# A New Approach to Complex Dynamic Geofencing for Unmanned Aerial Vehicles

Vihangi Vagal
*Risk Advisory*
*Deloitte LLP*
London, United Kingdom
vihangiv@gmail.com

Konstantinos Markantonakis
*Information Security Group*
*Royal Holloway University of London*
Egham, United Kingdom
k.markantonakis@rhul.ac.uk

Carlton Shepherd
*Information Security Group*
*Royal Holloway University of London*
Egham, United Kingdom
carlton.shepherd@rhul.ac.uk

***Abstract*—The anticipated widespread use of unmanned aerial vehicles (UAVs) raises significant safety and security concerns, including trespassing in restricted areas, colliding with other UAVs, and disrupting high-traffic airspaces. To mitigate these risks, geofences have been proposed as one line of defence, which limit UAVs from flying into the perimeters of other UAVs and restricted locations. In this paper, we address the concern that existing geometric geofencing algorithms lack accuracy during the calculation of complex geofences, particularly in dynamic urban environments. We propose a new algorithm based on alpha shapes and Voronoi diagrams, which we integrate into an on-drone framework using an open-source mapping database from OpenStreetMap. To demonstrate its efficacy, we present performance results using Microsoft's AirSim and a low-cost commercial UAV platform in a real-world urban environment.**

## I. Introduction

Geofencing is a widely used security technique for preventing UAVs from flying into controlled airspaces, such as power plants, airports, and military installations. In general, geofences compare the UAV's current geographical location with predefined or dynamically identified no-fly zones (NFZs) and other restricted areas. The UAV operator is then notified of any potential or existing contraventions. On some UAV platforms, this may be followed by the termination of the drone's rotors or the automatic triggering of an emergency landing. Geofences are also used to avoid dangerous UAV-to-UAV collisions, particularly in urban airspaces [1]–[3]. Numerous commercial and non-commercial geofencing systems have already been developed, such as NASA's Safeguard project [4] and DJI's Geospatial Environment Online (GEO) [5] system.

Existing geofencing algorithms, like polygonal- and circular-based geofencing, enclose restricted regions using geometric shapes/boundaries to which the UAV's current location is compared. This paradigm is inherently reliant on the accuracy of the geometric boundary with respect to a restricted physical area, e.g. airport, prison, or school. If the geofence boundary is larger than the physical object in reality, then the UAV may be prevented from entering legitimate locations. In urban environments, this may arise if restricted areas/NFZs are close to residential areas. Conversely, if the geofence is smaller than the physical object, then operators run the risk of mid-air UAV collisions and inadvertently entering NFZs, which is a criminal offence in many jurisdictions. This challenge is most pronounced in dynamic UAV environments, where precise geofences may not easily be determined in advance and must be continuously monitored and re-evaluated.

In light of these issues, we evaluate a new approach to geometric geofencing based on the application of $\alpha$-shapes, proposed by Edelsbrunner et al. [6], and Voronoi diagrams. We integrate our proposal using OpenStreetMap and provide an empirical analysis of existing geofencing techniques using Microsoft's AirSim and a Navio2 UAV in a real-world urban environment in the United Kingdom. This analysis includes experimental results of the accuracy and performance of our proposal against traditional polygonal geofencing.

We anticipate that our proposal will provide a new method for mitigating mid-air collisions, e.g. for UAV swarms, and for preventing incursions into restricted airspaces. While the focus of this paper is UAVs, geofences are also used in fleet and freight management, maritime scenarios, and for user authentication [7]–[9]. Our proposed methods may also be applicable in these domains. In short, our contributions are as follows:

- An analytical and experimental comparison of existing geofencing techniques, their accuracy, performance, and shortcomings.
- The evaluation of a novel geofencing method based on $\alpha$-shapes and Voronoi diagrams, which utilises an on-drone geofencing database.
- Implementation and experimental results of our proposed technique within a simulated environment using Microsoft's AirSim and an urban deployment using a Navio2-based UAV.

## II. Geofencing Methods

### A. Overview and Setup

Geofences typically fall into one of two paradigms: *static* geofences remain the same overtime over restricted regions, e.g. airports and military bases, which are referenced from a database of predefined locations. *Dynamic* geofences are generated in real-time around unforeseen sensitive areas and obstructions on the UAV's flight path; for example, as spheres around other UAVs. Both approaches necessitate up-to-date geofencing databases and reliable location data feeds.

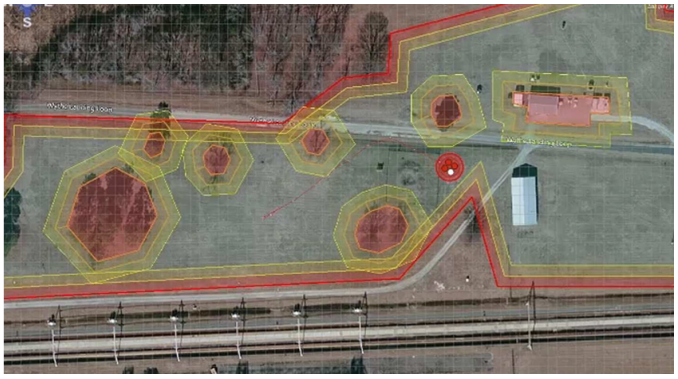

Fig. 1: Dual geofencing used by Safeguard [11], showing NFZ (red), termination (orange) and warning areas (yellow).

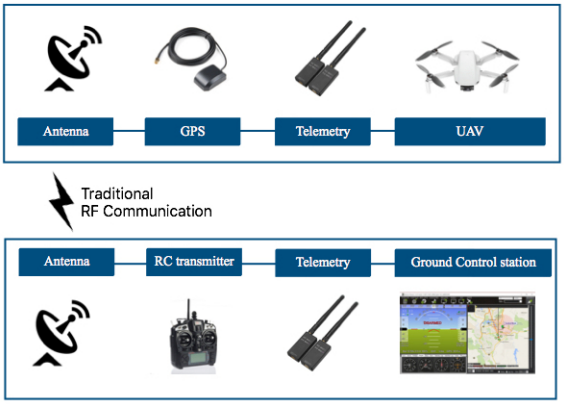


Fig. 2: High-level GCS-to-UAV communication architecture.

Besides methods of generation, geofences can be categorised into three modes of operation. *Keep-in* geofences limit UAVs from flying outside a predefined boundary. An example is NASA's UAS Traffic Management (UTM) system, which uses height, vertical and horizontal buffers for geospatial containment with respect to a cylindrical volume surrounding the UAV [10]. *Keep-out* geofences, meanwhile, are formed around restricted areas to prevent UAVs from flying into them. DJI, a leading commercial drone manufacturer, uses keep-out geofences for its UAVs. DJI UAVs are restricted to operate in specific geofenced areas, including airports, prisons, and power plants. The system employs an 'enhanced warning zone' around NFZs in which drone pilots are notified. If the NFZ is violated, the operator loses control over the UAV before undergoing an immediate landing procedure; the UAV cannot take off or power-up within an NFZ [5]. Lastly, *dual* geofences use keep-in *and* keep-out methods, which can be beneficial in urban environments [1]. They limit UAVs to a set area from the operator while respecting NFZs within that area. The NASA Safeguard system uses such an approach [4], [11], which uses multiple warning and UAV termination layers (see Fig. 1).

In this work, we consider geofencing from the perspective of a UAV that communicates with a ground control station (GCS). The essential components of such a setup are shown in Fig. 2 and described as follows:

- **Global navigation satellite system (GNSS)**: Provides the UAV's current location in the form of latitude, longitude and altitude coordinates in real time. GPS, GLONASS, BDS, or Galileo may be used depending on the target deployment region.
- **Ground Control Station (GCS)**: Communicates with the UAV during its operation, which may be used for direct control (manual operation) and for displaying geofencing infringements to the operator.
- **Telemetry module (TM)**: The ground telemetry module is attached to the GCS and the air telemetry module to the drone. The UAV and GCS send and receive essential data using these modules.
- **Radio-frequency transceiver (RF)**: The RF module is part of the GCS; it is coupled with TM and the antenna to communicate with the UAV.
- **Flight controller**: A hardware-firmware setup whereby UAV actuators are synchronously connected and controlled; ArduPilot [12], PX4 [13], Cleanflight [14], Navio [15], and Betaflight [16] are some widely used flight controllers for commercial UAVs.
- **Geofencing database**: A database referenced by the UAV that contains the details of restricted locations. The database may include coordinates of those regions and additional restrictions, such as permitted times of operation, noise levels, height, and video restrictions.

Today's commercial UAVs are heavily reliant on GNSS measurements for detecting potential and current geofence incursions. This is performed by comparing measurments to those in a geofencing database using a desired geometric geofencing algorithm (see Section II-B), which can be integrated into a GCS or on the UAV itself, including within autopilot modules. Geofences may be predetermined during the mission planning stage or by dynamically referencing a data source mid-flight. In both cases, geofences may be created with the assistance of open-source or proprietary mapping tools in tandem with human input and/or artificial intelligence. It is important to note that inaccurate and imprecise geofence definitions, GNSS sensor failures, adverse weather conditions, and actuator malfunctions may all lead to incursions into restricted airspaces.

### B. Geometric Algorithms

Geometric algorithms are used to continuously evaluate whether a UAV's GNSS coordinates are in contravention of active geofences in $\mathbb{R}^2$ or $\mathbb{R}^3$ Euclidean space.[1] Common geometric methods include polygonal, spherical, cylandrical, and elliptical geofencing [1], [3], [17], [18].

For a point of interest, $p$, e.g. the UAV's current location, polygonal geofencing operates by projecting an infinite ray, $\gamma$, through $p$. $p$ is considered to be within the geofence's

[1]Two-dimensional geofencing is typically used where the UAV's latitude and longitude are restricted, but not its altitude.

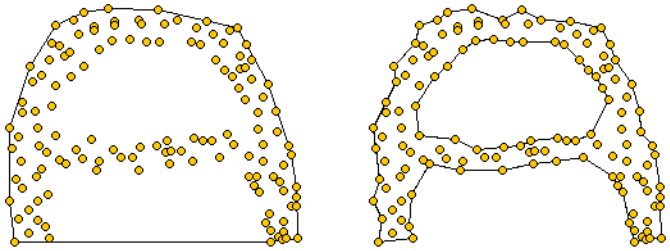

Fig. 3: Two $\alpha$-shapes of a discrete set of points in $\mathbb{R}^2$ using high (left) and lower (right) values of $\alpha$ [19].

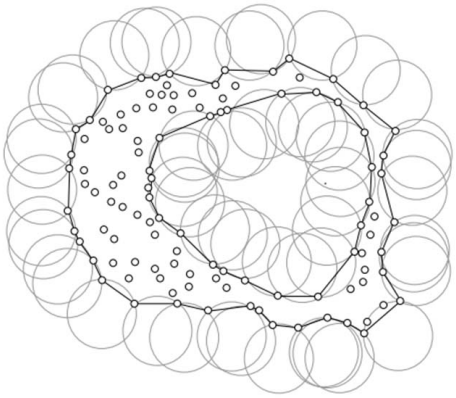

Fig. 4: Generating an $\alpha$-shape from disks with radius $\alpha$ [20].

perimeter if the number of edges that $\gamma$ intersects is odd on either side of $p$. In comparison, spherical geofencing computes the absolute distance, $d$, between the centre of the sphere of radius $r$ and a point of interest. A point is considered inside the boundary if $d < r$. This method has lower computational complexity than polygonal geofencing, but lacks the ability to tightly enclose complex physical locations or objects. Circular geofencing is the two-dimensional analogue of this method. Cylandrical and elliptical geofences function similarly by forming virtual cylinder- and ellipse-based perimeters and testing whether the UAV's coordinates lie within those perimeters.

The main drawback of non-polygon methods is that the distance between object edges and the drawn boundary may significantly differ for complex, polygon-shaped objects. These methods make it difficult to precisely enclose restricted areas, particularly when multiple restricted areas are nearby. This is further compounded by the use of buffer spaces for mitigating operational uncertainties, such as adverse weather conditions. In urban environments, it is conceivable that the interference between multiple buffered geofences may unnecessarily impede UAV flight plans and the reachability of destinations.

### C. Evaluated Method

To address the shortfalls of existing geometric algorithms for complex dynamic environments, we evaluate a novel framework based on the application of alpha shapes [6].

An *alpha* shape, or $\alpha$-shape, of a discrete set of points, $S \in \mathbb{R}^n$, is a polytope determined from $S$ and a real value, $\alpha$. It is considered a subgraph of the Delaunay triangulation and a generalisation of the convex hull, which is the intersection of all convex sets containing $S$. When $\alpha \to \infty$, the $\alpha$-shape converges to the convex hull of $S$; as $\alpha \to 0$, the $\alpha$-shape degenerates to the point set $S$. Intuitively, one can tailor this value, $0 \leq \alpha \leq \infty$, to produce an alpha shape graph of varying fineness, as shown in Fig. 3. Alpha shape graphs have already been shown to be an effective method for generating keep-in and keep-out geofences in complex urban environments [1]. More formally:

**Definition 1** (Alpha shapes [6], [19])**.** Let a generalised disk of radius, $r$, have the following properties:

- If $\alpha > 0$, it is an ordinary closed disk of radius $r = 1/\alpha$.
- If $\alpha = 0$, it is a half-plane.
- If $\alpha < 0$, it is the complement of a closed disk with $r = -1/\alpha$.

Given a set of points, $S$ and a value for $\alpha$, an alpha shape graph is constructed as follows:

1) For each point, $p_i \in S$, create a vertex $v_i$.
2) Create an edge between two vertices $v_i$ and $v_j$, $i \neq j$, when there exists a generalised disk with $r = 1/\alpha$ containing $S$ and which satisfies the property that $p_i$ and $p_j$ lies on its boundary (Fig. 4).

*Voronoi diagrams*, also known as Dirichlet tessellations, can also be combined with $\alpha$-shapes. A Voronoi diagram is a tessellation method for partitioning a discrete set of points, $S$, as such:

**Definition 2** (Voronoi diagram)**.** Let $d$ be a distance function, such as the Euclidean distance, between two points in a finite set, $(p_i, p_j) \in S$. The collection of all points closest to $p_i \in S$ is known as the Voronoi region, $V_S(p_i)$, for a metric space, $\mathcal{X}$, e.g. $\mathbb{R}^3$: $V_S(p_i) = \{x \in \mathcal{X} \mid d(x, p_i) \leq d(x, p_j)\ \forall\ i \neq j\}$. The Voronoi diagram, $\mathbf{V}(S)$, is then defined as $\mathbf{V}(S) = \{V_S(p_1), V_S(p_2), \ldots, V_S(p_n)\}$.

Voronoi diagrams have been proposed in the UAV path planning literature as an efficient method for circumnavigating obstructions and restricted areas in dynamic environments [21]–[23]. It is possible to use weighted Voronoi graphs in which weights are allocated to graph edges to inform the difficulty of traversing challenging areas [21]. The shortest path between two points can then be deduced using a path finding method, such as A* search or Dijkstra's algorithm. In this work, we propose a combination of Voronoi diagrams and $\alpha$-shapes, illustrated in Fig. 5, for unifying the benefits of $\alpha$-shape geofencing and the path planning applications of Voronoi diagrams.

## III. System Workflow

From a UAV system-level perspective, the flowchart of our geofencing method contains three main stages shown in Fig. 6. The first stage is an initialisation step for enforcing the activation of the UAV's geofencing functionality and ascertaining its current location. Next, in stage two, the platform loads the geofencing repository, i.e. the list of

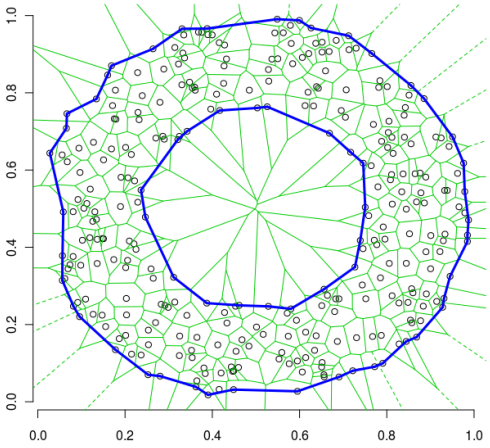


Fig. 5: Combining an $\alpha$-shape and a Voronoi diagram to form geofencing boundaries (blue) with internal edges for facilitating UAV path planning [24].

NFZs and restricted airspaces, and computes the $\alpha$-shapes corresponding to these locations on the UAV's flight path. The coordinates of the $\alpha$-shapes are then stored for future reference; the pre-computation of $\alpha$-shapes before a take-off allows for faster evaluation and lower response times mid-flight. This repository can be acquired in an online fashion or, in offline environments, the UAV may receive a recent copy of the geographic operating environment pre-flight. The UAV's current location is then compared to these shapes to detect current and potential geofencing violations. Lastly, stage three alerts the UAV operator if a geofencing violation is detected; if needed, the UAV flight controller may terminate the rotors or initiate an emergency landing.

## IV. Implementation

### A. AirSim Experiments

AirSim is a cross-platform, open-source simulator developed by Microsoft for autonomous vehicle research based on the Unreal Engine [25]. It supports software-in-the-loop simulation with widely used, off-the-shelf flight controllers, such as the PX4 and ArduPilot, and hardware-in-loop with PX4 within a virtual environment (Fig. 7). AirSim provides access to C++ and Python APIs for vehicle control and to retrieve continuous information about the target vehicle. We used these to implement and trial our proposed geofencing method, including its comparison to existing geofencing methods, prior to a real-world deployment. The simulator uses the NED coordinate system, i.e. X (horizontal movement), Y (vertical moment), Z (altitude) coordinates, which was used as the basis for determining the UAV's location within the aforementioned workflow in Fig. 6. The AirSim built-in GPS module was used to ascertain the UAV's current location.

The algorithm first computes the nearest objects to the UAV using the pre-computed database of objects in the virtual environment. In our implementation, this was stored as a JSON

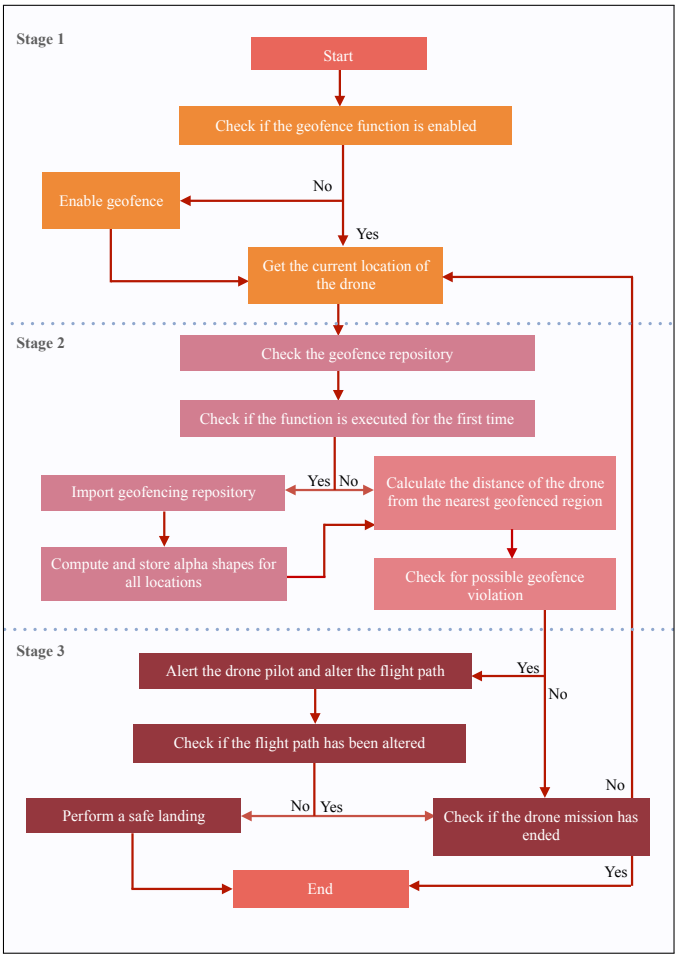


Fig. 6: System flowchart.

file, which was parsed before computing and saving the $\alpha$-shapes for the current operating region. For the computation of the $\alpha$-shapes themselves, we leveraged the `alphashapes` Python package [26]. Using this, we instrumented the simulation platform to implement a keep-out geofencing around restricted objects. Moreover, we implemented keep-in geofencing by limiting the UAV to a particular geographic area within the virtual environment.

The UAV's flight path was determined from the user-generated inputs through the Visual Studio command prompt. The path heading was automatically re-adjusted when there is a possibility of entering a restricted location; if this was overridden by the user, then the UAV undergoes an emergency landing before it enters that area.

### B. Navio2 Drone Platform

After prototyping the proposed method in AirSim, we then evaluated its performance in an urban environment using an off-the-shelf, low-cost (<$500 USD) commercial drone platform. This made use of a Navio2 add-on shield [15]—an autopilot system for the widely used Raspberry Pi board [27]—to implement a quadcopter-based UAV (Fig. 8). The components for implementing the hardware platform are as follows:

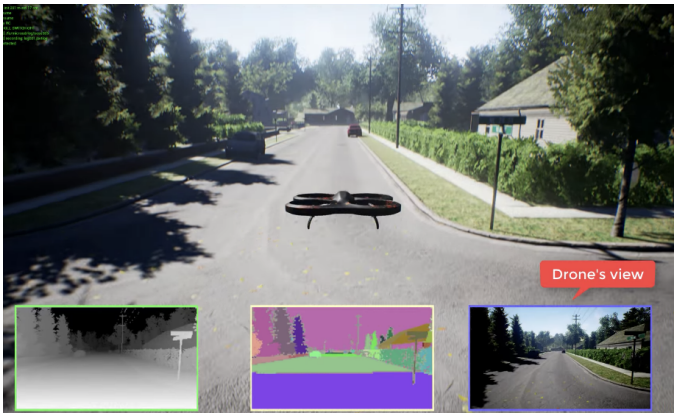


Fig. 7: AirSim UAV simulation platform [25].

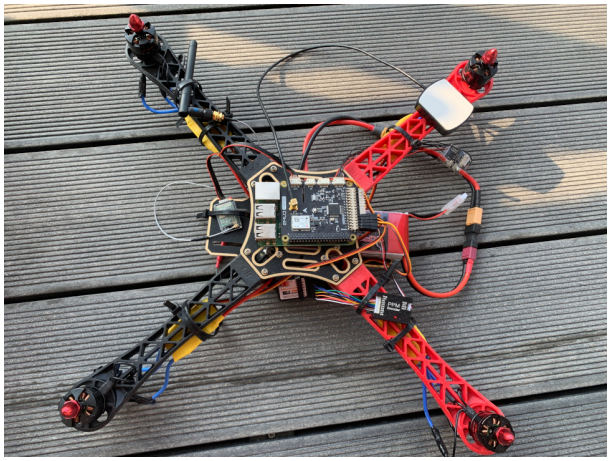

Fig. 8: Assembled Navio2-based quadcopter.

- **Raspberry Pi 3B+**: A single-board computer with a Broadcom BCM2837B0 system-on-chip (with a quad-core ARM Cortex-A53 at 1.4GHz), 1GB DDR2 SRAM, dual-band wireless LAN, Bluetooth 4.2/BLE, and a 40-pin general-purpose input/output (GPIO) interface [27].
- **Navio2 Shield**: A Raspberry Pi add-on shield that provides a GNSS receiver—supporting GPS, GLONASS, Galileo and BDS—dual IMUs, a remote-controlled I/O co-processor with 14 PWM output channels for motors and servos, and a high-resolution barometer. The Navio2 supports the Ardupilot open-source autopilot firmware and exposes Python APIs for software-based control; its average current draw is 150mA during operation.
- **Radio Controller**: A six-channel transceiver for enabling the UAV to be controlled manually by the user.
- **Telemetry**: A pair of air and ground modules attached to the drone and ground control station. The telemetry modules communicate in-flight data, e.g. position and speed, wirelessly between the UAV and GCS.
- **Battery**: A lithium polymer (LiPo) battery supplies power to the drone with a 3000mAh capacity.
- **Actuators**: A combination of motors, propellers and ESCs were selected based on the thrust-to-weight ratio; specifically, 30A brushless motors with 20V ESC and 1048 propellers were used. (ESCs are only required when using brushless motors; brushed motors only need a direct current voltage input).
- **MavProxy**: The MAVProxy is Ardupilot's GCS implementation a UAVs. It provides a portable GCS for UAVs that support the MAVlink Protocol [28].
- **Mission Planner**: The Ardupilot mission (APM) planner is an open-source ground station application for MAVlink based autopilots [12]. It assists in mission planning using GPS way-points and control events.

### C. Geofencing Database

We used OpenStreetMap as the mapping source for underpinning our geofencing database. OpenStreetMap is an open-source database that supports WGS-84 coordinates practised by many GNSS units [29]. Geographical information about supported countries, e.g. USA and UK, can be exported in OSM format from the OpenStreetMap website. This data can be imported into a geographic information system application, like QGIS [30], for exporting maps into alternative file formats and coordinate systems, i.e. WGS-84 and OSGB-36.

For our experiments, we imported geographic information about our testing environment, located in the south east region of the United Kingdom, into QGIS (Fig. 9) which was subsequently exported to JSON format. This information contains features about notable physical locations, e.g. location type (military, airport, leisure etc.), its name, short description, and coordinates. Listing 1 shows a sample JSON entry. The features and coordinates were extracted from the file in Python for constructing the $\alpha$-shapes for each restricted location.

```
{'type': 'Feature', 'properties': { 'osm_id':
    '533025', 'osm_way_id': null, 'name': 'Canada
    Copse', 'type': 'multipolygon', 'aeroway': null,
     'amenity': null, 'admin_level': null, 'barrier'
    : null, 'boundary': null, 'building': null, '
    craft': null, 'geological': null, 'historic':
    null, 'land_area': null, 'landuse': 'forest', '
    leisure': null, 'man_made': null, 'military':
    null, 'natural': null, 'office': null, 'place':
    null, 'shop': null, 'sport': null, 'tourism':
    null, 'other_tags': null }, 'geometry': { 'type'
    : 'MultiPolygon', 'coordinates': --truncated--
```

Listing 1: Sample JSON from OpenStreetMap.

We note that, while OpenStreetMap is a widely used geographic database, it has several shortcomings. Its development is community-driven and may lack the maintenance and support of proprietary solutions. It also lacks information about ambient noise, pollution, altitude, and other environmental factors that may practically limit the operation of UAVs.

## V. Evaluation

After computing 1,309 geofences in the urban environment, it was revealed that the proposed method consistently and more tightly enclosed physical locations compared to dynamically drawn polygonal geofences (i.e. not with human input). Fig. 10 shows an example polygonal geofence, which

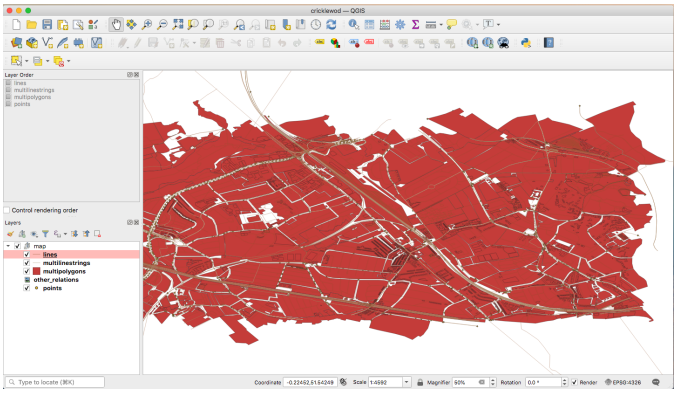

Fig. 9: QGIS visualisation.

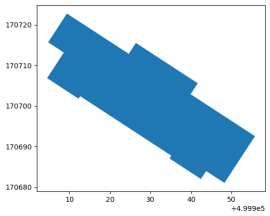


Fig. 10: Polygonal geofence.

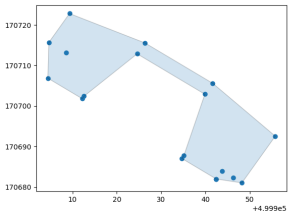


Fig. 11: Proposed method.

inaccurately enclosed a far greater physical area than our proposed method shown in Fig. 11. This process involved labour-intensive human assessment to evaluate the accuracy of both algorithms with respect to ground truth location data. The initial results are promising, but further experiments across varying geographic regions are warranted to reliably calculate the accuracy of our method.

The computational performance of both approaches was also evaluated at each workflow stage from from Section III. Fig. 12 shows the execution time of the proposal versus polygon-based geofencing using AirSim and our physical UAV platform at each stage (on the x-axis). After the initial computation of the geofences, the execution time for computing the geofences declines to under 500ms on our physical platform. Generally, the polygonal and proposed methods execute within the same order of magnitude—under three seconds for all phases—during operation. However, on average, the proposed method incurs an approximately one second overhead for geofence computation using the Navio2 UAV. Table I presents a comparison of the algorithms based on time and the evaluation platform. Compared to polygonal geofencing, our method requires an additional 1–2 seconds on both AirSim and the physical UAV to generated the geofenced areas. The average run-time execution time for detecting geofence violations is broadly the same: for AirSim, this was 0.006s *vs.* 0.006s (polygonal *vs.* proposed respectively) and, for Navio2, this was 0.269s *vs.* 0.270s.

In summary, the results demonstrate that our proposal has more reliable results than polygon geofencing in relation to geofencing accuracy. This comes with the drawback of

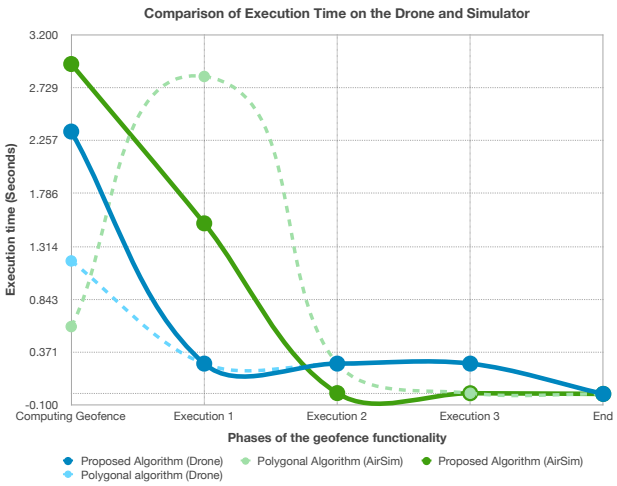


Fig. 12: Execution time of polygonal geofencing versus our proposed method on AirSim and the Navio2 UAV platform.

| **Comparison Stage Platform** | **Polygonal Algorithm** | **Proposed Algorithm** |
|---|---|---|
| AirSim geofence computation | 0.698 seconds | 3.44 seconds |
| AirSim average detection time | 0.006 seconds | 0.006 seconds |
| Navio2 geofence computation | 1.185 seconds | 2.338 seconds |
| Navio2 average detection time | 0.269 seconds | 0.270 seconds |

TABLE I: Average performance comparison.

a moderate geofence computation overhead, which can be partially mitigated by determining static $\alpha$-shape geofences before the flight commences. In general, our investigations suggest that the proposed algorithm provides greater accuracy for complex geofences without a dramatic performance impact in practice.

## VI. Conclusion

Existing geofencing algorithms can struggle with precisely enclosing restricted airspaces in dynamic environments; for example, when generated in-flight. This can pose safety and accessibility issues in complex urban areas with an abundance of closely located restricted locations, e.g. airports, schools and prisons, and permitted airspaces. To address this, we designed and evaluated a new geofencing solution for such scenarios using $\alpha$-shapes and Voronoi diagrams. We implemented our proposal in a simulated and real-world urban environment using a low-cost, commercially available drone platform. Our results suggest that greater geofencing precision can be achieved while retaining computational performance in the same order of magnitude as polygonal geofencing. We hope that our work provides UAV system designers with a useful option for generating accurate geofences in complex and dynamic operating environments.

## REFERENCES


[1] J. Cho and Y. Yoon, "How to assess the capacity of urban airspace: A topological approach using keep-in and keep-out geofence," *Transportation Research Part C: Emerging Technologies*, vol. 92, pp. 137–149, Jul. 2018.

[2] M. N. Stevens and E. M. Atkins, "Multi-mode guidance for an independent multicopter geofencing system," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, 2016, p. 3150.

[3] M. Stevens and E. Atkins, "Geofence definition and deconfliction for UAS traffic management," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[4] E. T. Dill, R. V. Gilabert, and S. S. Young, "SAFEGUARD: An assured safety net technology for UAS," in *IEEE/AIAA 37th Digital Avionics Systems Conference*. IEEE, 2018.

[5] DJI, "Fly Safe Drone Flying Tips, Policies & Regulations, and More DJI," 2020, https://www.dji.com/uk/flysafe [Accessed: 8th June 2021].

[6] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 551–559, 1983.

[7] N. Wawrzyniak and T. Hyla, "Application of geofencing technology for the purpose of spatial analyses in inland mobile navigation," in *2016 Baltic Geodetic Congress (BGC Geomatics)*. IEEE, 2016, pp. 34–39.

[8] F. Reclus and K. Drouard, "Geofencing for fleet and freight management," in *9th International Conference on Intelligent Transport Systems Telecommunications*. IEEE, 2009, pp. 353–356.

[9] J. Haofeng and G. Xiaorui, "Wi-Fi secure access control system based on geo-fence," in *IEEE Symposium on Computers and Communications*. IEEE, 2019, pp. 1–6.

[10] S. Johnson, A. Petzen, and D. Tokotch, "Exploration of detect-and-avoid and well-clear requirements for small UAS maneuvering in an urban environment," in *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017, p. 3074.

[11] NASA, "NASA Langley's Safeguard system for UAVs aims to take flight," 2017, https://www.nasa.gov/langley/business/feature/nasa-langley-s-safeguard-system-for-uavs-aims-to-take-flight/ [Accessed: 9th June 2021].

[12] ArduPilot Project, "ArduPilot," 2021, https://ardupilot.org/ [Accessed: 9th June 2021].

[13] Dronecode, "PX4 autopilot: Open source autopilot for drones," 2021, https://px4.io/ [Accessed: 9th June 2021].

[14] Cleanflight, "Cleanflight," 2021, http://cleanflight.com/ [Accessed: 9th June 2021].

[15] Emlid, "Navio2 autopilot HAT for Raspberry Pi," May 2020, https://emlid.com/navio [Accessed: 30th June 2021].

[16] T. Betaflight, "Betaflight," 2021, https://betaflight.com/ [Accessed: 9th June 2021].

[17] T. Gurriet and L. Ciarletta, "Towards a generic and modular geofencing strategy for civilian UAVs," in *International Conference on Unmanned Aircraft Systems*. IEEE, 2016, pp. 540–549.

[18] M. N. Stevens, H. Rastgoftar, and E. M. Atkins, "Specification and evaluation of geofence boundary violation detection algorithms," in *International Conference on Unmanned Aircraft Systems*. IEEE, 2017, pp. 1588–1596.

[19] F. Bélair, "Everything you always wanted to know about alpha shapes but were afraid to ask," 1998, http://cgm.cs.mcgill.ca/~godfried/teaching/projects97/belair/alpha.html [Accessed: 10th June 2021].

[20] Articque, "C&D and Articque Platform," 2020, https://www.articque.eu/news/new-features-february-18th-2020/ [Accessed: 8th June 2021].

[21] X. Chen and X. Chen, "The UAV dynamic path planning algorithm research based on Voronoi diagram," in *26th Chinese Control and Decision Conference*. IEEE, 2014, pp. 1069–1071.

[22] Y. V. Pehlivanoglu, "A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV," *Aerospace Science and Technology*, vol. 16, no. 1, pp. 47–55, 2012.

[23] S. A. Bortoff, "Path planning for UAVs," *Proceedings of the American Control Conference*, vol. 1, no. 6, p. 364, Oct. 2000.

[24] J. Castells, "Alpha shape with Voronoi diagram using R," 2012, https://jcastellssala.com/2012/04/16/shape-generation-with-r/alpha_shape_vornoi_01/ [Accessed; 9th June 2021].

[25] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*. Springer, 2018, pp. 621–635.

[26] K. E. Bellock, "Alpha shape toolbox," 2021, https://pypi.org/project/alphashape/ [Accessed: 5th July 2021].

[27] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," 2021, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed: 6th July 2021].

[28] Erle Robotics, "Introduction to Erle Robotics documentation," Feb. 2019, http://docs.erlerobotics.com/simulation/intro [Accessed: 2nd July 2021].

[29] OpenStreetMap contributors, "OSM SWOT – OpenStreetMap Wiki," Wikipedia, 2021, https://www.openstreetmap.org [Accessed: 5th July 2021].

[30] QGIS Project, "QGIS – A Free and Open Source Geographic Information System," 2021, http://qgis.osgeo.org [Accessed: 6th July 2021].