij} - \rho_f^i)$
    end if
```

are needed. If, for a different application, it is not possible to determine the time of conflict from the reference point, this linear projection step may be skipped and the step described in the next section could be used for both cases, where both polygons are dynamic or where one is static and the other is dynamic.

E. STAGE 4: TIME OF INTERSECTION VIA NONLINEAR PROGRAMMING

The next step is performed only if both polygons in a potential conflicting pair are dynamic. In this case, we use nonlinear programming to find the minimum and maximum times that a convex combination of the vertices of each polygon can be found to be at the same point. This problem has a linear cost function, but nonlinear constraints, both because the location of the vertices varies nonlinearly with time, and because there is a cross-term between the convex coefficients and the time-varying vertices. We used the Sequential Least Squares Programming method in the Scipy package in python to solve this nonlinear program.

$$\min_{\underline{\alpha},\underline{\beta},t} \; t \tag{14}$$

$$s.t. \sum_{k=1}^{N_i} \alpha_k \underline{p}^i(k,t) = \sum_{k=1}^{N_j} \beta_k \underline{p}^j(k,t) \tag{15}$$

$$\sum_{k=1}^{N_i} \alpha_k = 1 \tag{16}$$

$$\sum_{k=1}^{N_j} \beta_k = 1 \tag{17}$$

$$\alpha_k, \beta_k \geq 0 \; \forall k \tag{18}$$

$$t_{min} \leq t \leq t_{max} \tag{19}$$

The limits on time are taken from the conflicting time range calculated in stage 3. It is important to note that if the vertices are moving at a constant velocity, then this problem becomes a linear programming problem, by replacing equation (15) with

$$\sum_{k=1}^{N_i} \alpha_k \underline{p}^i(k,t_0^i) + \alpha_k v^i t = \sum_{k=1}^{N_j} \beta_k \underline{p}^j(k,t_0^j) + \beta_k v^j t \tag{20}$$

where v^i and v^j are the velocities of the vertices.

V. APPLICATION TO RHUMB LINE-BASED PROBLEM

In the previous section, we assumed the edges of the polygons are straight lines. Because the polygons we are considering are on the surface of the Earth, the line between two points along the surface of the Earth is not uniquely defined. We will consider the case in which both the edges of the polygon, as well as the movement of the polygon are described as rhumb lines. A rhumb line is defined as a line with constant angle, or bearing, relative to either the true or magnetic North Pole. Considering a point moving at a constant velocity, at a constant bearing, the rhumb line is described by first defining our polygons in spherical coordinates, i.e., latitude and longitude, as a set of vertices with the k^{th} vertex for the i^{th} polygon defined as

$$\underline{p}_s^i(k,t) = \begin{bmatrix} p_\lambda^i(k,t) \\ p_\phi^i(k,t) \end{bmatrix} \tag{21}$$

where the λ component represents the longitude, and the ϕ component represents the latitude, and t represents the time. The rhumb lines between these points are not straight lines, so we cannot use these points in our solution approach as they are; instead, we need to do a transformation first.

One important feature to note about rhumb lines is that if two points are moving along a rhumb line with the same bearing, but different starting points, the distance between the points will vary. This means that in order for a dynamic polygon to maintain edges of the same length, the vertices must be defined using rhumb line's translations from the polygon's reference point. We define the reference point as

$$\underline{c}_s^i(t) = \begin{bmatrix} c_\lambda^i(t) \\ c_\phi^i(t) \end{bmatrix} \tag{22}$$

Rhumb line motion over time is described by the following:

$$c_\phi^i(t) = c_\phi^i(t_0^i) + \frac{s^i(t-t_0^i)}{R}\cos(\theta^i) \tag{23}$$

$$c_\lambda^i(t) = c_\lambda^i(t_0^i) + \tan(\theta^i) \ln \left[\frac{\tan\left(\frac{\pi}{4} + \frac{1}{2}c_\phi^i(t)\right)}{\tan\left(\frac{\pi}{4} + \frac{1}{2}c_\phi^i(t_0^i)\right)} \right] \tag{24}$$

where s^i and θ^i are the speed in knots (nautical miles per hour) and bearing in degrees of object i, and R is the radius of Earth in nautical miles. In order for these equations to describe a line, we use the Mercator projection, which is a common method of projecting the surface of the earth onto a plane. The key useful property of the Mercator projection for our purposes is that rhumb lines are straight lines in the Mercator plane. The Mercator transformation is given as

$$c_x^i(t) = c_\lambda^i(t) \tag{25}$$

$$c_y^i(t) = \ln\left[\tan\left(\frac{\pi}{4} + \frac{1}{2}c_\phi^i(t)\right)\right] \tag{26}$$

We will refer to the Mercator transformed reference point vector as $\underline{c}_m^i(t)$. We can then use equations (23) and (24) to get the Mercator coordinates over time as shown in equations (27) and (28), at the bottom of the next page.

The vertices of the polygon can be defined by a set of steps to take from the reference point to each vertex. These M steps are defined by bearings $\psi_m^i(k)$ and distances $d_m^i(k)$ and the vertices can be found by repeatedly applying equations (23) and (24) as shown in equations (29) and (30), at the bottom of the next page.

Also important to note is that rhumb line translations do not commute, that is, finding the front of the polygon first, then moving to the left or to the right of the direction of travel

TABLE 2. Distances and bearings for rhumb line rectangle. Each pair represents $(d_m^i(k), \psi_m^i(k))$.

		m	
		1	2
k	1	(d_f^i, θ^i)	$(d_s^i, \theta^i + \frac{\pi}{2})$
	2	(d_f^i, θ^i)	$(d_s^i, \theta^i - \frac{\pi}{2})$
	3	$(d_b^i, \theta^i + \pi)$	$(d_s^i, \theta^i + \frac{\pi}{2})$
	4	$(d_b^i, \theta^i + \pi)$	$(d_s^i, \theta^i - \frac{\pi}{2})$

gives a different point than moving to the left or right first, then moving along the direction of travel. For our application, we will define our polygons as rectangles, and define our steps as in Table 2

In Section IV-C, we needed to find the convex hull of the polygons at the initial and final times. For a rectangle moving along a track perpendicular to two of the sides, this is straightforward. In this case, we can simply take the front vertices at the end time and the back vertices at the start time, that is, $\underline{p}^i(1, t_f^i)$, $\underline{p}^i(2, t_f^i)$, $\underline{p}^i(3, t_0^i)$, $\underline{p}^i(4, t_0^i)$. In Section IV-D, we needed to find the time that a dynamic object would be at a point given the location along the track. In this particular case, as shown in Algorithm 4, this can be done by inverting the Mercator projection of the point in question, calculating the distance along the rhumb line from $\underline{c}_s^i(t_0^i)$, and using the fact that in spherical coordinates, the object is travelling at a constant speed along the rhumb line.

VI. RESULTS

In this section, we illustrate our method with several example problems, and then present timing results using randomly generated sets of shapes.

A. EXAMPLE 1: PAIR OF STATIC POLYGONS

The first example involves two static polygons shown in Figure 2. The first step is to check the AABBs of the two polygons using an R-tree (Section IV-B). Because the blue and green dashed lines overlap in Figure 2, this step will determine that there is a potential conflict. The next step is to find the area of intersection with Shapely (Section IV-C). This step will give us the red shaded region in Figure 2, identifying the conflicting area. Since this conflicting area exists, we have determined that there is in fact a conflict, and no further steps are needed.

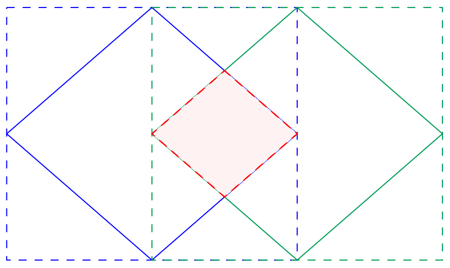


FIGURE 2. Example with two static polygons. Solid lines indicate polygon objects, dashed blue and green lines indicate AABB for the polygons, dashed red line indicates the identified conflicting area.

B. EXAMPLE 2: A STATIC POLYGON AND A DYNAMIC TRACK

This example consists of a static polygon and a dynamic polygon, shown in Figure 3. The parameters for this example are shown in Table 3.

In this example, the AABBs are not shown, but the first step will be the same as in the previous example. The next step will be to use Shapely to find the area of intersection between the static polygon and the convex hull of the dynamic polygon at the start and end times. Once this is found, the vertices of the area of intersection are projected onto the reference point track, resulting in the dashed red lines in Figure 3 (Section IV-D). In this case, the time of intersection is determined to be between 4.04 and 10.58 hours after the start time of the dynamic polygon.

C. EXAMPLE 3: TWO DYNAMIC TRACKS

In this example, there are two dynamic tracks, shown in Figure 4. The parameters for the blue track are the same as in the previous example, shown in Table 3, the parameters for the green track are shown in Table 4. The first three steps are the same as in the previous examples, resulting in the dashed red and black lines in Figure 4. In this case, the projection of the intersection vertices results in two sets

$$c_x^i(t) = c_x^i(t_0^i) + \tan(\theta^i) \ln \left[\frac{\tan\left(\frac{\pi}{4} + \frac{1}{2}(c_\phi^i(t_0^i) + \frac{s^i(t-t_0^i)}{R} \cos(\theta^i))\right)}{\tan\left(\frac{\pi}{4} + \frac{1}{2} c_\phi^i(t_0^i)\right)} \right] \tag{27}$$

$$c_y^i(t) = \ln \left[\tan\left(\frac{\pi}{4} + \frac{1}{2} \left(c_\phi^i(t_0^i) + \frac{s^i(t-t_0^i)}{R} \cos(\theta^i) \right) \right) \right] \tag{28}$$

$$p_x^i(k,t) = c_x^i(t_0^i) + \sum_{q=1}^{M} \tan(\psi_q(k)) \ln \left[\frac{\tan\left(\frac{\pi}{4} + \frac{1}{2}(c_\phi^i(t) + \sum_{r=1}^{q} \frac{d_r(k)}{R} \cos(\psi_r(k)))\right)}{\tan\left(\frac{\pi}{4} + \frac{1}{2}(c_\phi^i(t) + \sum_{r=1}^{q-1} \frac{d_r(k)}{R} \cos(\psi_r(k)))\right)} \right] \tag{29}$$

$$p_y^i(k,t) = \ln \left[\tan\left(\frac{\pi}{4} + \frac{1}{2} \left(c_\phi^i(t) + \sum_{q=1}^{M} \frac{d_q^i(k)}{R} \cos(\psi_q^i(k)) \right) \right) \right] \tag{30}$$

Algorithm 4: Finding Time From Projection Fraction

```
Function track_time(ρ):
    Input: rho: Fraction of the track to conflict point
    Output: Time at which the reference point of the
            area is at the point described by rho
    /* This function calculates the
       time at which the moving haven
       reference point is at a given
       fraction along a track              */
    /* First, find the point along the
       Mercator line corresponding to
       the fraction                        */
        $\underline{q}_m = \rho(\underline{c}^i_m(t^i_f) - \underline{c}^i_m(t^i_0)) + \underline{c}^i_m(t^i_0)$
    /* Next, invert the Mercator
       projection                          */
        $\underline{q}_s = \begin{bmatrix} q_x \\ 2\arctan(\tanh(\frac{q_y}{2})) \end{bmatrix}$
    /* Calculate the distance along a
       rhumb line to the projected
       point                               */
        $\Delta\psi = \ln\left[\frac{\tan\left(\frac{\pi}{4} + \frac{q_\phi}{2}\right)}{\tan\left(\frac{\pi}{4} + \frac{c^i_\phi(t^i_0)}{2}\right)}\right]$
    if $\Delta\psi$ is close to 0 then
        /* This means the track is
           East/West, so the latitude
           remains the same                */
            $w = \cos q_\phi$
    else
            $w = \frac{(q_\phi - c^i_\phi(t^i_0))}{\Delta\psi}$
    end if
        $d = \sqrt{(q_\phi - c^i_\phi(t^i_0))^2 + w^2(q_\lambda - c^i_\lambda(t^i_0))^2}$
        $t = d/s^i$
```

of times. Projecting onto the blue track gives a range of 3.16 to 10.58 hours, projecting onto the green line gives 1.33 to 6.95 hours. This leads us to use 3.16 to 6.95 hours as our limits on time when solving our nonlinear programming problem (Section IV-E). Solving the nonlinear programming problem results in an exact time of conflict between 4.12 hours and 4.63 hours.

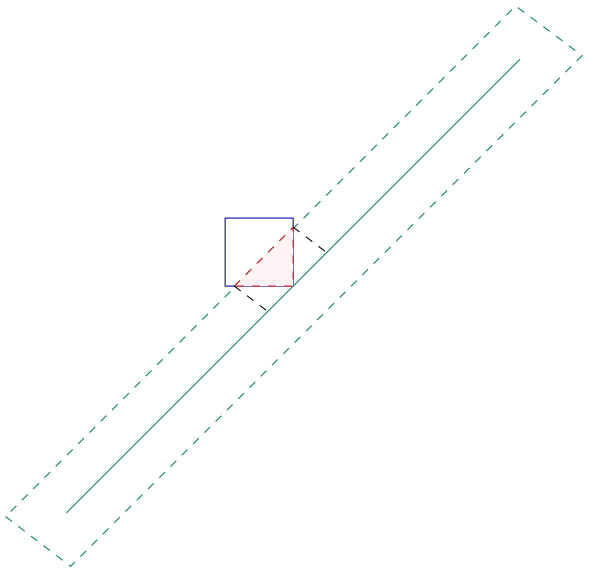


FIGURE 3. Example with one static and one dynamic polygon. Blue is static polygon, green line indicates the track of the dynamic polygon, green dashed line indicates the convex hull of the dynamic polygon at the start and end times, dashed red shaded area indicates the area of intersection, black dashed lines indicate projection of the intersection vertices.

TABLE 3. Parameters for Example 2.

$c^i_\lambda(t^i_0)$	39 deg
$c^i_\phi(t^i_0)$	29 deg
$c^i_\lambda(t^i_f)$	41 deg
$c^i_\phi(t^i_f)$	31 deg
d^i_f	10 nautical miles
d^i_b	10 nautical miles
d^i_s	10 nautical miles
s^i	15 knots

TABLE 4. Parameters for Example 2.

$c^i_\lambda(t^j_0)$	39 deg
$c^i_\phi(t^j_0)$	30 deg
$c^i_\lambda(t^j_f)$	41 deg
$c^i_\phi(t^j_f)$	30 deg
d^j_f	10 nautical miles
d^j_b	10 nautical miles
d^j_s	10 nautical miles
s^i	15 knots

D. COMPUTATIONAL COMPLEXITY ANALYSIS

In order to test the runtime of our approach, we generated sets of random areas and tracks. All areas were generated within a box with latitudes between 0 and 30 degrees, and longitudes between −45 and −15 degrees. Areas were generated with 3 to 10 vertices, by generating 3 to 10 angles between 0 and 2π and for each angle generating a radius between 50 and 100 nautical miles from a randomly generated center. Tracks were generated by randomly placing the first waypoint, and placing a second point between 50 and 100 nautical miles away, with the dimensions of the rectangle varying between 20 and 40 nautical miles on a side.

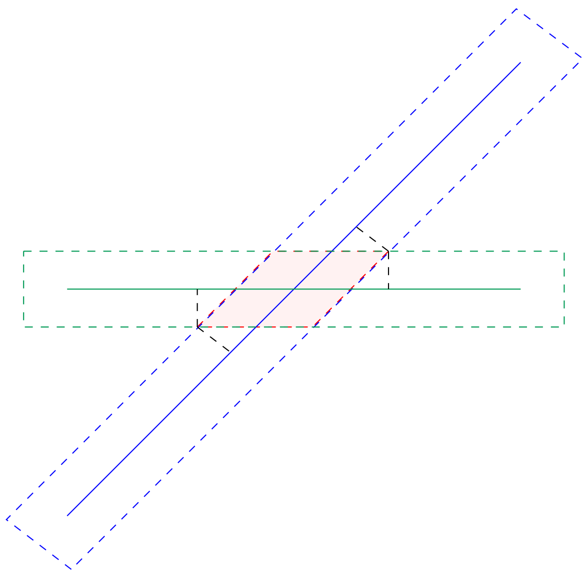


FIGURE 4. Example with two dynamic polygons. Solid lines indicate tracks of the polygons, dashed blue and green lines indicate the convex hull of the polygons at the start and end times, red shaded area indicates the area of intersection, dashed black lines indicate the projection of intersection vertices.

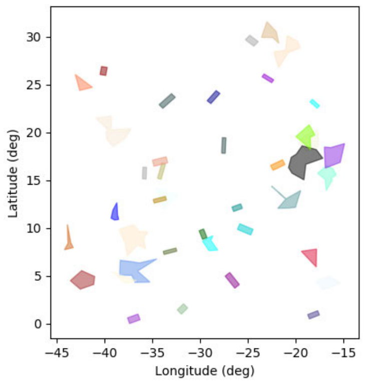


FIGURE 5. Example of a set of randomly generated areas.

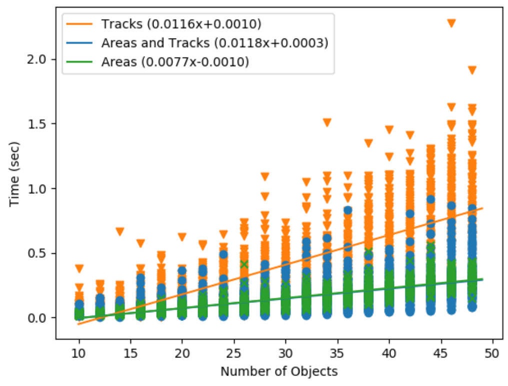


FIGURE 6. Total runtime as a function of the number of objects.

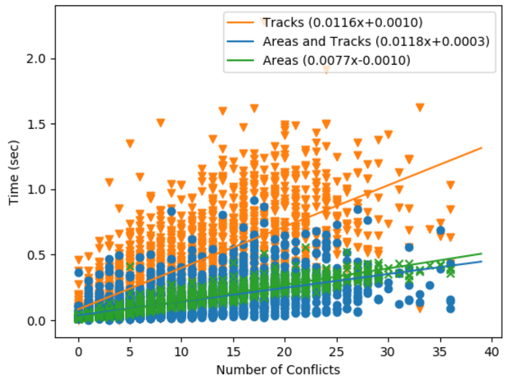


FIGURE 7. Total runtime as a funciton of the number of conflicts.

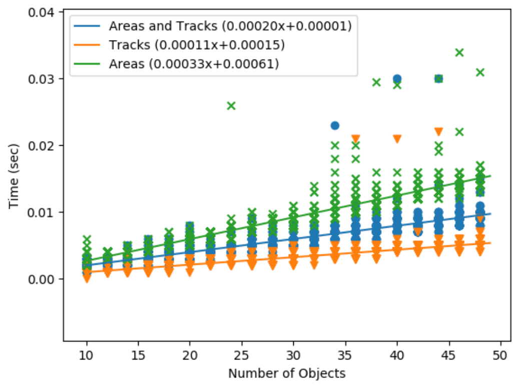


FIGURE 8. Total broad phase runtime as a function of the number of objects.

We generated areas for 3 cases, one with only static areas, one with both areas and tracks, and one with only tracks. The tests were run on a computer with an AMD Ryzen 7 2700X processor and 32GB of RAM. An example of the set of areas generated is shown in Figure 5. The overall time spent is shown in Figures 6, which shows that there is significant variation in the amount of time taken for a given number of objects. This is because the number of conflicting objects, can vary significantly, which affects the number of times the later stages must be executed. Figure 7 shows that for cases with only static areas, the execution time is close to linear with the number of conflicts. When there are dynamic tracks involved, there is still variation because there is more possibility for false positive detections in the earlier stages, which cause the later stages, especially stage 4 to take longer. The time spent in each step was measured, and is shown in Figures 8-12. Figure 8 shows that stage 1 is very fast at eliminating potential conflicts, and scales well with the number of objects. Figure 9 shows that stage 2 is taking the most amount of time, but it is handling more conflicts, and if for example, this stage was skipped and nonlinear programming was used instead, Figure 11 and 12 shows that for the same number of potential conflicts, this would take significantly longer. This means that shapely is useful for static vs. static and static vs. dynamic conflicts, as well as for eliminating false positive conflict detections. Similarly, Figure IV-D shows that stage 3 takes little time, making it preferable for static vs. dynamic conflicts over nonlinear programming. For stage 4, the time was measured separately for the cases where the potential conflict

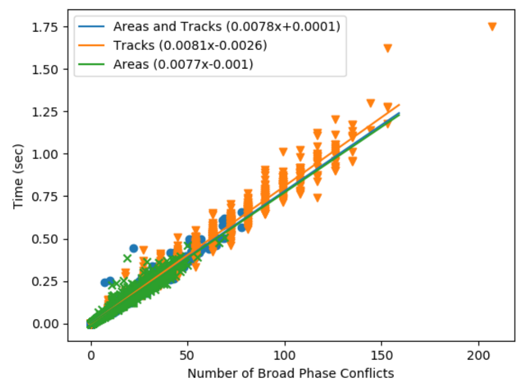


FIGURE 9. Total shapely runtime for all objects, as a function of the number of potential conflicts that resulted from broad phase.

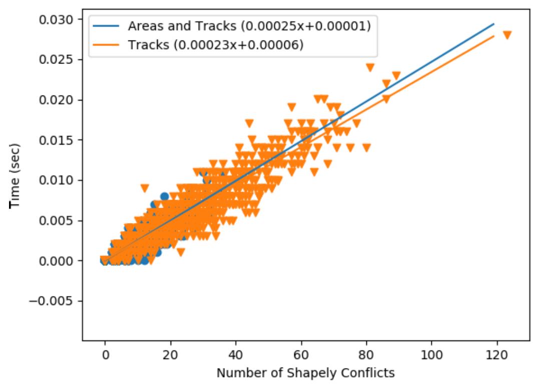


FIGURE 10. Total projection runtime for all objects, as a function of the number of potential conflicts that resulted from shapely step.

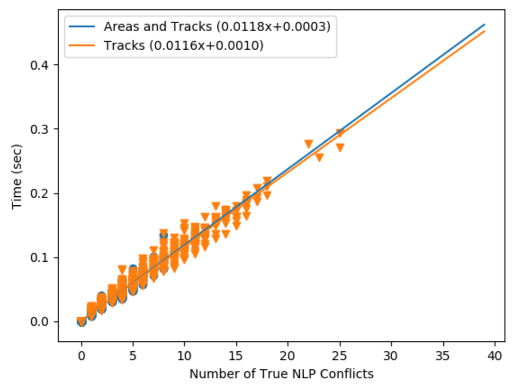


FIGURE 11. Runtime for nonlinear programming phase for all cases where a true conflict was detected, as a function of how many times this happened.

was determined to be true (Figure 11, and where it ended up being a false alarm (Figure 12. The time for false alarms was significantly longer than the time for true conflicts, which highlights the need for the previous phases to limit the number of false alarms as much as possible.

Stages 2 through 4 deal with independent pairs of potential conflicts, and therefore can be sped up using

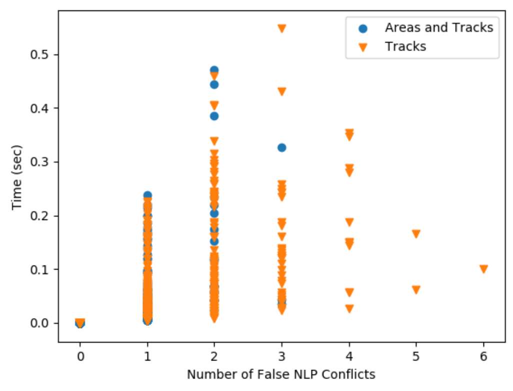


FIGURE 12. Runtime for nonlinear programming step for all cases where NLP determined that there was no conflict, as a function of how many times this happened.

parallelization. This would result in speedups particularly with very large numbers of objects. We have used our method on examples with 500-800 objects with the entire process taking 3-5 seconds.

VII. CONCLUSION

In this paper, we presented an approach to detect conflicts between large sets of static and dynamic polygons. We solved the problem in four sequential stages. First, R-trees were used to rapidly eliminate non-conflicts, but with the possibility of false detection of conflicts. Next, the area of intersection was found between two potentially conflicting polygons using the Shapely library, a package for set-theoretic analysis and manipulation of planar features. For static polygons, this is the last step needed. In the next stage, the conflict vertices from the previous stage were projected onto the track of the dynamic polygon, allowing the time of conflict to be determined. For pairs of one static and one dynamic polygon, this is the last stage needed. The last stage for finding conflicts between pairs of dynamic polygons was a nonlinear programming problem that finds the minimum and maximum time where a convex combination of the vertices of the polygons were at the same location. We demonstrated the conflict identification approach via three examples of increasing complexity, and presented computation times for randomly generated sets of polygons. In the future, we plan to develop automated methods to determine optimal areas and routes that avoid conflicts, allowing us not only to identify where there are problems with a plan, but to be able to proactively recommend ways to adjust plans by mission planners to resolve conflicts. We also plan to investigate methods to estimate/predict the locations and sizes of dynamic polygons with more complicated movements, and in the presence of noise.

REFERENCES

[1] A. Akgunduz, P. Banerjee, and S. Mehrotra, "A linear programming solution for exact collision detection," *J. Comput. Inf. Sci. Eng.*, vol. 5, no. 1, pp. 48–55, Mar. 2005.

[2] R. Alligier, N. Durand, and G. Alligier, "Efficient conflict detection for conflict resolution," in *Proc. Int. Conf. Res. Air Transp. (ICRAT)*, Jun. 2018, pp. 2–9.

[3] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[4] G. V. D. Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graph. Tools*, vol. 4, no. 2, pp. 7–25, Jan. 1999.

[5] Y.-J. Chiang, J. T. Klosowski, C. Lee, and J. S. Mitchell, "Geometric algorithms for conflict detection/resolution in air traffic management," in *Proc. 36th IEEE Conf. Decis. Control*, vol. 2, Dec. 1997, pp. 1835–1840.

[6] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer, 2008.

[7] L. Dorr, *Fact Sheet—General Aviation Safety*. Washington, DC, USA: Federal Aviation Administration, 2018.

[8] S. A. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra by multi-level Voronoi marching," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2000, pp. 2101–2106.

[9] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Inf.*, vol. 4, no. 1, pp. 1–9, 1974.

[10] F. Ganovelli, J. Dingliana, and C. O'Sullivan, "Buckettree: Improving collision detection between deformable objects," in *Proc. Spring Conf. Comput. Graph. (SCCG)*, vol. 11, 2000, p. 19.

[11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Autom.*, vol. 4, no. 2, pp. 193–203, Apr. 1988.

[12] S. Gillies *et al.*, "Shapely: Manipulation and analysis of geometric objects," Tech. Rep., 2007. [Online]. Available: https://github.com/Toblerity/Shapely/blob/master/CITATION.txt

[13] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1996, pp. 171–180.

[14] C. Jackins and S. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Comput. Graph. Image Process.*, vol. 14, no. 3, pp. 249–270, 1980.

[15] M. Jardin, "Grid-based strategic air traffic conflict detection," in *Proc. AIAA Guid., Navigat., Control Conf. Exhib.*, Aug. 2005, p. 5826.

[16] G. Li, H. P. Hildre, and H. Zhang, "Toward time-optimal trajectory planning for autonomous ship maneuvering in close-range encounters," *IEEE J. Ocean. Eng.*, vol. 45, no. 4, pp. 1219–1234, Oct. 2020.

[17] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-trees: Theory and Applications*. London, U.K.: Springer, 2010.

[18] J. McDermott, "Electric boat gets uss hartford back to sea," *Day*, Jul. 17, 2011.

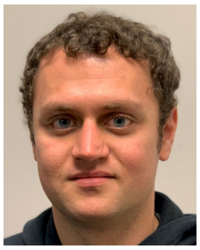
[19] B. Mirtich, "Efficient algorithms for two-phase collision detection," in *Practical Motion Planning Robotics: Current Approaches Future Directions*. New York, NY, USA: Wiley, Dec. 1997, pp. 203–223.

[20] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Trans. Graph.*, vol. 17, no. 3, pp. 177–208, Jul. 1998.

[21] J. Park and J. Kim, "Predictive evaluation of ship collision risk using the concept of probability flow," *IEEE J. Ocean. Eng.*, vol. 42, no. 4, pp. 836–845, Oct. 2017.

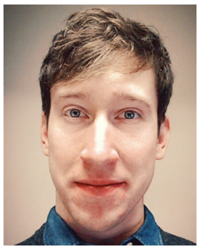
[22] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2002, pp. 334–345.

[23] S. Wollkind, J. Valasek, and T. Ioerger, "Automated conflict resolution for air traffic management using cooperative multiagent negotiation," in *Proc. AIAA Guid., Navigat., Control Conf. Exhib.*, Aug. 2004, p. 4992.



ADAM BIENKOWSKI (Member, IEEE) received the B.S. and M.Eng. degrees in electrical and computer engineering from the University of Connecticut, Storrs, CT, USA, in 2013 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, under the advisement of Dr. K. R. Pattipati.

He was an Electrical Engineer at General Dynamics Electric Boat, Groton, CT, from 2013 to 2017. His current research interests include modeling dynamic and uncertain environments for asset allocation and path planning, context aware decision support systems, and optimization and machine learning-based techniques for mission planning and coordination.



DAVID SIDOTI received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from the University of Connecticut, Storrs, CT, USA, in 2011, 2016, and 2018, respectively. He is a Computer Scientist under the Meteorological Applications Development Branch with the U.S. Naval Research Laboratory, Marine Meteorology Division (MMD), Monterey, CA, USA. He is NRLMMD's primary machine learning subject matter expert and contributes to a broad portfolio of machine learning projects relating to atmospheric and oceanographic forecasting. His current interests include multi-objective algorithms for dynamic scheduling and resource management in weather-impacted environments and deep learning applications to numerical weather prediction.

He was a co-recipient of the Tammy Blair Award for best student paper at FUSION 2016. In 2018, he was recognized and awarded a Distinguished Scholar Jerome and Isabella Karle's Fellowship.

KRISHNA R. PATTIPATI (Life Fellow, IEEE) received the B.Tech. degree (Hons.) in electrical engineering from the Indian Institute of Technology, Kharagpur, in 1975, and the M.S. and Ph.D. degrees in systems engineering from UCONN, Storrs, in 1977 and 1980, respectively. He was with ALPHATECH, Inc., Burlington, MA, USA, from 1980 to 1986. He has been with the Department of Electrical and Computer Engineering, UCONN, since 1986, where he is currently a Board of Trustees Distinguished Professor and the UTC Chair Professor of Systems Engineering. He is also a Co-Founder of Qualtech Systems, Inc., a firm specializing in advanced integrated diagnostics software tools (TEAMS, TEAMS-RT, TEAMS-RDS, TEAMATE, and PackNGo), and serves on the board of Aptima, Inc. He has published over 500 scholarly journal and conference papers in these areas. His research interests include application of systems theory, optimization and inference techniques to agile planning, anomaly detection, and diagnostics and prognostics.

He is an Elected Fellow of IEEE for his contributions to discrete-optimization algorithms for large-scale systems and team decision-making and of the Connecticut Academy of Science and Engineering. He was selected by the IEEE Systems, Man, and Cybernetics (SMC) Society as an Outstanding Young Engineer of 1984 and received the Centennial Key to the Future Award. He was a co-recipient of the Andrew P. Sage Award for the Best SMC Transactions Paper for 1999, the Barry Carlton Award for the Best AES Transactions Paper for 2000, the 2002 and 2008 NASA Space Act Awards for "A Comprehensive Toolset for Model-based Health Monitoring and Diagnosis," and "Real-time Update of Fault-Test Dependencies of Dynamic Systems: A Comprehensive Toolset for Model-Based Health Monitoring and Diagnostics," the 2005 School of Engineering Outstanding Teaching Award, and the 2003 AAUP Research Excellence Award at UCONN. He has served as the Editor-in-Chief for the IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, from 1998 to 2001.