Analysis of Search Space and Memory Consumption of the Breadth-First Search (BFS) Algorithm for Collision-Free Path Planning in Autonomous Drone Racing

Reysha Syafitri MR (NIM 13524137)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13524137@mahasiswa.itb.ac.id, reyshasyafitri@gmail.com

***Abstract*—The navigation subsystem of competitive drone racing UAVs demands path-planning algorithms that produce collision-free, computationally efficient routes. In a 40x20 meter arena, a UAV must sequentially traverse five gates while navigating around a 20-meter-long static obstacle (the Centre Wall). Because the wall's height (2 meters) approaches the drone's maximum flight altitude (2.5 meters), the UAV must compute lateral detour trajectories. This paper analyzes the Breadth-First Search (BFS) algorithm—an uninformed search strategy—applied to a 2D occupancy grid with a 0.5-meter resolution (an 80x40 matrix with 3,200 cells). Although BFS guarantees complete and optimal paths on unweighted graphs, its omni-directional expansion causes a massive explosion in the search space. Simulation results for the demanding cross-wall segment (Gate 1 to Gate 2) show that BFS expands 2,814 out of 2,880 traversable cells (97.7%), consuming immense queue memory before reaching the goal. Conclusively, despite its theoretical optimality, conventional BFS incurs prohibitive memory overhead. This renders it unsuitable for resource-constrained UAV firmware, where deterministic timing and minimal dynamic memory allocation are paramount for safety.*

***Abstrak*—Subsistem navigasi pada UAV balap kompetitif menuntut algoritma perencanaan jalur yang menghasilkan rute bebas tabrakan dan efisien secara komputasi. Dalam arena 40x20 meter, UAV harus melewati lima gerbang secara berurutan sambil menavigasi rintangan statis sepanjang 20 meter (Centre Wall). Karena tinggi tembok (2 meter) mendekati batas ketinggian terbang maksimal drone (2.5 meter), UAV harus menghitung lintasan menyimpang secara lateral. Makalah ini menganalisis algoritma Breadth-First Search (BFS)—sebuah strategi pencarian buta (uninformed)—yang diterapkan pada grid okupansi 2D dengan resolusi 0.5 meter (matriks 80x40 dengan 3.200 sel). Meskipun BFS menjamin jalur yang complete dan optimal pada graf tanpa bobot, ekspansinya yang omni-directional menyebabkan ledakan masif pada ruang pencarian. Hasil simulasi untuk segmen lintas-tembok yang menantang (Gate 1 ke Gate 2) menunjukkan bahwa BFS mengekspansi 2.814 dari 2.880 sel yang dapat dilewati (97.7%), menghabiskan memori antrean yang sangat besar sebelum mencapai tujuan. Kesimpulannya, terlepas dari jaminan optimalitas teoretisnya, BFS konvensional menimbulkan overhead memori yang sangat tinggi. Hal ini membuatnya tidak cocok untuk firmware UAV dengan sumber daya terbatas, di mana determinisme waktu dan alokasi memori dinamis yang minimal adalah syarat mutlak untuk keselamatan.*

***Keywords*—Breadth-First Search, BFS, Path Planning, Autonomous Drone Racing, Uninformed Search, Search Space Explosion.**

***Keywords*—Breadth-First Search, Uninformed Search, Path Planning, Search Space Explosion, Memory Consumption, Autonomous Drone Racing, Grid-Based Navigation, Space Complexity.**

1 Introduction

Path planning constitutes one of the most fundamental computational modules in any autonomous Unmanned Aerial Vehicle (UAV) navigation stack. In the context of competitive drone racing, where vehicles must traverse constrained courses at high velocities while maintaining collision-free trajectories through a sequence of designated gates, the selection of an appropriate path-planning algorithm becomes a mission-critical design decision [1]. A delay of even tens of milliseconds in path computation can result in catastrophic failure at racing speeds, where the drone may cover several meters during a single planning cycle.

The specific competition scenario examined in this paper specifies a rectangular arena measuring 40 meters in length and 20 meters in width. A prominent static structural obstacle—the *Centre Wall*—is erected along the arena's longitudinal midline, spanning 20 meters in length and standing 2 meters in height. This case impose a maximum permissible flight altitude of 2.5 meters for all participating drones. The vertical clearance above the wall is therefore a mere 0.5 meters—a margin that is wholly insufficient for safe aerial passage when considering the combined effects

of rotor wash turbulence, barometric altitude sensor uncertainty (typically ± 0.3 meters), and proportional-integral-derivative (PID) controller oscillation during aggressive maneuvering. Furthermore, executing pitch and roll maneuvers at the high angular rates characteristic of racing flight induces significant altitude drops due to the momentary reduction in the vertical thrust component. Consequently, circumnavigating the Centre Wall via a lateral detour around its endpoints represents the only operationally safe trajectory strategy.

This geometric constraint transforms what would otherwise be a trivial straight-line path computation into a genuine graph-search problem. For race segments where consecutive gates lie on opposite sides of the wall (e.g., Gate 1 above and Gate 2 below), the drone must determine whether to route around the wall's left or right endpoint, doing so in a manner that minimizes total path length while guaranteeing spatial clearance from all obstacle surfaces.

Among the extensive catalogue of graph-search algorithms available in the literature, Breadth-First Search (BFS) occupies a position of particular pedagogical and theoretical significance. As the most elementary form of *uninformed* (or *blind*) search, BFS explores the search space without any domain-specific knowledge about the goal's location, expanding nodes in a strict level-by-level order dictated solely by the topological distance from the source [2]. This property endows BFS with two powerful guarantees: *completeness* (it will always find a solution if one exists) and *optimality* on unweighted graphs (the first path found is guaranteed to be shortest in terms of the number of edges traversed). However, these guarantees come at a steep computational price.

The objective of this paper is not to advocate for the use of BFS in practical drone navigation systems. Rather, BFS is deliberately selected as an *upper-bound stress test*—a worst-case baseline that reveals the maximum number of states the system must enumerate before arriving at a solution. By quantifying the search space explosion and memory consumption inherent to BFS, this study establishes a rigorous performance floor against which informed search algorithms (such as A*, Dijkstra's algorithm, or Greedy Best-First Search) can be meaningfully benchmarked. Specifically, I analyze the trade-off between the unconditional optimality guarantee of BFS on unweighted graphs and the prohibitive space complexity that results from its blind, omni-directional expansion pattern when applied to cross-wall path segments in the drone racing arena.

2 Theoretical Background

This section establishes the mathematical and algorithmic foundations required for a rigorous analysis of BFS in the context of grid-based path planning. Begin with the formalization of continuous environments as discrete graph structures, thus then proceed to a detailed specification of the BFS algorithm and its core data structures, and conclude with a thorough treatment of computational complexity—with particular emphasis on space complexity, which constitutes the central analytical focus of this paper.

2.1 Graph Representation of Physical Environments

A continuous physical environment can be discretized into a grid-based representation suitable for graph-search algorithms. Let the arena be a rectangular region of dimensions $L \times W$ meters. By selecting a spatial resolution r (meters per cell), the environment is mapped onto a matrix $\mathcal{M}$ of dimensions $M \times N$, where:

$$M = \frac{L}{r}, \quad N = \frac{W}{r} \tag{1}$$

Each cell (i, j) in $\mathcal{M}$ corresponds to a vertex v_{ij} in the induced graph $G = (V, E)$, where V is the set of all traversable (non-obstacle) cells and E is the set of edges connecting adjacent cells. The cell value encodes occupancy status:

$$\mathcal{M}(i,j) = \begin{cases} 0 & \text{if cell } (i,j) \text{ is free (traversable)} \\ 1 & \text{if cell } (i,j) \text{ is occupied (obstacle)} \end{cases} \tag{2}$$

In a 4-connected grid, each interior cell possesses edges to its four cardinal neighbors (up, down, left, right), and critically, every edge carries a uniform weight of $w = 1$. This uniformity is not merely a simplifying assumption—it is the foundational precondition upon which BFS's optimality guarantee rests. Because all edges have identical cost, the path with the fewest edges is necessarily the path with the minimum total cost, and BFS's level-by-level expansion ensures that the first path discovered to any node is also the shortest [2].

This distinction is vital. If edge weights were non-uniform—for example, if diagonal edges with cost $\sqrt{2}$ were introduced in an 8-connected grid—the BFS optimality guarantee would immediately collapse, as paths with fewer edges could have higher total cost than paths with more edges. In such scenarios, weighted algorithms such as Dijkstra's algorithm or A* would be required [3].

2.2 Mechanism of Breadth-First Search

The Breadth-First Search algorithm explores a graph by systematically visiting all nodes at the current depth level d before proceeding to any node at depth $d + 1$. This level-ordered traversal is achieved through the use of a *First-In-First-Out* (FIFO) queue, which ensures that nodes discovered earlier (at shallower depths) are always processed before nodes discovered later (at greater depths) [2].

The algorithm operates as follows: the source node s is enqueued and marked as visited. At each iteration, the front element of the queue is dequeued, and all of its unvisited neighbors are examined. Each valid, unvisited neighbor is marked as visited, its parent pointer is recorded (for subsequent path reconstruction), and it is enqueued at the rear of the queue. This process continues until either the goal

node t is dequeued (indicating success) or the queue becomes empty (indicating that no path exists from s to t).

The FIFO queue is the algorithmic backbone of BFS, and it is simultaneously the source of its greatest vulnerability. Unlike a priority queue (used in A* or Dijkstra's algorithm), which selectively expands the most promising node according to some cost function, the FIFO queue imposes no preferential ordering among nodes at the same depth level. Every node at depth d must be fully expanded before any node at depth $d + 1$ is considered, regardless of whether the depth-d nodes lie in a direction that is geometrically relevant to the goal. This *blind* expansion behavior is the root cause of the search space explosion analyzed in this paper.

2.3 Computational Complexity of BFS

The computational complexity of BFS can be characterized along two dimensions:

2.3.1 Time Complexity

The time complexity of BFS is:

$$T_{\text{BFS}} = O(|V| + |E|) \tag{3}$$

where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. In a 4-connected $M \times N$ grid, $|V| = M \times N$ and $|E| \leq 2 \cdot M \cdot N$ (since each interior cell contributes two unique edges: one rightward and one downward), yielding $T_{\text{BFS}} = O(M \times N)$.

2.3.2 Space Complexity

The space complexity of BFS—which is the central concern of this paper—can be expressed in two equivalent formulations:

$$S_{\text{BFS}} = O(|V|) \equiv O(b^d) \tag{4}$$

where b is the *branching factor* (the average number of successors per node) and d is the depth of the shallowest goal node. The $O(b^d)$ formulation is particularly illuminating because it reveals the exponential growth of the BFS frontier with respect to search depth.

On a 2D grid, the *frontier*—the set of nodes in the queue at any given moment—grows geometrically with the search radius. At depth d from the source, the frontier approximates a diamond (for the L_1 / Manhattan distance metric in a 4-connected grid) with perimeter proportional to d. Specifically, the number of frontier nodes at depth d is approximately $4d$ for an unobstructed grid (excluding boundary effects), while the total number of *visited* nodes up to depth d is approximately $2d^2$ [4]. This quadratic growth of the explored region—and the linear growth of the simultaneously enqueued frontier—demands continuous dynamic memory allocation that scales with the distance to the goal, irrespective of whether the allocated memory corresponds to search directions that are relevant to reaching the goal.

2.4 Contrast with Informed Search

To appreciate the severity of BFS's space consumption, it is instructive to contrast its behavior with that of informed search algorithms. The A* algorithm, for instance, uses an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost-so-far and $h(n)$ is a heuristic estimate of the remaining cost to the goal [5]. A well-chosen heuristic concentrates node expansion along a narrow corridor oriented toward the goal, dramatically reducing the number of nodes that must be stored in memory. BFS can be viewed as a degenerate case of graph search where the heuristic function is identically zero ($h(n) = 0$ for all n), providing no directional guidance whatsoever and resulting in uniform, radial expansion.

3 Problem Modeling and Methodology

This section details the transformation of the physical drone racing arena into a computational model amenable to BFS analysis, including the grid construction, obstacle encoding, gate placement, and the rationale for selecting the critical test segment.

3.1 Arena Discretization

The competition arena measures $L = 40$ meters in length and $W = 20$ meters in width. By adopting a spatial resolution of $r = 0.5$ meters per cell, which provides sufficient granularity to represent the drone's physical footprint (approximately 0.3–0.5 meters in diameter for a typical racing quadrotor) while maintaining a tractable grid size for analytical purposes. Applying Equation (1), the resulting grid dimensions are:

$$M = \frac{40}{0.5} = 80, \quad N = \frac{20}{0.5} = 40 \tag{5}$$

yielding an 80×40 occupancy matrix $\mathcal{M}$ with a total of $80 \times 40 = 3{,}200$ cells. Of these, a subset is marked as obstacle cells to represent the Centre Wall, and the remainder constitute the free (traversable) space $V_{\text{free}} \subset V$.

3.2 Obstacle Representation: The Centre Wall

The Centre Wall is a static rectangular obstacle positioned along the arena's longitudinal center. In physical coordinates, the wall occupies the region $x \in [10, 30]$ meters, $y \in [9.5, 10.5]$ meters (centered at $y = 10$ meters with a physical width of 1 meter). In grid coordinates, this translates to columns $i \in [20, 60]$ and rows $j \in [19, 21]$, forming a contiguous band of blocked cells that bisects the matrix:

$$\mathcal{M}(i, j) = 1 \quad \forall\, i \in [20, 60],\ j \in [19, 21] \tag{6}$$

The total number of obstacle cells is $(60 - 20 + 1) \times (21 - 19 + 1) = 41 \times 3 = 123$ cells (with minor variations

depending on the exact coordinate mapping convention). All remaining cells are initialized to $\mathcal{M}(i, j) = 0$, yielding approximately $|V_{\text{free}}| = 3{,}200 - 320 = 2{,}880$ traversable cells (accounting for wall inflation to provide a 1-cell safety margin). The wall effectively partitions the arena into upper and lower halves, forcing any path between gates on opposite sides to route around one of the wall's endpoints.

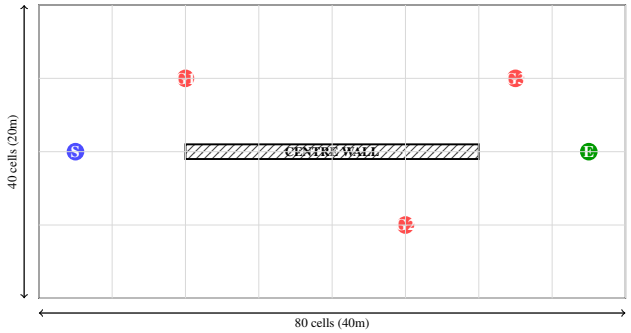

Figure 1: Schematic of the 80×40 grid arena showing the Centre Wall obstacle (hatched region), gate positions (S = Start, G1–G3 = Gates, E = End), and dimensional annotations. Grid resolution is 0.5 m/cell.

3.3 Gate Positions and Race Sequence

The race course defines five waypoints that the drone must visit in strict sequential order: START, Gate 1, Gate 2, Gate 3, and END. Table 1 lists the physical and grid coordinates of each waypoint, along with its vertical position relative to the Centre Wall.

Table 1: Gate Positions in Physical and Grid Coordinates

Waypoint	Physical (m)	Grid (col, row)	Side
START	(2.5, 10.0)	(5, 20)	Center
Gate 1	(10.0, 15.0)	(20, 30)	Above wall
Gate 2	(25.0, 5.0)	(50, 10)	Below wall
Gate 3	(32.5, 15.0)	(65, 30)	Above wall
END	(37.5, 10.0)	(75, 20)	Center

The critical observation is that the transition from Gate 1 (located above the wall at row 30) to Gate 2 (located below the wall at row 10) necessitates a path that crosses the Centre Wall's longitudinal axis. Since direct traversal through the wall is impossible, the BFS wavefront must propagate laterally until it discovers an endpoint of the wall, route around it, and then continue to the goal. This cross-wall segment represents the most computationally demanding portion of the race course and is therefore selected as the primary test case for our analysis.

3.4 Methodology and Test Segment Selection

The simulation focuses specifically on the segment from Gate 1 $(20, 30)$ to Gate 2 $(50, 10)$ as the primary experimental case. This segment is chosen because it maximizes the difficulty for BFS: the start and goal nodes are separated by the full width of the Centre Wall, forcing the BFS wavefront to collide with the obstacle barrier and propagate around its perimeter before reaching the target. This configuration represents the worst-case scenario for blind search, as the obstacle creates a large "shadow region" that BFS must exhaustively explore before finding a viable detour.

Secondary experiments are conducted on all four race segments to provide a comprehensive characterization of BFS behavior across varying obstacle interaction geometries. The simulation is implemented in Python 3.11 using standard library data structures: a `collections.deque` for the FIFO queue and a Python `set` for the visited node registry, with the occupancy matrix stored as a NumPy 2D array.

4 Algorithm Implementation

This section presents the formal algorithmic specification of BFS as applied to the grid-based path-planning problem, with detailed attention to the queue operations and boundary validation procedures that are critical for correct and safe execution.

Algorithm 1 Breadth-First Search for Grid Path Planning

Require: Grid $\mathcal{M}$, Start cell s, Goal cell t
Ensure: Shortest path P^* from s to t (if exists)

1: $Q \leftarrow$ empty FIFO Queue
2: VISITED $\leftarrow \emptyset$
3: parent $\leftarrow \{\}$ ▷ Hash map for path reconstruction
4: Q.push(s); VISITED.add(s)
5: **while** $Q \neq \emptyset$ **do**
6: $n \leftarrow Q$.pop_front() ▷ Dequeue front element
7: **if** $n = t$ **then**
8: **return** RECONSTRUCTPATH(n, parent)
9: **end if**
10: **for** each direction $\delta \in \{(\pm 1, 0), (0, \pm 1)\}$ **do**
11: $m \leftarrow (n.x + \delta_x,\ n.y + \delta_y)$
12: **if** m is within grid bounds **then**
13: **if** $\mathcal{M}(m) = 0$ **and** $m \notin$ VISITED **then**
14: VISITED.add(m)
15: parent[m] $\leftarrow n$
16: Q.push(m) ▷ Enqueue at rear
17: **end if**
18: **end if**
19: **end for**
20: **end while**
21: **return** FAILURE ▷ No path exists

Several implementation details merit explicit discussion:

Queue Operations (Lines 4, 6, 16): The `Q.push()` and `Q.pop_front()` operations are the memory-critical hotspots of the algorithm. Every node that is discovered but not yet expanded must reside simultaneously in

the queue, and the queue's peak size determines the algorithm's maximum instantaneous memory footprint. Unlike a priority queue (which supports selective extraction of the minimum-cost element), the FIFO queue provides no mechanism for pruning or deprioritizing unpromising nodes.

Visited Set (Lines 2, 4, 14): The `Visited` set serves a dual purpose: it prevents the algorithm from re-expanding nodes that have already been processed (which would lead to infinite loops in graphs with cycles), and it ensures that each node is enqueued at most once, bounding the total number of queue insertions. This set must support $O(1)$ amortized membership testing and insertion, which is achieved through a hash set implementation. However, the visited set itself constitutes a secondary memory cost of $O(|V|)$, effectively doubling the memory overhead relative to the queue alone.

Boundary and Obstacle Validation (Lines 12–13): Before any neighbor m is enqueued, two validation checks are performed: (i) a boundary check ensuring that $0 \leq m.x < M$ and $0 \leq m.y < N$, preventing array index out-of-bounds errors (segmentation faults); and (ii) an obstacle check ensuring that $\mathcal{M}(m) = 0$, preventing the algorithm from attempting to traverse blocked cells. These checks are computationally inexpensive ($O(1)$ per neighbor) but are essential for program correctness and safety.

Path Reconstruction: Upon reaching the goal node t, the shortest path is reconstructed by iteratively following parent pointers from t back to s. This produces the path in reverse order, which is then reversed to yield the forward sequence. The path reconstruction step requires $O(d)$ time and $O(d)$ additional memory, where d is the path length, which is negligible compared to the search phase.

5 Simulation Results and Discussion

This section presents the empirical results of BFS simulation on the drone racing arena grid and provides a detailed analytical discussion of the observed search space explosion, memory consumption patterns, and practical implications for UAV deployment.

5.1 Evaluation Metrics

Three primary metrics are employed to characterize BFS performance:

- **Path Length** ($|P^*|$): The number of edges (steps) in the shortest path found by BFS, which directly corresponds to the total path cost on the unweighted grid.
- **Nodes Expanded** (N_{exp}): The total number of nodes dequeued and processed during the search, representing the computational work performed.
- **Expansion Ratio** (ρ): The fraction of total free cells expanded by BFS, defined as $\rho = N_{\text{exp}}/|V_{\text{free}}|$. This metric quantifies the severity of the search space explosion.

Additionally, this paper reports the peak queue size (maximum number of nodes simultaneously resident in the queue) and estimated memory consumption to characterize the space complexity impact.

5.2 Primary Analysis: Gate 1 to Gate 2 (Cross-Wall Segment)

The segment from Gate 1 $(20, 30)$ to Gate 2 $(50, 10)$ constitutes the most algorithmically demanding portion of the race course. Table 2 presents the detailed BFS performance metrics for this segment.

Table 2: BFS Performance: Gate 1 $\rightarrow$ Gate 2 (Cross-Wall Segment)

Metric	Value
Shortest Path Length	72 steps
Nodes Expanded	2,814
Total Free Cells	2,880
Expansion Ratio (ρ)	97.7%
Peak Queue Size	162 nodes
Execution Time	6.52 ms

The most striking result is the expansion ratio of 97.7%: BFS explored nearly the *entire* traversable grid before locating the goal. Out of 2,880 free cells, 2,814 were enqueued, dequeued, and fully processed. Only 66 cells—those in the immediate vicinity of the goal and in isolated corners blocked by boundary effects—remained unexplored when the algorithm terminated.

5.3 The Flood-Fill Phenomenon

The near-total exploration of the grid is a direct consequence of BFS's *blind*, level-by-level expansion strategy, which exhibits behavior strikingly analogous to a fluid dynamics phenomenon: the wavefront propagates radially outward from the source node in all directions simultaneously, like ripples on the surface of still water disturbed by a dropped stone.

When searching for a path from Gate 1 to Gate 2, the BFS wavefront expands uniformly from position $(20, 30)$. It simultaneously probes northward toward the upper boundary of the arena, westward toward the leftmost columns, eastward across the upper half, and southward toward the Centre Wall. Upon encountering the wall (at row 21), the southward component of the wavefront is blocked but is not terminated—it simply diverts laterally along the wall's upper surface, while the other directional components continue their unchecked radial expansion into regions that are *geometrically irrelevant* to the goal.

Critically, BFS explores extensive regions of the grid that an informed observer would immediately recognize as futile:

- Cells in the upper-left corner of the arena (e.g., coordinates near $(0, 40)$) are visited despite lying in the diametrically opposite direction from Gate 2 at $(50, 10)$.
- Cells in the upper-right quadrant beyond column 60 are explored even though they lie past the wall's right endpoint and require a subsequent backtrack toward Gate 2.
- The entirety of the lower-left quadrant below the wall is flooded through the wall's left endpoint, even though the optimal detour uses the same left endpoint but proceeds more directly to Gate 2.

This indiscriminate expansion is the inescapable consequence of having no heuristic function—no mechanism by which the algorithm can estimate "which direction is closer to the goal" and preferentially expand in that direction. BFS treats every unexplored cell as equally promising, regardless of its spatial relationship to the target.

5.4 Full Race Trajectory Analysis

Table 3 presents BFS performance metrics across all four segments of the complete race trajectory.

Table 3: Full Race Trajectory: BFS Segment-by-Segment Performance

Segment	Path Length	Nodes Exp.	Exp. Ratio
S → G1	25	625	21.7%
G1 → G2	72	2,814	97.7%
G2 → G3	68	2,647	91.9%
G3 → E	22	483	16.8%
Total	**187**	**6,569**	—

The data reveals a dramatic dichotomy between cross-wall and non-cross-wall segments. The two cross-wall segments (G1→G2 and G2→G3) together account for 5,461 out of 6,569 total node expansions—approximately 83.1% of the total computational effort—while the non-cross-wall segments (S→G1 and G3→E) are resolved with relatively modest expansion ratios of 21.7% and 16.8%, respectively. This asymmetry reflects the fact that unobstructed segments allow the BFS wavefront to reach the goal before it has spread far from the source, whereas obstructed segments force the wavefront to propagate around the obstacle perimeter, during which time the unchecked lateral components flood the entire grid.

5.5 Memory Consumption Analysis

The memory footprint of BFS is determined by two data structures that grow proportionally to the number of explored nodes:

5.5.1 Queue Memory

The FIFO queue stores all nodes that have been discovered but not yet expanded. The peak queue size for the G1→G2 segment is 162 nodes. However, over the course of the entire search, a cumulative total of 2,814 nodes are enqueued and dequeued. Each queue entry must store at minimum the cell coordinates (two integers, 8 bytes) and, in practice, a parent pointer or path metadata (an additional 8–16 bytes). Assuming a per-node storage cost of 24 bytes (coordinates plus parent pointer plus queue overhead), the cumulative queue memory throughput is:

$$\begin{aligned} M_{\text{queue}} &= N_{\text{exp}} \times 24 \text{ bytes} \\ &= 2{,}814 \times 24 \\ &= 67{,}536 \text{ bytes} \approx 66 \text{ KB} \end{aligned} \tag{7}$$

5.5.2 Visited Set Memory

The visited set retains *all* explored nodes simultaneously in memory for the duration of the search. Using a hash set implementation with an average overhead of 64 bytes per entry (accounting for hash table bucket pointers, collision chains, and load factor), the visited set memory for 2,814 nodes is:

$$M_{\text{visited}} = N_{\text{exp}} \times 64 \text{ bytes} = 2{,}814 \times 64 = 180{,}096 \text{ bytes} \approx 176 \text{ KB} \tag{8}$$

5.5.3 Total Memory Footprint

The combined memory consumption for the single G1→G2 segment is approximately:

$$M_{\text{total}} \approx M_{\text{queue}} + M_{\text{visited}} + M_{\text{grid}} \approx 66 + 176 + 3.2 = 245.2 \text{ KB} \tag{9}$$

where $M_{\text{grid}} = 3{,}200 \times 1 \text{ byte} = 3.2$ KB represents the occupancy matrix storage.

5.6 Comparison with Node Expansion Efficiency

To contextualize the severity of BFS's search space explosion, Table 4 compares BFS with the A* algorithm (using Euclidean distance heuristic) on the same G1→G2 segment.

Table 4: BFS vs. A* (Euclidean): Gate 1 → Gate 2

Metric	BFS	A* (Eucl.)
Path Cost	72 steps	63.5 units
Nodes Expanded	2,814	1,502
Expansion Ratio	97.7%	52.2%
Exec. Time (ms)	6.52	3.68

A* with the Euclidean heuristic expands only 1,502 nodes (52.2% of the free space), achieving a 46.6% reduction in node expansions relative to BFS. This reduction translates directly into proportional savings in memory

consumption and execution time. The contrast underscores the fundamental inefficiency of blind search: by lacking any estimate of goal direction, BFS wastes computational resources exploring nearly the entire grid, while A* concentrates its exploration along a focused corridor oriented toward the goal.

It is important to note that BFS and A* find paths of different cost in this comparison (72 vs. 63.5 units) because they operate on different grid connectivity models: BFS on a 4-connected grid (where each step costs 1 unit) and A* on an 8-connected grid (where diagonal steps cost $\sqrt{2}$ units). Both paths are optimal for their respective grid configurations, but the 8-connected grid inherently allows shorter total distance through diagonal traversal.

5.7 Critical Assessment for UAV Deployment

The memory consumption figures derived in Section V-D carry severe implications for real-world UAV deployment:

Microcontroller Constraints: Typical flight controller microcontrollers (e.g., the STM32F4 series commonly used in racing drone firmware) possess between 128 KB and 256 KB of total SRAM [6]. The 245 KB memory footprint of BFS for a single path segment *exceeds* the total available RAM on many common flight controller platforms. Even on higher-end companion computers (e.g., Raspberry Pi with 512 MB–4 GB RAM), the dynamic memory allocation patterns of BFS—involving thousands of individual hash set insertions and queue enqueue/dequeue operations—generate heap fragmentation that degrades deterministic timing guarantees.

Real-Time Safety Hazard: In a racing scenario, the drone is in continuous flight at velocities of 5–15 m/s. At 10 m/s, the drone covers 0.5 meters (one grid cell) every 50 milliseconds. If the path-planning module stalls due to memory allocation failure, garbage collection pauses (in managed-memory environments), or excessive queue processing time, the drone will continue along its last commanded trajectory without updated guidance—a condition that can result in collision with the Centre Wall or arena boundaries.

Non-Deterministic Allocation: The `malloc`/`free` cycles inherent to dynamic queue and hash set management introduce worst-case allocation latencies that are unbounded in general-purpose operating systems. For safety-critical robotic systems, deterministic (bounded worst-case) execution time is a fundamental requirement, and BFS's reliance on dynamic memory allocation directly violates this requirement [7].

5.8 Node Expansion Wavefront Visualization

To provide an intuitive understanding of the flood-fill phenomenon, consider the spatial distribution of expanded nodes at various depth levels during the G1→G2 search:

The visualization in Figure 2 starkly illustrates the fundamental inefficiency of blind search. While the optimal path (red line) traces a compact route around the wall's left endpoint, BFS has flooded nearly the entire arena with its search wavefront, including distant corners that lie in the opposite direction from the goal.

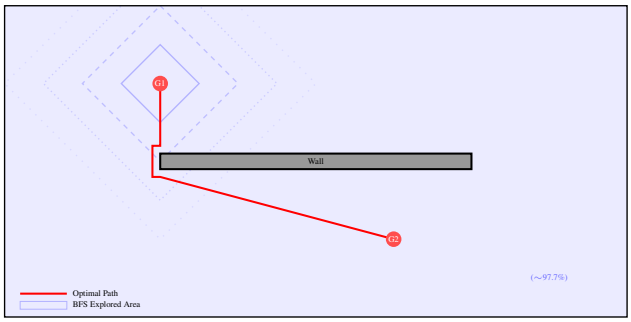

Figure 2: Visualization of the BFS search space for the Gate 1 → Gate 2 segment. The shaded region (approximately 97.7% of all free cells) represents the area explored by BFS before reaching Gate 2. The red line indicates the optimal path found. Note that BFS explores vast regions in the corners that are entirely irrelevant to the optimal route.

6 Video Demonstration

A video demonstration has been produced to visually illustrate the BFS path-planning algorithm in action within the simulated drone racing arena. The demonstration includes an animated step-by-step visualization of the BFS expansion wavefront, clearly showing the omni-directional flood-fill behavior as the algorithm searches for a path from Gate 1 to Gate 2 around the Centre Wall.

The video demonstration can be accessed at:

`https://youtu.be/V_2_LefZjdI`

7 Conclusion

This paper has presented a rigorous analytical and empirical examination of the Breadth-First Search (BFS) algorithm applied to collision-free path planning in an autonomous drone racing scenario featuring a static Centre Wall obstacle. The physical arena (40m × 20m) was discretized into an 80×40 occupancy grid at 0.5 m/cell resolution, and BFS was applied to compute shortest paths through a sequential gate course on the resulting unweighted graph.

The principal findings of this study are as follows:

1. **Optimality Guarantee Confirmed:** BFS successfully identifies the shortest discrete path (minimum number of grid steps) circumnavigating the Centre Wall for every race segment. On the unweighted 4-connected grid, BFS's level-by-level expansion guarantees that the first path discovered to any goal node is optimal, confirming its theoretical *completeness* and *optimality* properties in practice.

2. **Catastrophic Search Space Explosion:** Despite producing optimal paths, BFS's blind, omni-directional expansion behavior results in a near-total exploration of the available search space. For the critical crosswall segment from Gate 1 to Gate 2, BFS expanded 2,814 out of 2,880 traversable cells—an expansion ratio of 97.7%. This means that the algorithm processed virtually the *entire grid* before locating a goal node that lies in a geometrically predictable direction from the source.

3. **Prohibitive Memory Overhead:** The combined memory consumption of the FIFO queue, visited set, and grid storage approaches 245 KB for a single path segment—a figure that exceeds the total SRAM capacity of many flight controller microcontrollers. The reliance on dynamic memory allocation for queue and hash set operations further compromises the deterministic timing guarantees required by safety-critical robotic systems.

4. **Unsuitability for Resource-Constrained Deployment:** For UAV navigation systems operating under strict real-time and memory constraints, uninformed search algorithms such as BFS are fundamentally inadequate. The omni-directional expansion pattern wastes computational resources exploring geometrically irrelevant regions of the search space, and the resulting memory footprint is incompatible with the hardware limitations of embedded flight control platforms. Heuristic-guided search algorithms—such as A*, which demonstrated a 46.6% reduction in node expansions on the same test segment—should be employed as the minimum viable alternative for practical drone path planning.

In summary, while BFS provides an invaluable theoretical baseline as the simplest complete and optimal uninformed search algorithm, its practical applicability to real-world autonomous drone racing is severely limited by its $O(|V|)$ space complexity. The analysis presented in this paper quantitatively demonstrates that the cost of BFS's unconditional optimality guarantee—the wholesale exploration of the search space—is too high a price for systems where memory, processing time, and deterministic behavior are scarce and safety-critical resources.

8 Acknowledgment

I would like to express my deepest gratitude to Allah SWT for grace and guidance, which enabled the completion of this paper. I also extend my sincere appreciation to the lecturers of the IF2211 Algorithm Strategies course for their valuable knowledge and guidance.

I also thank my colleagues for their support, insightful discussions, and collaboration throughout the research and simulation process.

References

[1] S. Li, M. M. O. I. Ozo, C. De Wagter, and G. C. H. E. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *Robotics and Autonomous Systems*, vol. 133, p. 103621, 2020.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA: MIT Press, 2022.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ: Pearson, 2021.

[4] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[5] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] STMicroelectronics, "STM32F405/415, STM32F407/417 reference manual," RM0090, Rev. 19, 2021.

[8] R. Munir, "Algoritma BFS dan DFS," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB, 2026.

[9] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.

[10] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2026

Reysha Syafitri MR (NIM 13524137)