Journal of Intelligent & Robotic Systems (2019) 95:239–250
https://doi.org/10.1007/s10846-018-0930-5

Geofence Boundary Violation Detection in 3D Using Triangle Weight Characterization with Adjacency

Mia N. Stevens[1] · **Hossein Rastgoftar**[2] · **Ella M. Atkins**[2]

Received: 10 October 2017 / Accepted: 5 September 2018 / Published online: 15 September 2018

Abstract

This paper introduces a computationally efficient geofence boundary violation detection method using the Triangle Weight Characterization with Adjacency (TWCA) algorithm. The geofence is defined as a maximum and a minimum altitude, and a horizontal boundary specified as a polygon that does not self-intersect. TWCA initialization divides the horizontal component and bounding box of each geofence into a finite set of triangles, then determines the triangle containing the vehicle. During flight, each position update is checked for containment within the vertical geofence boundaries using inequalities and the horizontal geofence boundaries using TWCA. TWCA searches for the triangle containing the vehicle position using breadth-first search of the adjacency graph. The root node of the search is the triangle occupied at the previous time step. This algorithm is applicable to three-dimensional geofences containing both keep-in (inclusion) geofences and keep-out (exclusion) geofences.

Keywords Geofence · Unmanned aerial vehicle · Unmanned aircraft system · Safety · UTM

1 Introduction

Unmanned aircraft systems (UAS) continue to proliferate and can now be operated commercially within line-of-sight through the FAA's Part 107 rules [4] and beyond-line-of-sight with Part 107.31 waiver. UAS applications range from last-mile package deliveries to agricultural and infrastructure inspection to disaster relief support. Hobbyist flight is also commonplace. A micro-air vehicle (MAV) or very small UAS (< 250 g) may pose little risk to people or property, but such a vehicle has limited range and cannot carry payload beyond a small camera. Even small UAS can pose a safety risk through fast-spinning propeller cuts and direct impact. NASA is working with industry and academic partners to develop a UAS Traffic Management (UTM) system of which a key component is electronic geofencing [7].

Geofences assign each UAS an empty flight volume in which they are authorized to operate. The geofence can also be used as a mechanism to assure a low-flying UAS only operates low over a property with landowner permission. A geofence can be classified as a keep-in (inclusion) geofence or a keep-out (exclusion) geofence. The keep-in geofence defines a bounded flight volume for the UAS, while the keep-out geofence defines general volumes to avoid as well as cut-outs within a keep-in geofence. A keep-out geofence marks a no fly zone for the UAS. Public properties such as national monuments and private properties such as a backyard pool may be protected by low-altitude keep-out geofences.

Given defined geofence boundaries, the geofencing system consists of two logic units: the detection of geofence

A short version of this paper was presented in ICUAS 2017. This work was supported in part by a subcontract from Soar Technology, Inc. under Phase II SBIR Contract No. FA8650-15-C-2629.

✉ Mia N. Stevens
minist@umich.edu

Hossein Rastgoftar
hosseinr@umich.edu

Ella M. Atkins
ematkins@umich.edu

[1] Robotics Institute, University of Michigan, Ann Arbor, MI 48109, USA

[2] Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA

violations and the response to a geofence violation. There are many possible responses to a geofence violation including but not limited to alerting the pilot, cutting the aircraft power, or an alternative guidance scheme designed to respect the geofence boundaries [17]. To prevent the vehicle from violating the geofence boundaries, the geofence system must activate before the boundary is crossed. The activation point can be represented as a stopping distance calculated based on the vehicle flight characteristics and the current wind speeds, which can be used to shift the geofence boundaries. The design of these shifted geofence boundaries is discussed in other works [3, 18], which can then be analyzed using the methods presented in this paper. This paper focuses on the detection of geofence violations through the application of the Triangle Weight Characterization with Adjacency (TWCA) algorithm [2, 8, 14, 15]. TWCA is compared to the Ray Casting algorithm [1, 5, 11], a common solution to this type of problem, as a baseline for algorithm analysis.

This paper contributes a survey of algorithms applicable to the geofence boundary violation detection problem. General geofence boundaries may be non-convex, and multiple geofences can potentially overlap causing intersecting boundaries. This paper also contributes a methodology for benchmarking geofence boundary violation detection strategies. These benchmarking techniques are applied to compare Ray Casting and TWCA. This is the first paper to our knowledge that evaluates and compares multiple geofencing boundary detection strategies.

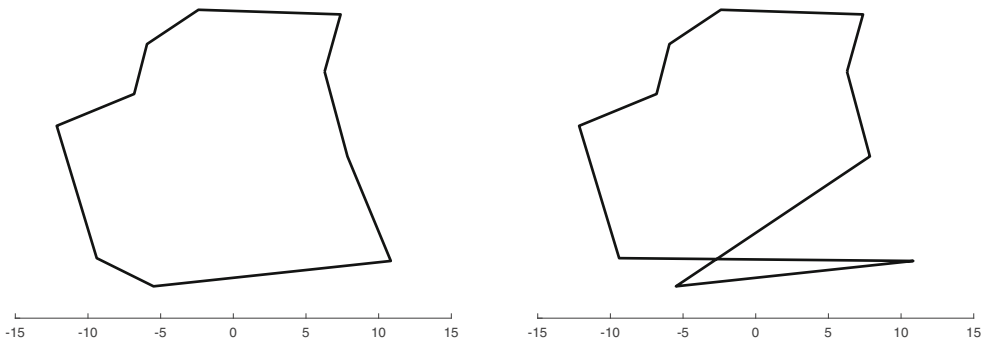
Section 2 states geofence characteristics and assumptions made in the proposed geofence violation detection framework. Section 3 discusses methods commonly applied to solve problems similar to the detection of horizontal geofence boundary violations. Section 4 presents the Triangle Weight Characterization with Adjacency (TWCA) algorithm, while Section 5 presents results of introducing randomly-generated UAS flight paths through randomly generated geofence boundaries to compare TWCA with a traditional Ray Casting approach to boundary violation detection. Section 6 discusses the application of geofencing with TWCA in real-world environments and areas for future work, followed by a brief conclusion in Section 7.

2 Problem Statement

The *static UAS geofence* proposed in this work has the following characteristics:

- A geofence consists of exactly one surrounding keep-in boundary and any number of interior keep-out boundaries.
- Geofence boundaries remain unchanged for the duration of a flight, i.e., the geofence volume is static.
- Each geofence boundary is a polyhedron with vertical and horizontal boundaries.
- The polyhedron is formed by extruding a horizontal plane polygon vertically. Geofence vertical boundaries are specified as altitude ceiling and floor above ground level (AGL) or mean sea level (MSL). Note that buildings and terrain with variable elevations can be modeled with multiple keep-out polyhedra.
- The horizontal geofence boundary is a polygon that is not self-intersecting as shown in Fig. 1. Each horizontal geofence boundary is specified as a list of vertices in a local ENU (East-North-Up) or local NED (North-East-Down) format, in clockwise or counterclockwise order around the polygon boundary.
- The vehicle is powered on, initialized, and launched from a position within the keep-in geofence polyhedron and outside all keep-out geofence polyhedra.
- The geofencing system monitors vehicle state at regular interval δt.

This work assumes the internal representations of the geofence boundaries are expressed in meters relative to a locally defined origin point, the launch point of the



(a) Acceptable horizontal geofence boundary.

(b) Unacceptable horizontal geofence boundary.

Fig. 1 Examples of valid and invalid (unacceptable) horizontal geofence boundary specifications using the same vertex list with good (left) and bad (right) orderings

aircraft. Two subcategories of geofences are utilized: keep-in (inclusion) and keep-out (exclusion). A keep-in geofence defines the volume in which the aircraft is allowed to operate. A keep-out geofence defines a volume in which the aircraft is not allowed to operate, either due to permissions or physical barriers. *A geofence violation occurs when the UAS is outside the keep-in geofence or inside a keep-out geofence.* The Fig. 2 flow chart details the procedure utilized by the geofencing system to detect geofence violations. The inputs to the system are the current UAS position $\mathbf{r} = (x, y, z)$ and the geofence $\mathbf{g}$. The geofence is defined as $\mathbf{g} = [g_i, g_o]$ where g_i is the keep-in geofence polyhedron and $g_o = \{g_{o,1}, \ldots, g_{o,n}\}$ is the set of keep-out geofence polyhedra. $g_{o,j}$ is the jth of n keep-out geofence polyhedra. The altitude limits of a geofence $\mathbf{g}$ are denoted by $z_{g_i/g_{o,j}}$. The horizontal geofence polygon vertices are denoted by coordinate pairs $(x_{g_i/g_{o,j}}, y_{g_i/g_{o,j}})$ listed in either clockwise or counter-clockwise order around the polygon.

For each vehicle state update, three checks are performed for the keep-in geofence and for each keep-out geofence. The first check determines if the vehicle is within the altitude limits of the geofence. The second check determines if the vehicle is within the bounding box of each geofence. Each bounding box is defined as a rectangle orthogonal to the global axes that contains the original horizontal geofence polygon [6]. Vehicle position inside or outside the bounding box is determined using four inequality tests. If the vehicle is outside the bounding box, then it is outside the geofence. If the vehicle is inside the bounding box, then the third check determines if the vehicle is within the horizontal geofence boundary. The third check is an application of the *point-in-polygon* problem [13].

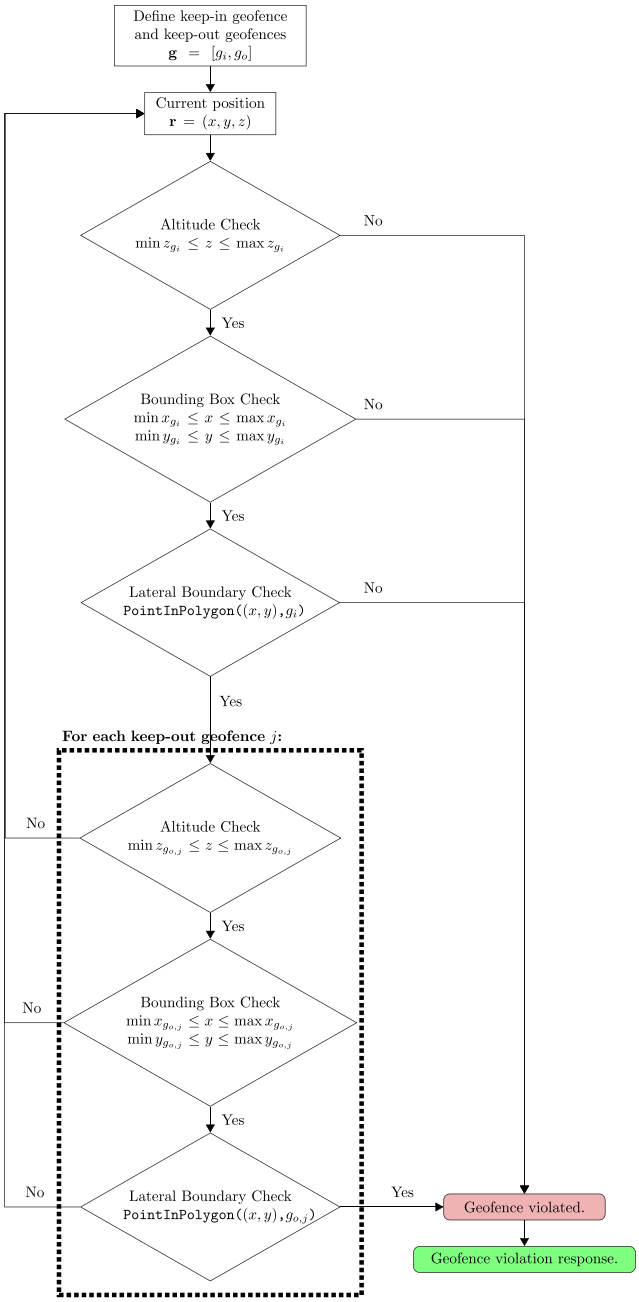


Fig. 2 Geofence violation detection algorithm for a single keep-in geofence and a known number of keep-out geofences. Options for the `PointInPolygon`$((x, y), g)$ algorithm are presented in Figs. 4 and 5. The *point-in-polygon* problem is presented in [13]

3 Horizontal Geofence Violation Detection Algorithms

A point-in-polygon algorithm can be applied to determine whether a geofence horizontal boundary is violated. The point-in-polygon problem is commonly discussed in the fields of computer graphics, computational geometry, and geographical information systems (GIS). Point-in-polygon algorithms are benchmarked based on the complexity of a single position query. Surveys and explanations of point-in-polygon solutions can be found in an article by Nordbeck and Rystedt [12], a survey by Huang and Shih [6], and a book by Preparata and Shamos [13].

3.1 Grid-Based Algorithms

There are two primary types of grid-based point-in-polygon algorithms. The first method simplifies the polygon boundaries to lie along a grid such that each grid square can be designated as inside the polygon or outside the polygon. The run-time complexity of this algorithm is linear in the number of grid squares [12, 20] . The second method overlays the polygon boundaries on a grid then analyzes each position of interest with respect to the occupied grid section. The complexity of these algorithms depends on the

size of the grid squares and the number of polygon edges [9, 19, 20].

3.2 Decomposition Algorithms

Decomposition divides the polygon into subcomponents that are less complex to simplify the point-in-polygon inclusion check. Three decomposition methods are commonly utilized: Wedge, Swath, and convex decomposition. The Wedge method, only applicable to convex polygons, divides the polygon into triangles by connecting an interior point to each of the polygon vertices. There are the same number of triangles as number of original polygon vertices. This algorithm has a run-time complexity of $O(\log N)$ [6, 13]. The Swath method divides the polygon into horizontal swaths where the maximum and minimum y-values are designated by successive polygon vertices when the vertices are ordered by y-value. Each swath contains a subset of relevant polygon edges. The step to find the swath containing a position of interest has complexity $O(\log N)$ when searched for using a balanced binary tree [16]. Convex decomposition has a run-time complexity of $O(\log N)$ [10].

3.3 Ray Casting

The Ray Casting algorithm determines whether or not the position of interest, **r**, is inside a given polygon, p, by projecting an infinite ray from **r**. In this implementation, each ray is cast in the positive y-direction (Fig. 3). If an infinite ray intersects an odd number of polygon edges, then **r** is contained in p; otherwise **r** is outside of p. If the ray intersects a vertex of p, then that intersection is tallied as $count = count + 1/2$ instead of $count = count + 1$ to prevent double counting [1].

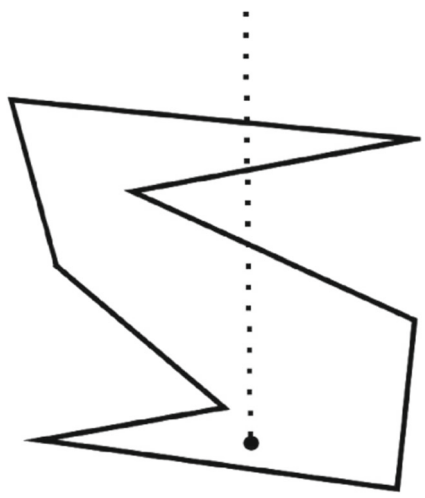


Fig. 3 Example polygon with ray directed along the y-axis for the Ray Casting algorithm [11]

An outline of the Ray Casting algorithm is shown in Algorithm 1. The algorithm is based on the formulation presented by Narkawicz and Hagen [11]. The Ray Casting algorithm iterates over all edges of p and does not have an initialization step. The complexity of the algorithm is $O(N)$, and if the geofence boundaries change from one time step to the next, code execution and results of the Ray Casting algorithm are not impacted. Ray Casting with a bounding box is used as the baseline for comparison with TWCA (Figs. 4 and 5).

4 Triangle Weight Characterization with Adjacency (TWCA)

The point-in-polygon algorithm being proposed by this paper for application in geofence systems is Triangle Weight Characterization with Adjacency (TWCA). This algorithm is closely related to the Wedge algorithm [6, 13]. Both algorithms divide the polygon into triangles then search the triangles for the one containing the position of interest. Unlike the Wedge algorithm, TWCA is designed to handle non-convex polygons and multiple trajectory-based inquiries. TWCA contains an initialization step and a run-time step as shown in Algorithm 2. The initialization step must be executed for all keep-in and keep-out geofences when the system first activates. If there are any changes to any of the geofence boundaries after the original initialization, each keep-in or keep-out geofence that is changed must be initialized again.

Algorithm 1 `PointInPolygon()` - Ray Casting

Input: **r** is the position of interest
p is a simple polygon
Output: **true** if p contains **r**, otherwise **false**

```
count = 0
s is an infinite ray in the +y direction, originating at r
for all edges e in p do
    if s intersects a vertex of e then
        count = count + 1/2
    else if s intersects e then
        count = count + 1
    end if
end for
if count is odd then
    return true
else
    return false
end if
```

Algorithm 2 `PointInPolygon()` - Triangle Weight Characterization with adjacency

Input: **r** is the position of interest
p is a simple polygon
Output: **true** if p contains **r**, otherwise **false**
`Initialization:`
Divide p into m y-monotone polygons
for all y-monotone polygons M in p **do**
  Divide polygon M into n triangles
end for
`Run-Time:`
for all N triangles in p **do**
  if N contains **r** **then**
    return true
  end if
end for
return false

For a keep-in geofence, TWCA initialization constructs a second polygon formed by the bounding box and the original geofence boundary (see Fig. 6b). Then, TWCA divides both polygons into triangles and generates an adjacency graph that spans both polygons. Prior to takeoff, the triangles are searched in random ordering to locate the vehicle. After takeoff, the triangle search for the vehicle begins at the previously-occupied triangle, and searches the adjacency graph in a breadth-first search with a visited list. A geofence boundary is violated when the vehicle is outside the bounding box or inside a triangle located between the geofence and the bounding box.

4.1 Bounding Box Definition

The bounding box completely contains its horizontal geofence. For a keep-in geofence g_i, the bounding box is defined with x-values of $(\min x_{g_i} - \Delta_x, \max x_{g_i} + \Delta_x)$ and y-values of $(\min y_{g_i} - \Delta_y, \max y_{g_i} + \Delta_y)$. The value of Δ can either scale relative to the size of the geofence or be a constant value. The value of Δ does not impact the activation of the geofence. Computational cost is highest when transitioning into the bounding box, comparable to the cost of Ray Casting for that geofence boundary check. This paper uses $\Delta = 0.2 * (\max x_{g_i} - \min x_{g_i}, \max y_{g_i} - \min y_{g_i})$ based on illustration considerations. The value of Δ could be optimized based on the expected vehicle flight trajectory given communication between autopilot and geofencing systems. To divide the space between the bounding box and the geofence into triangles, the bounding box needs to be joined to a copy of the geofence boundary. The bottom left corner of the bounding box is the joining point to the geofence boundary through the vertex with the minimum

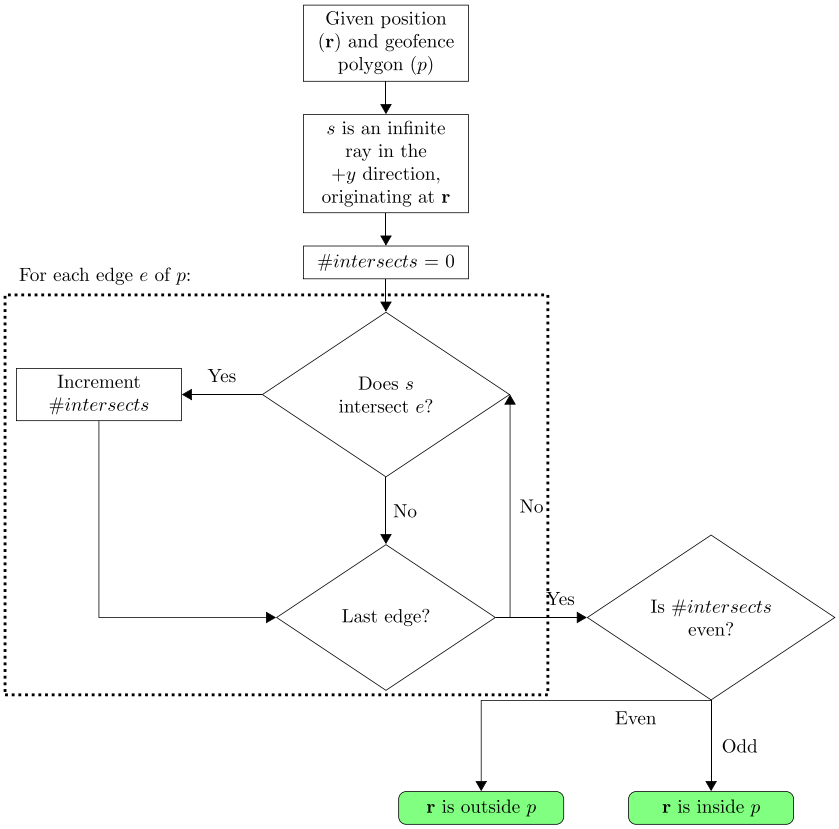


Fig. 4 Ray Casting algorithm. Green boxes indicate end states. The algorithm for Ray Casting is presented in [11]

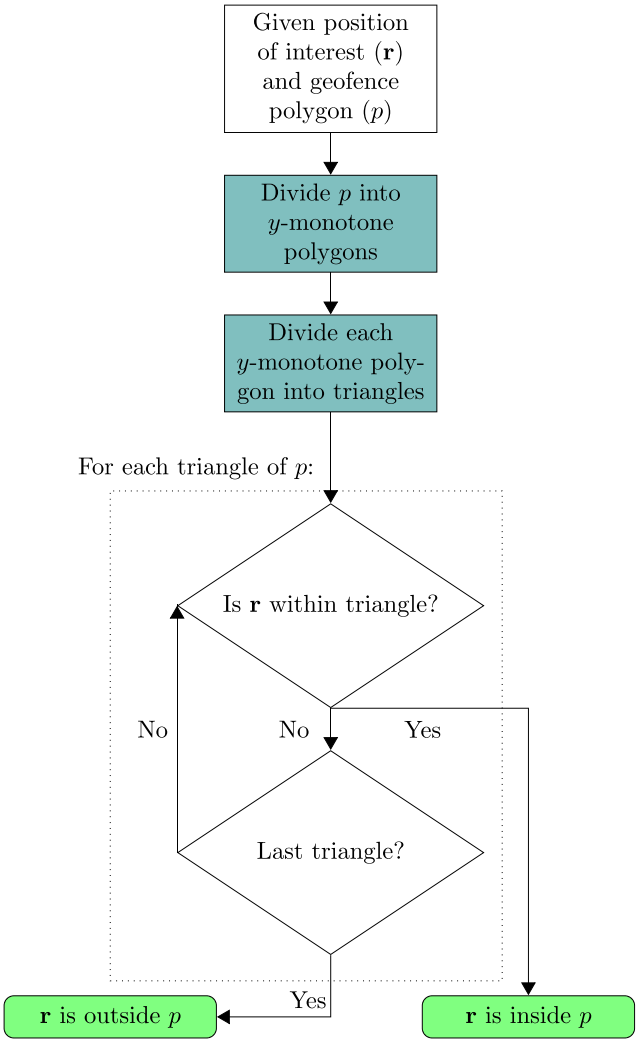


Fig. 5 Triangle Weight Characterization with Adjacency (TWCA) algorithm. Blue highlighting indicates initialization steps that are only executed once per flight. Green boxes indicate end states. The algorithm for dividing p into y-monotone polygons is shown in [8]. The algorithm for dividing y-monotone polygons into triangles is shown in [2]. The algorithm for determining if $\mathbf{r}$ is within a triangle is shown in [14]

x-value. If multiple geofence vertices have the minimum x-value, the vertex with the minimum y-value within the set of minimum x-value vertices is selected. To avoid issues in polygon division steps, a temporary shift is applied before division and removed afterwards. This shift is introduced to separate the co-located vertices of the bounding box polygon that is passed to the polygon division steps and has no impact on the geofence violation detection.

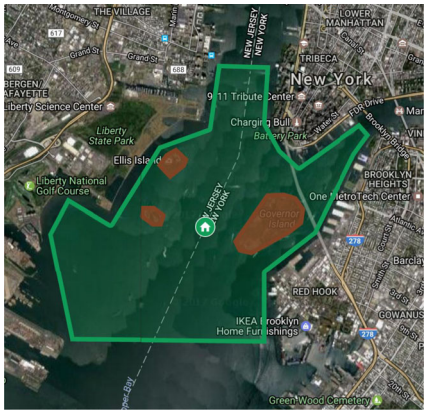
Inclusion of a bounding box in TWCA is important to reduce expected run-time complexity of the algorithm. In cases where the position of interest is outside the bounding box, no triangles need to be checked. When the position of interest is within the bounding box, it is within a triangle. Being within a triangle does not guarantee that the position of interest is not violating the geofence boundary, but each triangle is marked as inside or outside the geofence boundary when it is created. If the position of interest is known to be within a triangle, then the containing triangle can be found from the adjacency graph.

4.2 Polygon Division

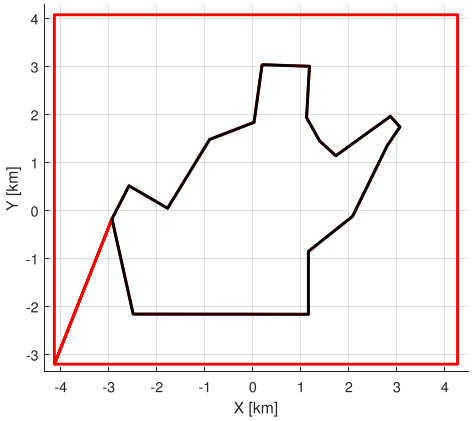
To divide an arbitrary geofence boundary into non-intersecting triangles, we implement the triangulation method described in Garey et al. [2] which relies on the regularization algorithm presented by Lee and Preparata [8]. To visualize the subdivision of an arbitrary polygon, TWCA is applied to the polygon shown in Fig. 6. TWCA initialization consists of two steps: divide the polygon into monotone polygons [8], and subdivide each monotone polygon into triangles [2]. Each of these steps is executed with respect to the y-axis but would also work if applied to the x-axis. Unlike the Wedge method, these methods do not create any additional vertices. There are other methods available to divide simple polygons into triangles without creating additional vertices, but those methods are not explored here.

4.2.1 Polygon to Monotone Polygons

A y-monotone polygon is defined as a polygon for which all lines parallel to the x-axis intersect a maximum of two edges of the polygon. To divide a polygon into monotone polygons, we iterate through the vertices from highest to lowest y-value, then from lowest to highest y-value, adding edges between vertices to create monotone polygons. Vertices with equivalent y-values are iterated over from left to right [8]. The original algorithm creates new edges between existing vertices both inside and outside the original polygon, but geofencing is only interested in the area of the original polygon. Thus, edges added that are outside the original polygon are ignored. An edge is determined to be outside the original polygon when the order of the edge vertices of the newly-defined polygon is opposite the order of the original polygon vertices, i.e., clockwise versus counterclockwise. Because some of the newly-generated edges are ignored, this algorithm is executed for each newly created polygon until no new edges are added. This ensures that the polygons being returned are all y-monotone. In Fig. 7a, the original polygon has been divided into three y-monotone polygons and the bounding box polygon has been divided into five y-monotone polygons.



(a) Urban geofence example with a keep-in geofence (green) and three keep-out geofences (red).



(b) Keep-in geofence (black) with bounding box (red).

Fig. 6 Example urban keep-in geofence located over Upper Bay, Hudson River, and East River with bounding box with keep-out geofences surrounding the contained islands

4.2.2 Monotone Polygon Conversion to Triangles

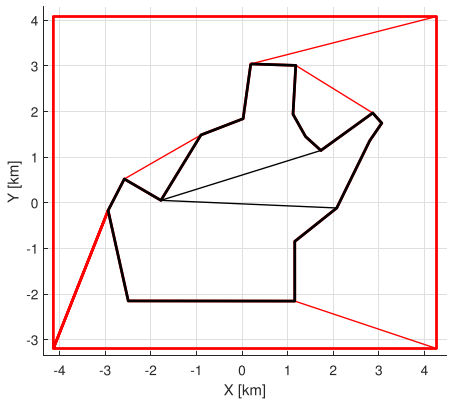
For each monotone polygon, the vertices are iterated over from highest to lowest y-value, iteratively adding edges to create triangles. Because the polygons are already y-monotone, all created edges are inside the polygon and therefore kept. For the example geofence, this algorithm is run eight times, once for each monotone polygon. Figure 7b illustrates the TWCA triangles with black lines for the geofence triangles and red lines for the bounding box triangles.

Any geofence represented by a simple polygon with v vertices can be divided into $\tau_g = v - 2$ triangles using this process. For the case of TWCA, both the original polygon and the bounding box must be divided into triangles. The bounding box consists of the original v vertices, the 4 vertices of the bounding box rectangle, and 2 vertices to connect the two sets of vertices. This results in the space between the geofence and the bounding box being divided into $\tau_b = (v+4+2)-2 = v+4$ triangles. The total number of triangles to be considered is $\tau = (v-2)+(v+4) = 2*(v+1)$ triangles.

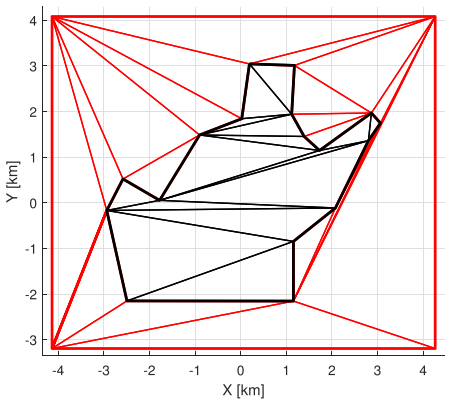
4.3 Triangle Occupancy Check

To determine if the launch position or any subsequent position, $\mathbf{r}$, is inside the geofence polygon, we check if $\mathbf{r}$ is inside each triangle. Let vertices of the i^{th} triangular cell be located at $\mathbf{r}_{i_1} = (x_{i_1}, y_{i_1})$, $\mathbf{r}_{i_2} = (x_{i_2}, y_{i_2})$, and $\mathbf{r}_{i_3} = (x_{i_3}, y_{i_3})$. Because a triangle is a $2-D$ convex hull, positions of the i^{th} triangular cell satisfy the following rank condition:

$$\text{Rank}\left[\begin{array}{cc}\mathbf{r}_{i_2}-\mathbf{r}_{i_1} & \mathbf{r}_{i_3}-\mathbf{r}_{i_1}\end{array}\right] = \left[\begin{array}{cc}x_{i_2}-x_{i_1} & x_{i_3}-x_{i_1}\\ y_{i_2}-y_{i_1} & y_{i_3}-y_{i_1}\end{array}\right] = 2. \tag{1}$$



(a) Geofence and bounding box divided into monotone polygons.



(b) Geofence and bounding box divided into triangles.

Fig. 7 Example geofence and bounding box divided into monotone polygons and then triangles

Therefore, position of an arbitrary point $\mathbf{r} = (x, y)$ in the motion plane can be uniquely expanded as

$$\begin{aligned}\mathbf{r} &= \mathbf{r}_{i_1} + w_{i_2}\left(\mathbf{r}_{i_2} - \mathbf{r}_{i_1}\right) + w_{i_3}\left(\mathbf{r}_{i_3} - \mathbf{r}_{i_1}\right) \\ &= \left(1 - w_{i_2} - w_{i_3}\right)\mathbf{r}_{i_1} + w_{i_2}\mathbf{r}_{i_2} + w_{i_3}\mathbf{r}_{i_3}.\end{aligned} \tag{2}$$

Setting $w_{i_1} = \left(1 - w_{i_2} - w_{i_3}\right)$, Eq. 2 can be rewritten as

$$\mathbf{r} = \sum_{k=1}^{3} w_{i_k}\mathbf{r}_{i_k} \tag{3}$$

where

$$\sum_{k=1}^{3} w_{i_k} = 1. \tag{4}$$

Considering Eqs. 3 and 4, distance weights w_{i_1}, w_{i_2}, and w_{i_3} are obtained from

$$\begin{bmatrix} x_{i_1} & x_{i_2} & x_{i_3} \\ y_{i_1} & y_{i_2} & y_{i_3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_{i_1} \\ w_{i_2} \\ w_{i_3} \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{5}$$

The distance weights satisfying Eq. 5 can expressed as follows:

$$w_{i_1}(x, y) = \frac{\left(x_{i_3} - x_{i_2}\right)\left(y - y_{i_2}\right) - \left(y_{i_3} - y_{i_2}\right)\left(x - x_{i_2}\right)}{\left(x_{i_3} - x_{i_2}\right)\left(y_{i_1} - y_{i_2}\right) - \left(y_{i_3} - y_{i_2}\right)\left(x_{i_1} - x_{i_2}\right)}$$

$$w_{i_2}(x, y) = \frac{\left(x_{i_1} - x_{i_3}\right)\left(y - y_{i_3}\right) - \left(y_{i_1} - y_{i_3}\right)\left(x - x_{i_3}\right)}{\left(x_{i_1} - x_{i_3}\right)\left(y_{i_2} - y_{i_3}\right) - \left(y_{i_1} - y_{i_3}\right)\left(x_{i_2} - x_{i_3}\right)}$$

$$w_{i_3}(x, y) = \frac{\left(x_{i_2} - x_{i_1}\right)\left(y - y_{i_1}\right) - \left(y_{i_2} - y_{i_1}\right)\left(x - x_{i_1}\right)}{\left(x_{i_2} - x_{i_1}\right)\left(y_{i_3} - y_{i_1}\right) - \left(y_{i_2} - y_{i_1}\right)\left(x_{i_3} - x_{i_1}\right)}. \tag{6}$$

$w_{i_k}(x, y) = \mathrm{c}$ ($k = 1, 2, 3$ and c is a constant) is a line parallel to a triangle side not passing through i_k. As examples, $w_{i_1} = \mathrm{c}$ is a line parallel to triangle side $i_2 - i_3$, $w_{i_2} = \mathrm{c}$ is a line parallel to triangle side $i_3 - i_1$, and $w_{i_3} = \mathrm{c}$ is a line parallel to triangle side $i_1 - i_2$. Also, $w_{i_k}(x_{i_j}, y_{i_j}) = \delta_{k,j}$, where $\delta_{k,j}$ is the Kronecker delta defined as follows:

$$\delta_{k,j} = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}. \tag{7}$$

In Fig. 8, the $x-y$ motion plane can be divided into seven sub-regions based on the signs of distance weights w_{i_1}, w_{i_2}, w_{i_3}. As shown, distance weights are all positive inside the i^{th} triangular cell.

If the distance weights of one of the triangles are all positive for the position of interest, then the position of interest is within the polygon.

Remark If a geofence area is sufficiently large, it may not be approximated by a planar surface. For this case, the geofence domain can be considered as a spherical surface with longitude ϕ and latitude λ. The proposed TWCA method can still be applied for boundary violation checking over a spherical surface. By substituting x, x_{i_1}, x_{i_2}, x_{i_3}, y, y_{i_1}, y_{i_2}, y_{i_3} by ϕ, ϕ_{i_1}, ϕ_{i_2}, ϕ_{i_3}, λ, λ_{i_1}, λ_{i_2}, λ_{i_3}, weights $w_{i_1}(\phi, \lambda)$, $w_{i_2}(\phi, \lambda)$, and $w_{i_3}(\phi, \lambda)$ can be obtained from Eq. 6. Similar to a planar representation, the UAS is enclosed by the i^{th} sector over the spherical geofence surface if $w_{i_1}(\phi, \lambda)$, $w_{i_2}(\phi, \lambda)$, and $w_{i_3}(\phi, \lambda)$ are all positive.

4.4 Adjacency Graph

The adjacency graph can be stored as an adjacency matrix, multiply linked list, or an array. A multiply linked list is typically preferred for low-level languages such as C++. However, an array is used in this work because case studies were implemented in MATLAB. Adjacent triangles are

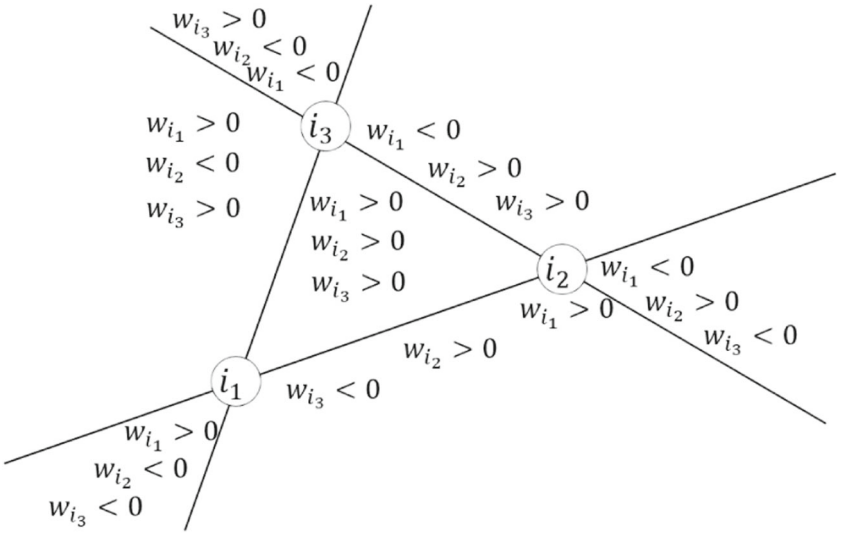


Fig. 8 Division of the motion plane into seven sub-regions based on the signs of distance weights w_{i_1}, w_{i_2}, and w_{i_3}

defined as triangles that share a common side. To efficiently search for an occupied triangle, each search is initialized at the triangle occupied at the prior time step. If that triangle is no longer occupied, a Breadth First Search is executed with a "visited" list that eliminates the possibility of checking the same triangle more than once.

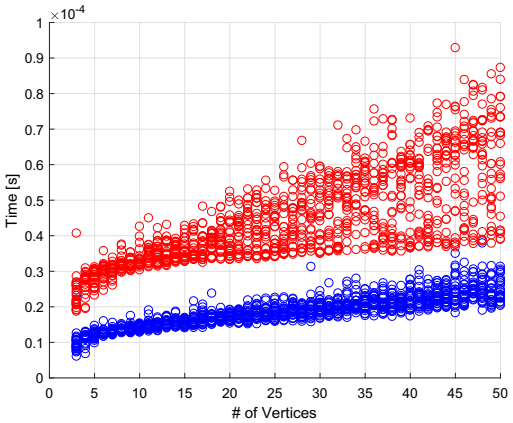
5 Results

To evaluate TWCA performance, it is compared against Ray Casting with a bounding box using MATLAB on a laptop running Windows 10. Embedded UAS applications would require these algorithms to be compiled in a language such as C so the execution times will be lower. The relative performance trends presented here are expected to translate to embedded codes. Trial geofences are randomly generated such that test cases are 25 instances of geofences with 3 to 50 vertices for a total of 1,200 geofences. Geofence vertices have a maximum possible magnitude of 50 m in the x and y directions. Each geofence is tested with 100 simulated flight paths from the origin $(0, 0)$ to a randomly generated end position. The flight path end positions have a maximum possible magnitude of 50 m in the x and y directions. Two sampling intervals of the flight path are tested: 10 positions along the flight path and 100 positions along the flight path. State sampling frequency can therefore be analyzed given that the adjacency graph search complexity depends on number of triangle passages taken from the previously-occupied triangle.

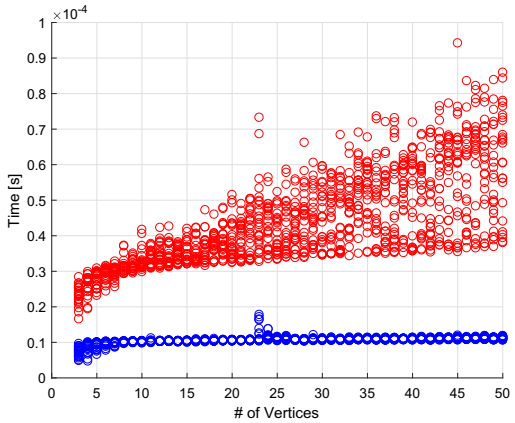
Figure 9 shows the average time for a position of interest query for each of the 1,200 randomly generated geofences. TWCA queries are shown in blue, while Ray Casting query times are shown in red. Figure 9a and b show the results when the position is sampled 10 times and sampled 100 times along each of the 100 flight paths respectively. The frequency of position samples along the flight path does not impact execution time of the Ray Casting algorithm because Ray Casting does not utilize any information from prior states during execution. However, the Ray Casting execution time is seen to scale linearly with the number of vertices in the geofence.

The frequency of position samples strongly impacts the expected execution time of TWCA. By sampling the position of the vehicle at a higher frequency, the average execution time of TWCA is shown to be independent of the number of geofence vertices rather than increasing linearly as in Ray Casting. This result is expected because as the time between vehicle position estimates increases, the likelihood that the vehicle has moved multiple triangles away from the previously occupied triangle increases as well.

Figure 10 displays results in terms of the percentage of triangles explored: τ_e/τ where τ is the total number of triangles and τ_e is the number of triangles explored. The black data points present the percentage of triangles explored for each geofence averaged over each position sample in each flight path. Both plots in Fig. 10 show that as the number of geofence vertices increases, the percentage of triangles explored for each position update decreases.

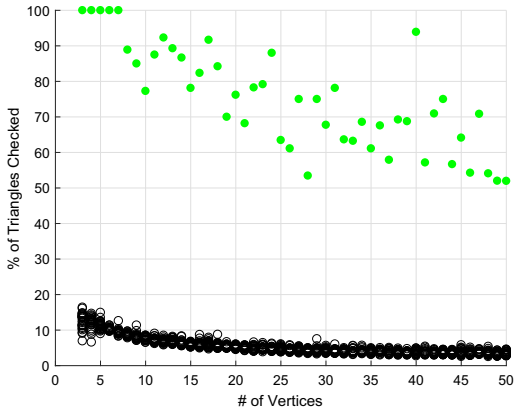


(a) Calculation time averaged over 10 positions sampled along 100 flight paths from the origin to a random waypoint. (Mean time of 1000 sampled positions per geofence.)

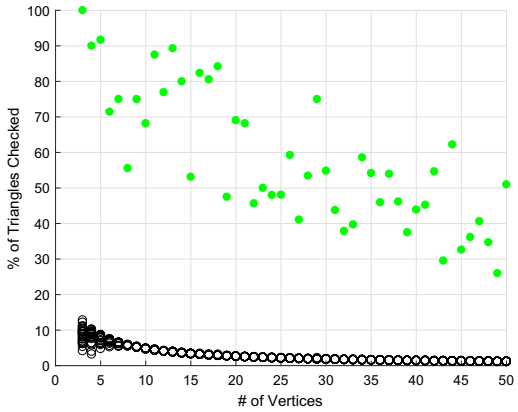


(b) Calculation time averaged over 100 positions sampled along 100 flight paths from the origin to a random waypoint. (Mean time of 10,000 sampled positions per geofence.)

Fig. 9 Average time per boundary violation check for each of 25 randomly generated geofences with 3 to 50 vertices for a total of 1,200 random geofences. Ray Casting results are shown in red. TWCA results are shown in blue. Note that sporadic inconsistencies in the data are likely due to a background process on the computer and not relevant to the results illustrated for geofences of size 23 to 25 vertices



(a) Maximum and average percentage of triangles explored for cases with vehicle position sampled 10 times on 100 flight paths from the origin to a random waypoint.



(b) Maximum and average percentage of triangles explored for cases with vehicle position sampled 100 times on 100 flight paths from the origin to a waypoint.

Fig. 10 Maximum percentage of triangles explored before locating the triangle containing the vehicle is shown in green. One maximum is reported for each set of 25 geofences with the same number of vertices and each position sampled for each flight path explored. Average percentage of triangles explored before locating triangle containing vehicle is shown in black. The average percentage of triangles is reported for each of the 1,200 geofences. The average is over each of the position samples for each of the explored flight paths

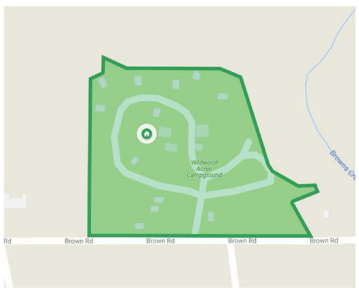
This trend is also present in the worst-case scenarios for TWCA. The green data points in Fig. 10 are the highest percentage of triangles checked for each number of geofence vertices. This data shows that the inverse relation between the number of geofence vertices and the percentage of triangles checked is still present in the worst observed cases. Although the trends are observable in both plots, the trends are more pronounced in Fig. 10b than in Fig. 10a. As discussed above, this reflects that TWCA executes with a time complexity independent of the complexity of the geofence by leveraging knowledge of the triangle occupied at the previous time step.

6 Case Study

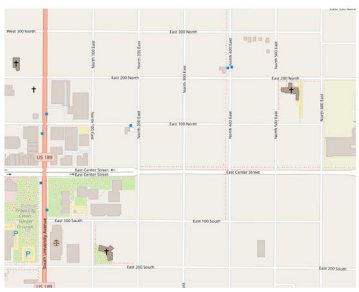
With a geofence boundary violation detection algorithm like TWCA that has an expected execution time independent of the complexity of its borders, it becomes possible to form the geofence based on data from a mapping database. The usage of a mapping database would reduce pre-flight pilot workload of manually entering geofence data.

In the near future, geofences generated using property maps could enable flights over areas such as private property and public parks. Figure 11a shows the outline of a privately owned campsite in a sparsely populated area. If the campsite owners want to photograph the campsite from a UAV, they might want to utilize a geofence to prevent flights over the adjacent properties and the public roads. They might also choose to manually define keep-out geofences around the campers and trees to preserve the privacy of their guests and prevent collisions.

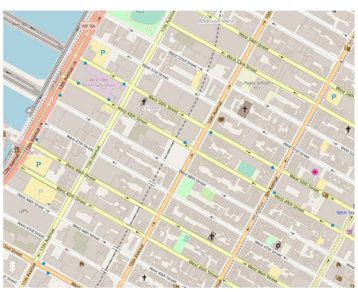
In the not too distant future, as UAS operate in increasingly populated areas, similar geofences could be automatically generated for suburban and urban flight volumes. The keep-in geofences might be created encompassing parks



(a) Bemus Point, New York.



(b) Salt Lake City, Utah.



(c) New York, New York.

Fig. 11 Examples of rural and urban environments that might be operational areas for UAS with geofences in the future. Generated using Google Maps at www.google.com/maps/ and OpenStreetMap at www.openstreetmap.org

or multiple blocks or entire city regions, while keep-out geofences might be used to denote buildings.

In a city aligned with the Earth coordinate axes such as Salt Lake City (Fig. 11b), the bounding box buffer distance Δ could be defined as a value that minimized overlap between keep-out geofences or that minimized the instances of being within a keep-out geofence bounding box. These optimizations become more complicated as the geofence becomes less regular. Consider the section of Manhattan in New York City shown in Fig. 11c. The city blocks are at a consistent angle to the cardinal axes; this inefficiency can be eliminated by applying a yaw (heading) transformation to the local map (ground) coordinate axes. In general, the coordinate axis orientation can be optimized to maximize the flight area not covered by a keep-out geofence bounding box.

In an urban environment, there are obstacles to be avoided other than other aircraft, including buildings, power lines, and street lights. The issue is further complicated by the existence of urban canyons, where GPS is denied due to the surrounding buildings. In these areas, even if every obstacle were designated a keep-out geofence, accumulated state estimation error might make it impossible for the geofence to guarantee a collision free flight. In these cases, the addition of a sense and avoid system could enable safe flight through a geofenced region without relying on a potentially inaccurate state estimate. The inclusion of a sense and avoid system in addition to a geofencing system is critical to safe flight.

When operating at higher altitudes, in shared airspace with manned aircraft, large UAS can still benefit from geofencing. The keep-in geofence ensures that the UAS does not exit its designated flight boundary, and keep-out geofences can ensure separation from buildings, terrain, and ultimately other aircraft operating in fixed regions / bounding boxes.

7 Conclusion

This paper has presented an efficient methodology for defining a static geofence for an unmanned aircraft system (UAS) containing both keep-in (inclusion) geofences and keep-out (exclusion) geofences. This paper developed Triangle Weight Characterization with Adjacency (TWCA) to detect horizontal geofence boundary violations with an execution time complexity that is independent from the number of vertices used to define the geofence. Case studies showed that TWCA on average has better performance than Ray Casting, particularly when geofence polyhedra are complex, with a large number of vertices.

References

1. Alciatore, D., Miranda, R.: A winding number and point-in-polygon algorithm. Glaxo Virtual Anatomy Project Research Report, Department of Mechanical Engineering, Colorado State University (1995)
2. Garey, M.R., Johnson, D.S., Preparata, F.P., Tarjan, R.E.: Triangulating a simple polygon. Inf. Process. Lett. **7**(4), 175–179 (1978)
3. Gilabert, R.V., Dill, E.T., Hayhurst, K.J., Young, S.D.: Safeguard: Progress and test results for a reliable independent on-board safety net for UAS. In: 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), pp. 1–9. IEEE (2017)
4. Government US: Operation and certification of small unmanned aircraft systems. In: Title 14 Code of Federal Regulations, Part 107 (2016)
5. Hormann, K., Agathos, A.: The point in polygon problem for arbitrary polygons. Comput. Geom. **20**(3), 131–144 (2001)
6. Huang, C.W., Shih, T.Y.: On the complexity of point-in-polygon algorithms. Comput. Geosci. **23**(1), 109–118 (1997)
7. Kopardekar, P.H.: Unmanned aircraft systems traffic management (UTM) safely enabling UAS operations in low-altitude airspace (2017)
8. Lee, D.T., Preparata, F.P.: Location of a point in a planar subdivision and its applications. SIAM J. Comput. **6**(3), 594–606 (1977)
9. Li, J., Wang, W.: Point-in-polygon tests by determining grid center points in advance. In: Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific, pp. 1–7. IEEE (2013)
10. Li, J., Wang, W., Wu, E.: Point-in-polygon tests by convex decomposition. Comput. Graph. **31**(4), 636–648 (2007)
11. Narkawicz, A., Hagen, G.: Algorithms for collision detection between a point and a moving polygon, with applications to aircraft weather avoidance. In: 16th AIAA Aviation Technology, Integration, and Operations Conference, p. 3598 (2016)
12. Nordbeck, S., Rystedt, B.: Computer cartography point-in-polygon programs. BIT Numer. Math. **7**(1), 39–64 (1967)
13. Preparata, F.P., Shamos, M.I.: Introduction. In: Computational Geometry, pp. 1–35. Springer (1985)
14. Rastgoftar, H.: Continuum Deformation of Multi-Agent Systems. Birkhäuser, Cambridge (2016)
15. Rastgoftar, H., Jayasuriya, S.: Evolution of multi-agent systems as continua. J. Dyn. Syst. Meas. Control. **136**(4), 041014 (2014)
16. Salomon, K.B.: An efficient point-in-polygon algorithm. Comput. Geosci. **4**(2), 173–178 (1978)
17. Stevens, M.N., Atkins, E.M.: Multi-mode guidance for an independent multicopter geofencing system. In: 16th AIAA Aviation Technology, Integration, and Operations Conference, p. 3150 (2016)
18. Stevens, M.N., Atkins, E.M.: Layered geofences in complex airspace environments. In: 2018 Aviation Technology, Integration, and Operations Conference, p. 3348 (2018)
19. Yang, S., Yong, J.H., Sun, J., Gu, H., Paul, J.C.: A point-in-polygon method based on a quasi-closest point. Comput. Geosci. **36**(2), 205–213 (2010)
20. Žalik, B., Kolingerova, I.: A cell-based point-in-polygon algorithm suitable for large sets of points. Comput. Geosci. **27**(10), 1135–1145 (2001)

Mia N. Stevens is a Robotics PhD candidate at the University of Michigan, Ann Arbor, MI, USA. She received a Bachelor's of Science in Aerospace Engineering with Information Technology from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2014, and a Master of Science in Robotics from the University of Michigan, Ann Arbor, MI, USA, in 2016. Her current research interests include software design for UAS, safety system development, and autonomous systems.

Dr. Hossein Rastgoftar received the B.Sc. degree in mechanical engineering-thermo-fluids from Shiraz University, Shiraz, Iran, the M.S. degrees in mechanical systems and solid mechanics from Shiraz University and the University of Central Florida, Orlando, FL, USA, and the Ph.D. degree in mechanical engineering from Drexel University, Philadelphia, PA, USA, in 2015. He is currently an assistant research scientist at the University of Michigan, Ann Arbor, MI, USA. His current research interests include dynamics and control, multiagent systems, human-cyber physical systems, optimization and Markov decision processes, multi criteria decision analysis, and differential game theory.

Dr. Ella M. Atkins is a Professor of Aerospace Engineering at the University of Michigan, where she directs the Autonomous Aerospace Systems Lab and is Associate Director of Graduate Programs for the Robotics Institute. Dr. Atkins holds B.S. and M.S. degrees in Aeronautics and Astronautics from MIT and M.S. and Ph.D. degrees in Computer Science and Engineering from the University of Michigan. She is past-chair of the AIAA Intelligent Systems Technical Committee and has served on the National Academy's Aeronautics and Space Engineering Board, the Institute for Defense Analysis Defense Science Studies Group, and an NRC committee to develop an autonomy research agenda for civil aviation. She pursues research in Aerospace system autonomy and safety.