OFFSET ALGORITHM:

Obstacle detection & avoidance in high-speed UAVs

By ***Obiegba Onoteoghene Peter***

Student, Information Systems Department, Faculty of Computing, Federal University of Technology Akure

Email: obiegbaonospeter@gmail.com

Abstract

This research presents the Offset Algorithm, a novel lightweight obstacle avoidance technique for Unmanned Aerial Vehicles (UAVs) operating in dynamic environments. The system utilizes a multi-layered frame structure constructed from LIDAR scan data to evaluate obstacle-free paths in real time. It incorporates a scan window mechanism that calculates weighted scores from multiple layered frames (varying in scan radius) to make efficient pathfinding decisions with minimal computation. The architecture prioritizes closer obstacles while maintaining awareness of farther obstructions, enabling high responsiveness and safety in cluttered environments.

Introduction:

UAVs are increasingly deployed in dynamic, cluttered environments for surveillance, delivery, and disaster response. One of the most crucial challenges they face is real-time obstacle avoidance. Traditional computer vision-based methods are computationally intensive and struggle with low visibility conditions. Lightweight systems that can process LIDAR data efficiently are vital to ensure high-speed and reliable operation.

The Offset Algorithm is a spatial reasoning framework designed for efficient obstacle detection and avoidance in UAVs using LiDAR data. At its core, the algorithm constructs multiple frames, each corresponding to a unique scan radius. These frames segment the drone's field of view into binary arrays of nanoboxes, where 1 indicates an obstacle within the scan radius and 0 indicates free space. This layered approach allows the system to evaluate obstacle proximity hierarchically—closer obstacles are prioritized over farther ones.

To evaluate potential paths, the algorithm deploys scan windows, which are sliding blocks of fixed size (typically four nanoboxes) that move laterally across the frame from the center outwards. For each scan window position in every frame, a score is computed based on how many of the nanoboxes are free. These scores are weighted—frames closer to the UAV have exponentially greater influence, ensuring urgent threats dominate the decision-making process.

In each cycle, scan windows on both sides (left and right) are evaluated in parallel. Once a path with an aggregate score below a defined threshold is found, the system halts further scanning and returns the best direction. This early termination and weight-based logic make the Offset Algorithm fast, lightweight, and highly adaptable for real-time autonomous UAV applications.

Problem Statement and Limitations of Existing Systems:

1. **High Computation Cost:** Most real-time avoidance systems require dense 3D processing.
2. **Delayed Decision-Making:** Existing algorithms may process full-frame data before determining a path.
3. **Poor Prioritization of Threats:** Equal treatment of near and far obstacles can lead to inefficient rerouting.

Offset Algorithm addresses these issues by:

1. Using binary frame representations for fast computation.
2. Layering environmental awareness based on distance.
3. Assigning greater decision influence to closer obstacles.

Core Innovations:

The Offset Algorithm introduces the following innovations:

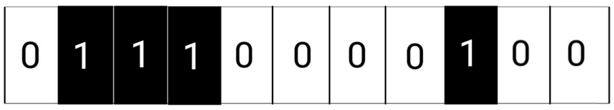
1. **Sliding Window with Early Termination:** Two simultaneous windows scan from the center outwards. As soon as a safe path is found (score under threshold), scanning stops.
2. **Multi-Radius Layered Frames:** Each LIDAR cycle is split into multiple frames of varying scan radii. This allows closer obstacles to be weighted more heavily while still considering farther obstacles.
3. **Geometric Weighting Strategy:** A simple exponential weighting system ensures near-frame influence dominates the aggregate score.
4. **Lightweight Aggregation:** The aggregation formula ensures rapid computation even with multiple frames.
5. **Unified Threshold-Based Decision:** A fixed threshold determines safe passage, simplifying the logic and improving reliability.

The Offset Algorithm: Scan Box Mechanics

The algorithm's core innovation lies in its dynamic scan box approach for obstacle detection. It systematically searches for collision-free paths by expanding two virtual boxes (left and right) from the drone's centerline, each cycle scanning deeper into the environment.

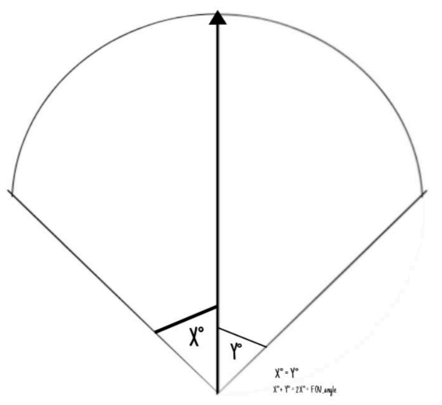
System Components and Core Innovations

1. **Box Configuration :**Fixed dimensions: 1m (width) × 2m (height), Initial position: Centered on UAV heading, Progressive expansion: Each cycle moves boxes laterally by a configurable k_step (e.g., 0.2m increments)
2. **Obstacle Check:** boxes convert to pixel regions in sensor data (e.g., 100×200px for 1cm resolution) , Binary evaluation: `True` if *all* pixels in the box are obstacle-free.
3. **Termination Logic:** Immediate exit when any box clears the obstacle check, Max range limit: Stops after reaching 50% of sensor range (e.g., 2.5m for 5m LIDAR)
4. **Frame (F):** A 1D array of binary nanoboxes (0 = free, 1 = blocked) representing beam status within a scan radius.



FRAME(F): 1D Array, 11 elements(nanoboxes)

5. **FOV (Field of View):** Angular range covered by LIDAR. Frame length is determined by the number of beams in this range.



Field Of View

6. **Scan Radius (R):** Maximum distance to consider for obstacles. Each frame has a unique scan radius.
7. **Scanbox:** A sub-array of N nanoboxes, typically 4, that defines the minimum fly-through space.
8. **Weight (W):** Assigned to each frame based on proximity. The closer the scan radius, the higher the weight: (Fn is the number of frames in the system)

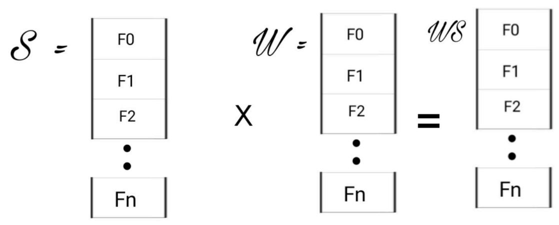
$$w_i = \frac{2^{F_n - i}}{2^{F_n} - 1}$$

9. **Scan Score (S):** Average obstacle score for a scanbox:

$$\text{ScanScore} = \sum_{i=1}^{N} \left(\frac{n_i \times 100}{N}\right)$$

10. **Weighted Score (Ws):** Weighted contribution of a scanbox score:

$$WS_i = S_i \times w_i$$



11. **Aggregate Score (A):**

$$A = \sum_{i=1}^{F_n} WS_i = \sum_{i=1}^{F_n} S_i \times w_i$$

Implementation Python:

The Offset Algorithm follows a systematic pipeline to determine a drone's safest path using LiDAR data. The process begins by generating or receiving raw LiDAR readings, which are fed into the Frame class. Each frame corresponds to a specific scan radius and transforms distance readings into binary arrays—1 for obstacles and 0 for free space—forming nanoboxes. Next, Scan Boxes are created by slicing 4 consecutive nanoboxes left and right from the drone's center field of view. These slices are passed to the Score Window function, which calculates a normalized score based on how free or blocked each scanbox is. Multiple scanboxes from different frames are then evaluated in parallel. The Aggregator applies weighted scores—giving precedence to closer frames—and computes a combined score. During the Decision Loop, the algorithm checks each offset position in both directions, comparing aggregate scores to a safety threshold, and selects the optimal path once found, halting further scans.

The system reads LiDAR data and processes it through multiple components:

- **Frame Class:** Constructs obstacle masks from distance values using defined scan radii.
- **Scan Box:** Samples 4-nanobox windows left and right from the drone's center position.
- **Score Window:** Scores the scanbox using normalized binary scores.
- **Aggregator:** Applies the frame weights and sums the scores to produce an aggregate score.
- **Decision Loop:** Performs parallel scanbox evaluation in cycles, stopping upon finding the best path.

Multi-Layer Parallel Scanning: Scan windows operate simultaneously in both left and right directions from the center of the frame. The system halts further scanning once a window meets or beats the threshold score.

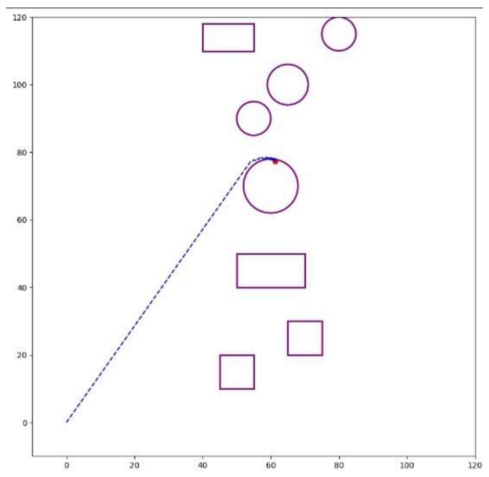
```
import random

class Offset_Algorithm:
    class Frame:
        def __init__(self, lidar_data,
scan_radius):
            self.R = scan_radius
            self.data =
self.create_nanobox_array(lidar_data)

        def create_nanobox_array(self,
lidar_data):
            return [
                1 if distance <= self.R else 0
                for _, distance in lidar_data
            ]

        def __str__(self):
            return f"Frame(R={self.R}):
{self.data}"

    def __init__(self, lidar_data, scan_radii,
threshold=5.0):
        self.lidar_data = lidar_data
        self.scan_radii = sorted(scan_radii)
        self.threshold = threshold
        self.frames = [self.Frame(lidar_data, r)
for r in self.scan_radii]
        self.mid = len(self.frames[0].data) // 2
        self.last_pos_L = self.mid + 2
        self.last_pos_R = self.mid - 1

    def score_window(self, slice_data):
        N = len(slice_data) or 1
        return sum((value * 100) // N for value
in slice_data)

    def aggregate_scores(self, scores):
        return sum(scores)
```

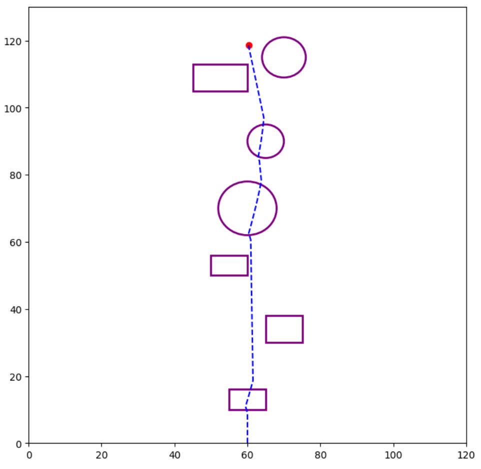
```
    def scan_box(self, array, current_pos,
direction="left"):
        if direction == "left":
            start = max(0, current_pos - 3)
            end = current_pos + 1
        else:  # "right"
            start = current_pos
            end = min(len(array), current_pos +
4)
        return array[start:end]

    def run(self):
        k = 0
        while k < self.mid:
            print(f"\n--- Cycle {k} ---")
            Wl_score, Wr_score = [], []

            for idx, frame in
enumerate(self.frames):
                scanbox_L =
self.scan_box(frame.data, self.last_pos_L,
"left")
                scanbox_R =
self.scan_box(frame.data, self.last_pos_R,
"right")

                frame_weight = 2 **
(len(self.frames) - idx - 1) / (2 **
len(self.frames) - 1)
                Fl_score =
self.score_window(scanbox_L) * frame_weight
                Fr_score =
self.score_window(scanbox_R) * frame_weight

                Wl_score.append(Fl_score)
                Wr_score.append(Fr_score)
```

```
                print(f"Frame R={frame.R:.1f} |
Left Window: {scanbox_L}, Score:
{int(Fl_score)}")
                print(f"Frame R={frame.R:.1f} |
Right Window: {scanbox_R}, Score:
{int(Fr_score)}")
                print(f"Frame{idx + 1} weight:
{frame_weight:.4f}")

            Wl_aggregate =
self.aggregate_scores(Wl_score)
            Wr_aggregate =
self.aggregate_scores(Wr_score)

            print(f"\nAggregate Left Score:
{Wl_aggregate}")
            print(f"Aggregate Right Score:
{Wr_aggregate}")

            # === Decision Section ===
            if k == 0 and Wl_aggregate <=
self.threshold and Wr_aggregate <=
self.threshold:
                print("=> Preferred Direction:
LEFT (first cycle tie, both meet threshold)")
                return "LEFT"

            if Wl_aggregate <= self.threshold and
Wr_aggregate <= self.threshold:
                direction = "LEFT" if
Wl_aggregate < Wr_aggregate else "RIGHT"
                print(f"=> Preferred Direction:
{direction} (both below threshold,
{direction.lower()} is better)")
                return direction
            elif Wl_aggregate <= self.threshold:
                print("=> Preferred Direction:
LEFT (only left meets threshold)")
                return "LEFT"
```

```
            elif Wr_aggregate <= self.threshold:
                print("=> Preferred Direction:
RIGHT (only right meets threshold)")
                return "RIGHT"
            else:
                print("=> No direction found —
both exceed threshold")

            self.last_pos_L -= 1
            self.last_pos_R += 1
            k += 1

        print("=> Exhausted all scan positions
without a clear path")
        return None

# Generate sample lidar data

def generate_lidar_data(radius=7.0,
angle_range=(-60, 60), step=2):
    lidar_data = []
    angle = angle_range[0]
    while angle <= angle_range[1]:
        distance = round(random.uniform(0.5,
radius), 2)
        lidar_data.append((angle, distance))
        angle += step
    return lidar_data

# Example usage
lidar_data = generate_lidar_data()
scan_radii = [1.0, 2.0, 3.0, 0.5, 5.0, 15.0,
20.0]
thresh = 5.0

oa = Offset_Algorithm(lidar_data, scan_radii,
thresh)
decision = oa.run()
print(f"Final Decision: {decision}")
```

Research Data:





Conclusion

The Offset Algorithm delivers a computationally lightweight, effective obstacle avoidance system by layering multiple frames with prioritized weighting. Its strength lies in its simplicity and deterministic logic, making it suitable for low-resource embedded UAVs. The design is modular, allowing for future integration with swarm intelligence systems, temporal memory models, or reactive escape behaviors.

This research lays the groundwork for a UAV navigation system that emphasizes speed, efficiency, and adaptiveness. Future iterations will focus on integrating partial scanning fallback threads and dynamic weight tuning based on terrain type or UAV speed.