RAPID: A Path Planning Framework for Autonomous Aircraft in Dynamic Risk Environments

Julian Gutierrez, Esther Conrad, Lauren M. White, John Siratt, and César A. Muñoz
Safety-Critical Avionics Systems Branch
NASA Langley Research Center
Hampton, VA
{julian.gutierrez, esther.d.conrad, lauren.m.white, john.siratt, cesar.a.munoz}@nasa.gov

***Abstract*—Risk-Aware Planner with In-flight Dynamics (RAPID) is a path planning framework designed for autonomous aircraft operating in dynamic risk environments. It addresses the challenge of generating flyable trajectories that balance speed and risk mitigation in real-time applications. The framework integrates Dubins paths to model kinematically feasible path segments that respect aircraft turning constraints, while temporal risk grids capture time-varying environmental hazards and obstacles. An adapted path planning algorithm is used to compute paths to multiple objectives through a risk-weighted spatial-temporal graph. The combination of Dubins paths, temporal risk assessment, and graph-based optimization creates a robust framework for effective autonomous flight planning in complex, evolving environments where traditional static planning approaches are inadequate. Serving as a proof of concept for risk-aware path planning, this system establishes a foundation that could eventually lead to increased safety metrics through operational hazard mitigation.**

I. Introduction

Path planning is a fundamental element of autonomous Unmanned Aerial Vehicles (UAV) and Urban Air Mobility (UAM) operations. In contingency management scenarios, it is essential to rapidly generate reliable alternative flight paths to maintain operational safety when unexpected conditions are encountered. Traditional path planning approaches often fail to adequately address the dual requirements of safety assurance and flyability constraints, particularly in time-critical situations where computational efficiency is paramount.

In UAV operations, paths must avoid both static obstacles (buildings, terrain, no-fly zones) and dynamic hazards (other aircraft, weather systems, GNSS availability, temporary restrictions). Additionally, flyable paths must respect aircraft kinematic constraints such as minimum turn radius. The challenge intensifies in UAM environments where dense airspace utilization and proximity to populated areas demand complex risk assessment and real-time path adaptation capabilities. Contingency management in emergency situations require real-time or near real-time path replanning that takes new risk factors into account. RAPID addresses the limitation of current approaches by integrating risk assessment, flyability constraints, and temporal risk evolution within a unified graph-based search designed for rapid computation.

II. Background

A. The Challenge of Dynamic Path Planning

Traditional autonomous systems often rely on static path planning algorithms, such as Dijkstra's [1] and A* [2], which assume constant traversal costs. While effective for unchanging topologies, this assumption breaks down in modern operations like Urban Air Mobility (UAM), where UAVs must navigate complex, time-varying landscapes [3], [4]. Today's operational environments introduce multidimensional, dynamic challenges: continuously evolving weather patterns [5], fluctuating population densities [6], navigation uncertainties like GPS denial or multipath interference [7], [8], and unpredictable airspace traffic [9]. Consequently, traditional static cost models cannot adequately represent systems where risk is a fundamental function of both space and time.

Since risk profiles shift continuously, planning algorithms must compute costs dynamically using both spatial and temporal contexts. For example, the same path may face high population exposure during certain times of the day while being safer during the night [6]. While prior works have explored dynamic and temporal planning through real-time path recomputation algorithms [10], adaptable roadmaps [11], and time-varying cost models [12], [13], significant gaps still remain. Existing literature has yet to adequately address the continuous integration of multi-dimensional, time-varying risks while strictly enforcing hard safety constraints during mission execution.

B. Kinematic Constraints and Feasible Trajectory Design

Beyond environmental dynamics, the physical properties of aircraft themselves impose fundamental constraints on feasible paths, e.g., the dynamics of fixed-wing aircraft prohibits instantaneous changes in direction. Deterministic search in a discretized state space using motion primitives that respect vehicle dynamics and continuity constraints is suggested in [14]. For example, the turn radius of any aircraft is governed by the relationship between velocity, maximum roll angle, and gravitational acceleration.These kinematic constraints create several considerations for path planning. For example, optimal

solutions in Euclidean space calculated by static algorithms without accounting for turning constraints may violate aircraft physical capabilities. Furthermore, a path that minimizes distance or Euclidean risk might require a turn sharper than the aircraft can execute, rendering it infeasible despite its theoretical optimality.

Graph-based planning algorithms must therefore explicitly incorporate turn radius constraints to generate feasible arc paths rather than only straight-line approximations [15]. This requirement introduces computational complexity: checking feasibility requires not just topological reachability but geometric validation against kinematic bounds. A Dubins path representation, which provides circle-based motion primitives and curve-constrained path planning, is a standard approach to address these constraints, particularly for aerial vehicles with bounded turning dynamics [16].

C. Multi-Dimensional Risk Integration

Safe autonomous operations require integrating multiple independent risk dimensions, including environmental hazards, temporal patterns, and operational constraints [17]. These multi-dimensional risks act as spatial proxies for physical hazards. For example, navigation risk maps help minimize exposure to areas with limited GNSS availability or high multipath likelihood, which can severely degrade a vehicle's state estimation. Weather grids might represent areas of high winds, low visibility, or precipitation with high risk; while population density maps can indicate elevated ground risk in the event of a vehicle failure. In all cases, higher assigned risk values correlate with areas the vehicle should actively avoid.

Within this context, we consider that a trajectory generated by a path planning algorithm is *safe* if it strictly satisfies operator-defined upper-bound risk thresholds. Importantly, our proposed framework is entirely agnostic to the underlying risk types. While an Urban Air Mobility (UAM) mission might prioritize population and GNSS risks, the architecture seamlessly adapts to different operational profiles or entirely different domains, such as submarine navigation. The methodology assumes appropriate risk grids are provided; comprehensively defining all real-world operational hazards remains outside the scope of this paper.

To achieve this safety standard, the planner must maintain high-fidelity individual risk channels rather than collapsing them into a single scalar value, which loses critical distinctions [6]. It must support mission-specific risk prioritization, enforce both hard thresholds (e.g., no-fly zones) and soft constraints [18], and allow for rapid weight adjustments. Traditional graph representations that use a single scalar cost fail these requirements, as they force predetermined weightings and obscure underlying risk composition [19]–[21]. Therefore, a multi-dimensional risk model is essential [22]. By maintaining separate risk channels throughout the search process, the proposed framework allows dynamic constraint enforcement and on-demand tactical weight adjustments without requiring computationally prohibitive algorithmic restarts.

D. Temporal Risk in Path Planning

Some approaches have addressed temporal dynamics in planning, including Safe Interval Path Planning (SIPP) [23] for time-dependent environments, or finding efficiencies in energy consumption for vehicles that depend on time-varying flows [24]. However, most existing approaches focus on dynamic obstacles or time-windows rather than risk-based temporal evolution. RAPID integrates a path planning algorithm that has been modified to ensure temporal factors are considered in the solution.

E. Motivation for the Proposed Approach

The convergence of dynamic temporal risks, kinematic constraints, and multi-dimensional risk integration creates a planning problem that exceeds the capabilities of traditional static algorithms like Bellman-Ford, Dijkstra or A*. Standard graph representations are not designed to enforce kinematic constraints (such as Dubins path geometry). Furthermore, highly volatile factors like wind or collision risk can quickly invalidate initial prediction assumptions mid-execution, so a modern planner must execute fast enough to enable in-flight replanning.

To address this, the proposed RAPID framework utilizes a static topological structure that persists across replanning events; when environmental conditions change, only edge costs require re-evaluation. This architectural design inherently supports rapid contingency strategies such as full replanning, repair-based adjustments, or warm-start initialization. The methodology section that follows details how RAPID combines explicit kinematic modeling with this flexible risk representation to achieve dynamic, multi-constrained path planning, establishing a foundation for ongoing work in real-time predictive replanning for autonomous operations.

III. Methodology

RAPID is a five-stage path planning framework that transforms raw environmental data into risk-optimized, kinematically feasible trajectories. While the underlying methodology can be extended to three-dimensional environments, the version evaluated in this work operates within a 2D spatial area over time. The framework requires three primary inputs: a polygonal representation of the building landscape, a set of predictive temporal risk grids, and user-defined configuration parameters. The temporal risk grids represent various spatial hazards normalized to a $[0, 1]$ scale and feature independent update rates to accommodate both static constraints, such as no-go zones, and highly dynamic, fast-changing risks. Users can define thresholds to dictate the risk levels at which the algorithm must strictly avoid. Additionally, configuration parameters govern the framework's behavior by establishing structural and kinematic constraints, including grid resolution, maximum edge length, minimum turn radius, speed, battery capacity, and risk combination methods.

RAPID processes these inputs through five sequential stages: (0) preprocessing, (1) vertex generation, (2) flyable graph construction, (3) heading constraint enforcement, and

(4) path construction. The final output is a structured flyable path composed of nodes and edges, along with detailed metadata for each edge regarding the expected traversal time, battery consumption, and associated risk. This graph-based trajectory can then be directly converted into standard waypoints for seamless integration with downstream systems, such as vehicle autopilots.

A. Stage 0: Preprocessing Stage

Building polygon preprocessing is done in three steps to reduce complexity while preserving safety margins. These polygon simplification steps are part of the optimization strategy discussed in Section III-G.

The first step in the preprocessing stage is a convex hull simplification of obstacle polygons with buffering. This replaces complex polygons that have concave features by computing their convex hulls while simultaneously adding a buffer, increasing the size of the polygon by the r_{turn} (where r_{turn} is the minimum turn radius of the vehicle). The second step, polygon merging, combines adjacent or overlapping polygons using union operations. This simplification step creates simpler obstacle regions that are more efficient to process. The third step applies the Douglas-Peucker simplification algorithm [25] with a 1.0-meter tolerance, further reducing detail while preserving essential geometry features. Combined, these three steps achieve vertex and polygon count reduction, reducing graph construction time with negligible impact on path quality. Figure 1 shows an example of the original polygons obtained from building models and the resulting simplified polygon.

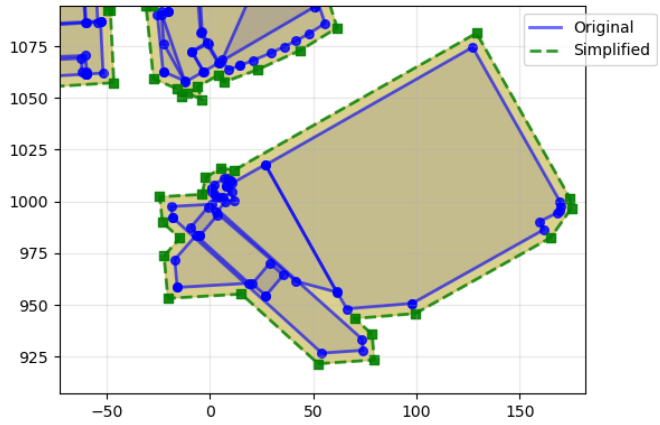

Fig. 1: Polygon simplification showing the difference between the original polygons (over 15 in view for this area) and the new simplified polygons (reduced to 3 in view in this area).

B. Stage 1: Vertex Generation

The environment is discretized into a uniform grid with resolution δ_{grid} (this value is a configurable parameter, which we set by default to 5 meters). Grid bounds are determined from the bounding box of all obstacles plus a configurable buffer margin. Vertices are generated from two principal sources: polygon vertices extracted from preprocessed building boundaries and grid vertices from regularly spaced points in free space. Figure 2 shows the vertices generated for the selected area of analysis (Columbus, Ohio).

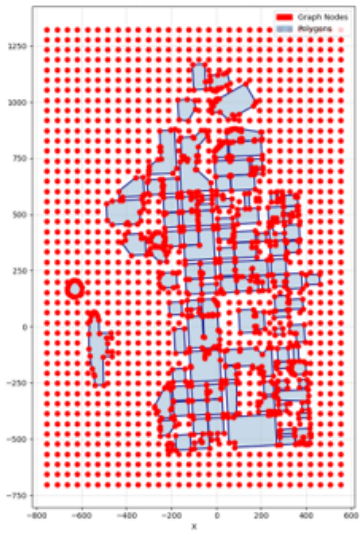

Fig. 2: Map of all the vertices added, including polygon vertices and evenly-spaced empty vertices.

C. Stage 2: Flyable Graph Construction

The graph foundation represents vehicle maneuvering capabilities through circles positioned at the vertex locations identified in Stage 1. Each circle $C = (c, r_{\text{turn}})$ represents a feasible turning location for circular arcs with specified radius. The strategic vertex placement at both obstacle boundaries and regular grid intervals ensures the graph captures navigation options around obstacles and through open areas.

An undirected graph is constructed from nodes along circle circumferences. The graph connectivity is established through tangent edges connecting circle pairs. For circles C_i and C_j with equal radii r_{turn}, tangent edges are computed through a sequence of operations: distance calculation between circle centers, angle computation via $\alpha = \arccos\left(\frac{2r_{\text{turn}}}{d}\right)$, tangent point pair generation, polygon intersection validation against buffered obstacle zones, and edge length checking.

Between consecutive tangent points (nodes in the undirected graph) on the same circle circumference, arc edges represent curved flight paths. Arc length is computed as $s = r_{\text{turn}} \times \theta$, where θ is the subtended angle. Arcs are retained only if entirely outside buffered polygon zones, ensuring kinematic feasibility. This arc-based representation allows RAPID to naturally enforce turning radius constraints without explicit post-processing.

To evaluate risk along edges in the graph, we must determine which grid cells each edge passes through and quantify the distance traversed in each cell. This computation is critical for risk aggregation and temporal constraint checking. For each edge connecting two nodes in the spatio-temporal graph, we first identify all grid cells that the edge's geometric projection (typically a line segment or circular arc in the

spatial plane) intersects. We employ a two-stage approach to accomplish this efficiently: (1) coarse identification of potentially intersected cells using a Digital Differential Analyzer (DDA) line-rasterization algorithm, and (2) precise geometric computation of intersection length for each candidate cell using the Cohen-Sutherland line-clipping algorithm [26].

The undirected graph is converted to a directed graph by duplicating each node into two directional variants representing clockwise and counterclockwise turns, doubling node count ($|V| \to 2|V|$) while enabling proper representation of turning constraints. Each edge attached to each node is evaluated to ensure proper connection through the node by comparing vector alignment of the edges.

A key advantage of the Stage 2 output is that the complete flyable directed graph can be precomputed and saved as a reusable asset for a given geographic area. Once constructed, this graph remains valid indefinitely unless fundamental parameters change, e.g., building, grid resolution or turn radius constraints. This precomputation eliminates the need to reconstruct the graph for each planning query within the same area, enabling orders-of-magnitude faster planning for repeated missions in the same environment.

D. Stage 3: Start and End Point Integration

RAPID was designed to integrate the departure heading through a dual tangential circle configuration. Given a start position $\mathbf{s}$ and desired heading θ, the algorithm creates two circles with radius r_{turn} positioned symmetrically perpendicular to the desired heading. Tangent lines are computed from each circle to nearby circle vertices, and only tangent edges whose bearing aligns with θ are retained. The temporary start node is added with these filtered edges, enforcing departure direction constraints.

End points connect to the graph via standard tangent lines from goal positions to nearby circles, supporting multiple simultaneous destinations for multi-goal planning. This unified integration approach for both start and end constraints provides a general mechanism for both trajectory boundary conditions.

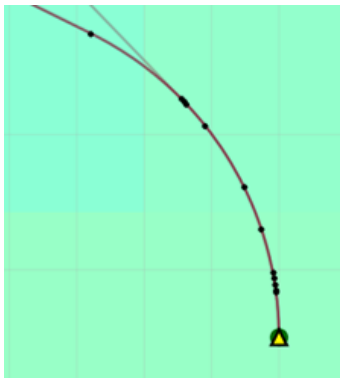

Fig. 3: Example demonstration of a turn a Dubins vehicle could take and the nodes along the arc.

E. Stage 4: Path Planning with Temporal Risk Constraints

The path planning objective is to find a path $P = [v_0, v_1, \ldots, v_n]$ minimizing cost $C(P) = \sum_{i=0}^{n-1} \text{cost}(e_i, t_i)$ subject to risk constraints $\text{risk}^{(k)}(e_i, t_i) \leq \tau^{(k)}$ for all risk types k, battery constraint $E_{\text{total}}(P) \leq B_{\text{capacity}}$, collision avoidance, and kinematic feasibility. Here, $\text{risk}^{(k)}$ denotes normalized risk in the range [0,1], $\tau^{(k)}$ is the operator-specified threshold for risk type k, and t_i is the estimated arrival time at vertex v_i.

RAPID supports generic risk dimensions, each with different temporal characteristics and update rates. The only assumption required is that each risk is normalized. To support continuous-time path planning while maintaining computational efficiency, the temporal handling processes risk data through discrete time intervals with queue-based relaxation. When a path traverses from one temporal interval to the next, edge evaluation uses the risk grid active during traversal, ensuring temporal fidelity without requiring full continuous interpolation. The system maintains a queue of nodes to process for each time interval:

- `time_interval_queues (TIQ)`: Map from update time to deque of nodes.
- `current_temporal_risk_grids (CTRG)`: Risk grids active in current time interval.

When edge costs change due to risk grid updates, the affected nodes are reinserted into future time queues for processing, ensuring convergence under time-varying cost functions.

1) Edge Risk Evaluation: For each edge, risk is evaluated through a systematic four-step process. First, traversal time is computed from the edge's distance and nominal speed. Second, the algorithm uses the grid cell information computed in Stage 2 to evaluate the individual risk at each grid cell using the current temporal risk grid. These values are compared against thresholds defined by the mission planner to ensure no specific risk exceeds acceptable limits at any point along the edge. Third, risk values are combined across cells using a weighted aggregation based on the distance traveled within each cell, giving higher weight to longer traversals. Fourth, the different risk types are combined using a selected method to produce a final edge risk metric. This overarching metric can then be used in conjunction with the individual risk factors to enable more informed decision-making.

Risk combination methods provide different operational semantics. The maximum method computes $R_{\text{combined}} = \max_k(R^{(k)})$, implementing a conservative approach where any high risk across dimensions is problematic. The weighted sum method computes $R_{\text{combined}} = \sum_k w^{(k)} R^{(k)}$ where $\sum_k w^{(k)} = 1$, providing balanced multi-dimensional risk aggregation. The average method is the same as the weighted sum method where all the weights are equal. These methods can be selected based on mission priorities and operational doctrine.

2) Cost Functions: RAPID supports multiple cost functions that allow for different mission priorities and operational constraints. These functions range from simple baseline ap-

proaches to sophisticated risk-aware variants with threshold enforcement. For all cost functions, if any risk type violates its configured threshold, the edge cost is set to infinity, marking it as infeasible. Otherwise, cost is computed according to the selected function:

Distance only serves as a baseline, ignoring risk entirely: $\text{cost}(e_i) = \text{distance}(e_i)$. This represents unconstrained shortest path planning, providing a reference point for comparison.

Distance with thresholds enforces hard constraints: $\text{cost}(e_i) = \text{distance}(e_i) \wedge \forall_j \text{risk}(e_j) < R_{max}$. This hard-constraint mode marks any edge violating risk thresholds as infeasible, preventing paths through constrained regions entirely, and provides binary feasibility decisions useful when strict compliance is mandated.

Max risk with thresholds is the primary risk-aware variant: $\text{cost}(e_i) = \text{distance}(e_i) \times \max_k(\text{risk}^{(k)}) \wedge \forall_j \text{risk}(e_j) < R_{max}$. This minimizes peak risk within constraint bounds while preferring shorter paths.

3) Modified Bellman-Ford Algorithm: While RAPID is agnostic to the path solver algorithm, this paper demonstrates the framework using a time aware variant of the Bellman-Ford algorithm, building upon the approach used by Gutierrez et al. [12].

The Bellman-Ford algorithm was selected as the default solver for several strategic reasons. First, the Bellman-Ford algorithm provides formally proven mathematical guarantees, and the theoretical limitations of its temporal variants are well understood, as demonstrated by Conrad et al. [27]. Second, its structure is highly parallelizable, which is a critical requirement for future extensions of RAPID into three dimensional and temporal environments where GPU acceleration will be necessary. Third, it naturally supports multiple goal planning, which is highly valuable in contingency management scenarios where the system is searching for any potential vertiport to direct the vehicle toward in the event of an emergency. Unlike the standard algorithm which iterates over vertex relaxation passes, this variant processes nodes in temporal order, enabling time aware edge evaluation and efficient handling of time varying costs. This is a variation of the algorithm described by Gutierrez et al. [12].

The path search begins by initializing $\text{dist}[\text{start}] = 0$ and all other distances to infinity. It maintains parent pointers for path reconstruction and builds a time-indexed queue structure partitioning nodes by time intervals: $\text{TIQ} = \{\text{update\_time}_i : \text{deque}_i\}$. The start node is placed in the queue corresponding to departure time.

During the main relaxation phase, the algorithm iterates over each time interval in chronological order. For each time interval, it updates the active temporal risk grids and processes all queued nodes for that interval. For each queued node u, the algorithm examines all outgoing edges (u, v). For each edge, it performs the following operations: evaluates edge risk using the current risk grid, checks whether risk violates thresholds (skipping infeasible edges), computes edge cost via the selected cost function, and if $\text{dist}[u] + \text{cost} < \text{dist}[v]$, updates $\text{dist}[v]$ and adds v to the queue corresponding to the estimated arrival time at v. This queue-based approach ensures nodes are processed with current risk data, naturally handling time-varying edge weights.

The algorithm supports several early termination strategies to reduce computational cost. It exits immediately when an iteration produces no distance updates, indicating convergence. It exits when all goal nodes have been reached with finite distance, useful for multi-goal queries. It may exit on finding the first path to any goal or waiting until all goals are found, depending on mission requirements.

The time-aware queue processing mechanism handles temporal cost variations efficiently. When risk grids change at update times, edges crossing temporal boundaries are evaluated with appropriate grid snapshots. When reverse dependencies are tracked (noting which nodes depend on edge evaluations), updating reverse edges upon grid changes keeps the algorithm forward-looking. This approach naturally accommodates time-varying obstacles, risk fields, and cost functions without explicit temporal interpolation or multiple passes.

A critical challenge in solving the three-dimensional path planning problem (2D space + time) using discrete time intervals is maintaining path viability when risk values improve after an edge has been evaluated. The algorithm addresses this through two complementary mechanisms. First, it invalidates nodes whose previously-optimal paths become unflyable; these nodes are requeued for reprocessing with corrected risk evaluations. Second, it stores complete paths to goal nodes at each risk grid update time, ensuring that viable paths are preserved even if subsequently discovered paths become infeasible. This dual strategy guarantees that correct solutions are not lost when temporal risk variations cause previously good routes to violate constraints.

4) Battery Constraint Handling: Energy consumption along an edge is proportional to the distance of the edge and inversely proportional to the speed. During Bellman-Ford relaxation, before updating a distance via an edge, the algorithm checks battery feasibility using a straight-line heuristic: if the straight-line distance from the candidate destination to any goal exceeds the remaining battery capacity, the edge is marked infeasible. This pruning heuristic is efficient, avoiding deep exploration of paths that cannot reach any goal. Additionally, a cumulative battery consumption is computed as edges are being explored; paths violating the battery constraint $\sum E_i \leq B_{\text{capacity}}$ are rejected, and the algorithm backtracks to explore alternative routes.

F. Pseudocode

Algorithm 1 presents the pseudocode for the modified Bellman-Ford path search algorithm with time-aware queue processing. The algorithm processes nodes organized by time intervals, updating risk grids as time advances. Early termination triggers when no relaxations occur during an iteration or when all goal nodes are reached.

G. Performance Optimizations

Key optimizations were designed and implemented to reduce computational and memory burden while maintaining so-

Algorithm 1 Time-Aware Bellman-Ford with Battery and Risk Constraints

Initialize dist[$\cdot$] $\leftarrow \infty$, dist[start] $\leftarrow 0$
TIQ $\leftarrow$ {update_time$_i$: deque() for each time interval}
TIQ[start_time].append(start)
goals_found $\leftarrow$ {}
for each time_interval in chronological order **do**
    CTRG $\leftarrow$ risk grids active at time_interval
    queue $\leftarrow$ TIQ[time_interval]
    updated $\leftarrow$ False
    while queue is not empty **do**
        $u \leftarrow$ queue.popleft()
        for each edge (u, v) from u **do**
            edge_risk $\leftarrow$ EVALUATEEDGERISK(edge, CTRG)
            if edge_risk violates thresholds **then**
                continue
            end if
            edge_cost $\leftarrow$ COMPUTECOST(edge, edge_risk)
            if CHECKBATTERYFEASIBILITY(v, dist[u] + edge_cost) **then**
                if dist[u] + edge_cost $<$ dist[v] **then**
                    dist[v] $\leftarrow$ dist[u] + edge_cost
                    parent[v] $\leftarrow u$
                    arrival_time $\leftarrow$ estimated time of arrival at v
                    TIQ[arrival_time].append(v)
                    updated $\leftarrow$ True
                    if v is a goal node **then**
                        goals_found.add(v)
                    end if
                end if
            end if
        end for
    end while
    if not updated **or** all_goals_found **then**
        break
    end if
end for
return RECONSTRUCTPATHS(parent, goals_found)

lution quality and algorithmic guarantees. These optimizations deliver substantial performance improvements in both graph construction and pathfinding stages.

1) Polygon Simplification: Building polygon preprocessing employs three complementary techniques to reduce geometric complexity. Convex hull simplification replaces complex polygons by computing their convex envelopes, reducing vertex counts by 60–70% while preserving safety margins through outward buffering by r_{turn}. Polygon merging combines adjacent or overlapping polygons via union operations, reducing polygon count by approximately 74%. Douglas-Peucker simplification with 1.0-meter tolerance further refines detail while preserving essential features. Combined, these techniques achieve 69% vertex count reduction and 74% polygon count reduction in testing, yielding 5–10× speedup in graph construction with negligible impact on path quality.

2) Graph Sparsification: Redundant edges that provide equivalent routing options without improving connectivity are removed from the graph structure. This is done via collinearity checks, baked into Stage 2. Graph sparsification contributes 5-10% additional performance improvement by reducing edge count without sacrificing solution quality. It additionally increases reliability of edge traversal by reducing longer edges.

3) Edge Caching: A memory-efficient lookup table stores precomputed edge evaluation results (cost, risk values, validity status) across pathfinding queries. When an edge is evaluated, cache hits reuse previous computations without redundant risk grid lookups. The cache stores pre-calculated edge risk evaluations and achieves 60-70% reduction in edge evaluation cost. The cache is strategically cleared when risk grids update at temporal boundaries, ensuring temporal correctness and consistency with dynamic risk evolution.

4) Path Invalidation and Storage Strategy: A critical challenge in discrete-time approximations of continuous path planning is maintaining viable solutions when risk improves temporally. When a risk grid update reveals that a previously-optimal path now violates constraints later down the path, the algorithm invalidates affected downstream nodes and requeues them for reprocessing with current risk grids. Conversely, whenever the algorithm reaches a goal node, the complete path is stored immediately, ensuring viable alternatives are preserved. This dual mechanism prevents loss of feasible solutions when temporal risk variations cause previously good routes to become infeasible after intermediate risk grid updates.

5) Early Termination: The Bellman-Ford algorithm terminates in multiple scenarios rather than iterating until theoretical completion. Early termination when an iteration produces no distance updates indicates convergence with optimality guarantees. Termination when all goal nodes have been reached eliminates unnecessary relaxation passes in multi-goal scenarios. Optional termination on finding the first path to any goal supports applications prioritizing speed over completeness. These strategies reduce actual iterations in the order of 80-90% from the theoretical worst case of $|V| - 1$.

IV. Experimental Setup and Problem Configuration

The RAPID algorithm was evaluated on a 2D representation of a simulated environment of Columbus, Ohio spanning approximately 2.8 km² and containing approximately 100 buildings. This dataset served as a representative test case for validating the algorithm's performance in complex urban geometry with realistic risk constraints and battery limitations. The city-scale problem presents substantial practical challenges for pathfinding, including dense obstacle geometry and high-dimensional risk data. Figure 4 shows the modeled space at the final snapshot of the combined risk, when all computed paths to each end point have been reached.

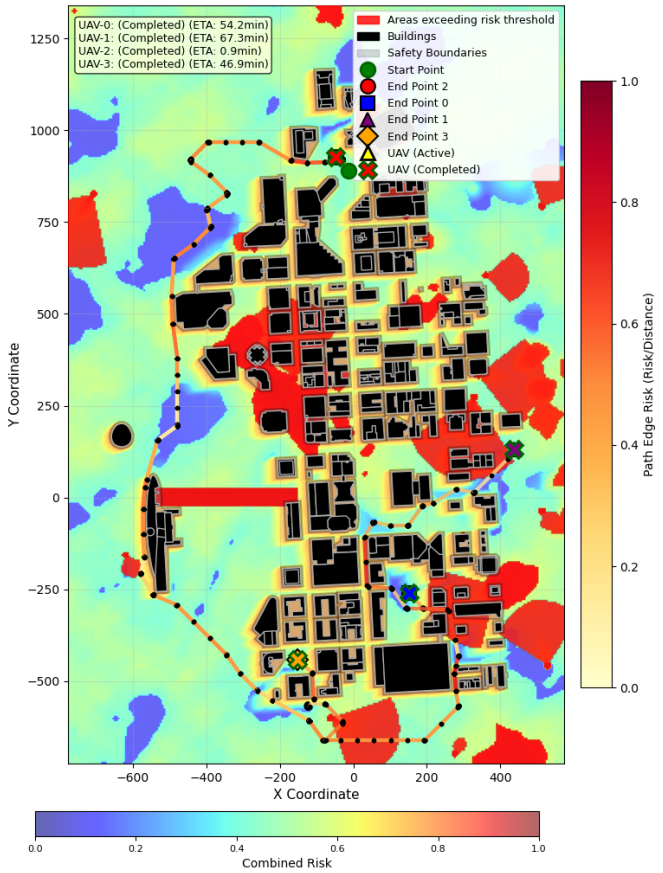

Fig. 4: Final path results for the modeled space showing the combined risk in the background, and the average cost of each edge along all the paths to the different endpoints. Red grid cells show cells which violate any given risk threshold.

The temporal risk prediction environment was modeled as a two-dimensional grid structure that changed dynamically across a 90-minute planning horizon. While the RAPID framework is fundamentally agnostic to the specific types of risk provided, the grids used in this evaluation encode six distinct hazards with distinct update rates, representative of a potential UAM operation:

1) **Ground Casualty Risk**: Estimated exposure risk based on population distributions and activity patterns within the urban area. Higher values indicate elevated population density where a catastrophic vehicle failure would pose a greater threat to people on the ground. Updated every 30 minutes.
2) **Weather/Wind Risk**: Temporal variations in weather conditions and wind patterns affecting flight safety and vehicle control throughout the planning window. Updated every 10 minutes.
3) **Terrain Collision Risk**: Terrain characteristics influencing navigation safety. Higher values denote peaks or topographical formations that present a physical collision hazard. Static risk.
4) **Navigation Systems Risk**: GPS accuracy degradation and navigation uncertainty, particularly in urban canyons. Higher values represent areas where GNSS line-of-sight is reduced, potentially producing erroneous position solutions [7]. Updated every 2 minutes.
5) **Vehicle Collision Risk**: Dynamic risk from other manned and unmanned aircraft traffic. Higher values represent areas projected along the directional movement vectors of other vehicles operating in the airspace. Updated every 2 minutes.
6) **No-Go Zones**: Restricted airspace, protected areas, and operational exclusion zones representing hard navigation constraints. Static risk.

Each risk dimension is normalized to a $[0, 1]$ scale with configurable thresholds. The algorithm was executed under different parameter configurations to test the performance aspects. The default risk environment was configured with 5m × 5m grid cells and a speed of 1 m/s and a minimum turn radius of 5 m was defined for the vehicle.

The algorithm was implemented in Python 3.10 using the following libraries: NumPy for numerical operations, Shapely for computational geometry, and pandas for result analysis. The results described in section V were collected under different parameter combinations: grid resolution, maximum edge length, speed of vehicle, vehicle minimum turn radius and battery capacity. Each combination was tested 5 times. Experiments were conducted on a high-performance compute server with the following specifications:

- **CPU**: Dual Intel Xeon Silver 4216 processors @ 2.10 GHz (64 cores total, 128 threads with hyperthreading)
- **Memory**: 125 GB RAM (with 2 NUMA nodes)
- **Operating System**: Linux (Red Hat Enterprise Linux 8.10, kernel 4.18.0)

V. Results

A. Computational Bottleneck Analysis

Figure 5 decomposes the execution time across the four algorithmic stages under baseline configuration (5 m grid resolution, 100 m maximum edge length, 5 m turn radius, 1 m/s speed, 150 Wh battery capacity). The breakdown reveals a striking computational bottleneck: Stage 2 (temporal risk grid generation and aggregation) accounts for 62.8% of total execution time, consuming approximately 11.2 seconds of the runtime (17.8 s).

Stages 1 and 3 contribute minimally (1.9% each), indicating that vertex construction and start and end point incorporation are negligible. Stage 4 (planning via Bellman-Ford) comprises 33.2% of execution time, representing the core algorithmic cost. This distribution suggests that optimization efforts should prioritize Stage 2 processing, such as GPU-accelerated graph generation. The relatively small Stage 4 overhead demonstrates that Bellman-Ford's worst-case O(n^3) complexity is manageable for the discretized graphs employed in this application. Additionally, the edge weight caching optimization can be used to further decrease the execution time of the path planning algorithm. If possible, a function to

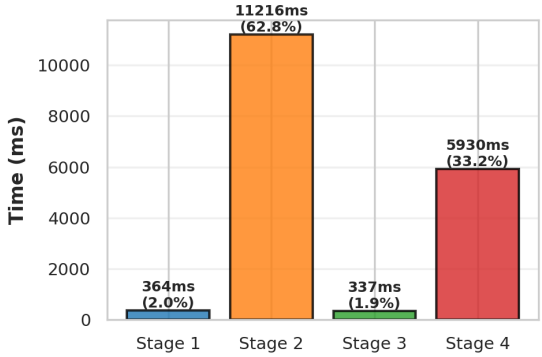

Fig. 5: Average proportional contribution of each algorithmic stage to total execution time under baseline configuration (grid resolution 5 m, turn radius 5 m, speed 1 m/s, battery 150 Wh).

pre-compute all edge weights across all time intervals can be executed prior to the path planning algorithm, which reduces the Stage 4 from 5.9 s to 1.8 s.

B. Risk Integration Overhead

Figure 6 quantifies the computational overhead introduced by risk consideration across three cost function variants: distance-only (baseline with no risk integration), distance-with-thresholds (risk used only for hard constraint violation detection), and max-risk-with-thresholds (risk integrated into edge weights for optimization). The distance-only approach achieves the fastest total execution time (15.2 seconds average), establishing a lower bound for planning performance.

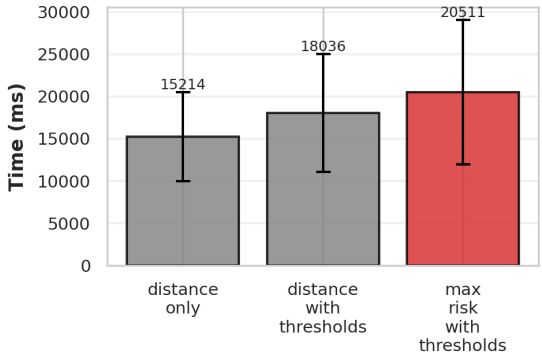

Fig. 6: Comparison of average execution time across three cost functions for RAPID under different parameter configurations.

The distance-with-thresholds variant increases execution time to 18.0 seconds (+18.6% overhead), reflecting the cost of evaluating threshold constraints at each candidate edge. The max-risk-with-thresholds approach, which integrates risk into the optimization objective, reaches 20.5 seconds (+34.9% overhead compared to distance-only). This progression demonstrates that while risk integration significantly increases computational cost, the overhead remains acceptable for operational planning horizons. The variance in execution times (reflected by error bars) increases with risk complexity, indicating that risk-weighted planning exhibits higher sensitivity to problem-specific configurations.

C. Discretization Parameter Effects

Figure 7 examines how two critical discretization parameters affect Stage 4 execution time. The top panel shows the impact of grid resolution, varying from 1 m to 50 m. Finer grid resolution (1 m) produces significantly larger risk grids, and consequently a larger number of grid cells to store and traverse through for each edge, and thus, longer planning times (12 s). As grid resolution coarsens, execution time decreases monotonically, reaching 5 s at 50 m resolution. This inverse relationship reflects the quadratic growth in the number of grid cells as resolution decreases.

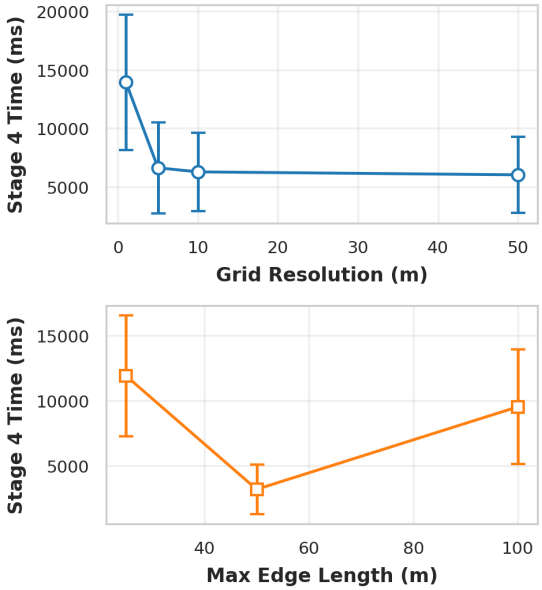

Fig. 7: Parameter Sensitivity Analysis. **(Top)** Impact of grid resolution on Stage 4 execution time. **(Bottom)** Impact of maximum edge length on Stage 4 execution time.

The bottom panel depicts the effect of maximum edge length, which exhibits a complex, U-shaped relationship where execution time reaches a minimum of 4 s at 50 m. This behavior stems from a natural trade-off between empty-space vertex density and the reach of connections to polygon vertices. When the maximum edge length is short (e.g., 25 m), the algorithm must generate a high density of vertices in empty space. Because there are so many vertices overall, a large number of them fall close to polygon boundaries, allowing for a massive amount of localized connections and resulting in a highly dense graph, $G(N = 92k, E = 150k)$. Conversely, when the edge length is long (e.g., 100 m), fewer vertices are placed in empty space, but their extended reach allows each vertex to successfully connect with a much larger number of distant polygon vertices. This increase in long-range connections drives the graph size back up to $G(N = 60k, E = 89k)$. The 50 m threshold represents a balance, minimizing both unnecessary empty-space nodes

and excessive long-range polygon connections, yielding the most compact graph, $G(N = 26k, E = 40k)$. Ultimately, this behavior highlights the importance of tuning edge length to the specific problem domain to balance adequate graph connectivity with computational tractability.

Stage 2 execution time varied directly with graph size as well. Grid resolution, on the other hand, had no measurable impact in the performance of Stage 2. This is due to the fact that even though more intersections happen with finer grid resolutions, the grid cell computations from Stage 2 are highly parallelized so the impact in performance is negligible.

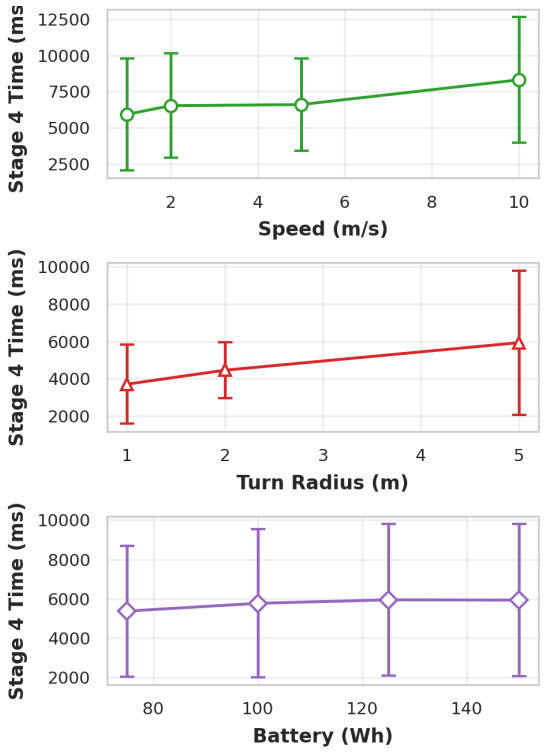

Fig. 8: Vehicle Parameter Effects on Execution Time. Stage 4 (planning) execution time sensitivity to vehicle-specific parameters while maintaining constant discretization. **(Top)** Speed effect (1–100 m/s), **(Middle)** Turn radius effect (1–5 m), **(Bottom)** Battery capacity effect (75–150 Wh).

Figure 8 shows how vehicle-specific parameters influence planning efficiency while maintaining constant discretization (5 m grid, 5 m turn radius baseline). The top panel shows the effect of vehicle speed (1-10 m/s) on execution time. Faster speeds increase execution time from 6 s to 8 s on average, because higher velocities compress the temporal planning horizon, requiring the path planning algorithm to expand more candidate nodes within tighter time windows. The speed increase produces proportional time increases due to the scaling of temporal discretization.

The middle panel demonstrates the impact of turn radius (1-5 m). Tighter turn radius constraints modestly increase execution time from 4 s to 6 s. Smaller turn radii provides paths that would otherwise be infeasible with larger Dubins-curve geometry, producing alternative paths that the algorithm converges faster. The battery capacity analysis (bottom panel) reveals minimal sensitivity, with execution time remaining approximately constant (9 - 10 s) across the range 75-150 Wh. This insensitivity occurs because battery constraints primarily affect path feasibility. The main effect battery capacity has in algorithm performance is if battery is depleted swiftly, resulting in an early termination of the algorithm, which only occurs if battery capacity was under 40 Wh in this scenario.

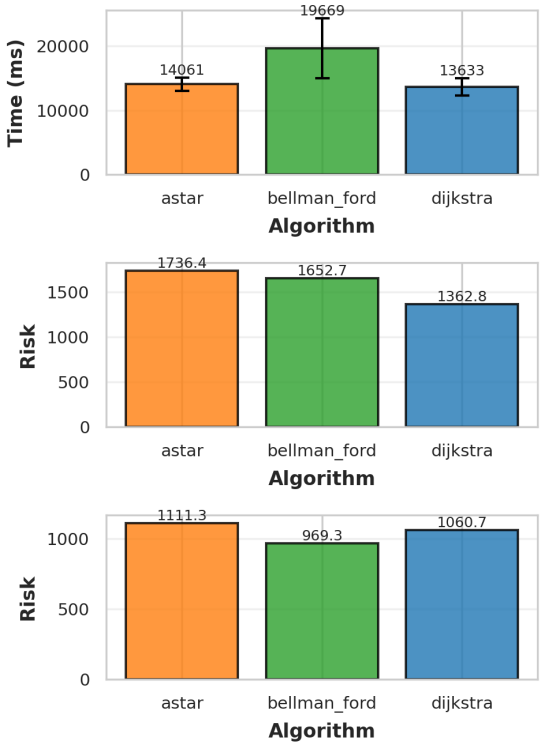

Fig. 9: Algorithm Comparison. **(Top)** Total execution time for three path planning algorithms (A*, Bellman-Ford and Dijkstra). **(Middle)** Path quality (accumulated risk) at endpoint 0 for vehicle speed 1 m/s using max-risk-with-thresholds cost function. **(Bottom)** Path quality at endpoint 0 for vehicle speed 100 m/s.

D. Algorithm Performance Comparison

The A* and Dijkstra algorithms were also extended to support time-dependent inputs, permitting multiple visits to the same edge across different time intervals to discover risk-reduced paths. Figure 9 presents a comprehensive comparison of the three graph algorithms evaluated on the temporal risk-aware path planning problem. The top panel shows total execution time across all stages. Dijkstra and A* achieve similar performance (13–14 seconds), while Bellman-Ford requires approximately 20 seconds. The increased computational cost of Bellman-Ford is justified by the formalization of how the algorithm should perform (temporal interval traversals are optimal [27]).

The middle and bottom panels compare path quality (accumulated risk) at endpoint 0 across the three algorithms at two

representative vehicle speeds: 1 m/s and 100 m/s. At lower speeds (1 m/s), all algorithms achieve different accumulated risk. At lower speeds, all algorithms are affected by the temporal windows, making it less clear which paths are going to be selected. However, at high speeds (100 m/s), the temporal constraints tighten significantly, reducing the exploration window and allowing all algorithms to reach the end points within the first time window. This result demonstrates the advantage of Bellman-Ford's exhaustive relaxation approach, guarantees to find the optimal lowest accumulated risk in such scenarios.

VI. Conclusions and Future Work

RAPID is a novel risk-aware path planning framework for autonomous aircraft in dynamic urban environments. By integrating temporal risk grids, circle-based kinematic geometry, and time-aware search, it ensures all generated paths are immediately kinematically feasible and strictly satisfy safety constraints. The system achieves a 10–20× speedup over naive implementations, enabling practical city-scale deployment. Pre-flight analysis for environments spanning 3 km² completes in 17.8 seconds while simultaneously computing risk-mitigating paths to four distinct destinations. For mid-flight contingency management, RAPID leverages warm-start initialization, early termination, and edge weight caching to execute rapid replanning (Stage 4) in just 1.8 seconds. This deterministic, unified architecture avoids computational fragmentation and provides a strong foundation for the formal verification required in safety-critical aviation.

Transitioning this framework to real-world operations will require calibrating risk models against historical data, such as weather accuracy and ADS-B collision patterns. Currently, RAPID assumes risk predictions over a fixed forecasting window (e.g., 90 minutes). Ongoing work focuses on developing a fast reactive replanning system that efficiently reuses valid path segments when initial predictions fail. Because RAPID uses the same multi-goal foundation for both predictive pre-flight analysis and dynamic in-flight contingency management, future research will systematically evaluate its performance under simulated pop-up threats. Additional future directions include performance optimization through GPU acceleration, expansion into three-dimensional spatial risk modeling, integration of probabilistic frameworks for forecast uncertainty, and deployment on real aircraft to enable safe, optimal autonomous operations in complex airspaces.

References

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

[2] 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.

[3] S. Aggarwal and N. Kumar, "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges," *Computer communications*, vol. 149, pp. 270–299, 2020.

[4] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[5] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[6] E. Ancel, F. M. Capristan, J. V. Foster, and R. C. Condotta, "Real-time risk assessment framework for unmanned aircraft system (UAS) traffic management (UTM)," in *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017, p. 3273.

[7] J. Gutierrez, S. Young, A. Moore, R. Gilabert, E. Dill, E. Bates, M. Peretic, K. Schmitt, and A. Scholz, "A high-performance computing predictive gnss performance monitor for autonomous air vehicles in urban environments," *NASA Technical Memorandum*, 2024.

[8] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems*, 2008, vol. 521.

[9] C. Muñoz, A. Narkawicz, G. Hagen, J. Upchurch, A. Dutle, M. Consiglio, and J. Chamberlain, "DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 2015, pp. 5A1–1.

[10] J. S. Zelek, "Dynamic path planning," in *IEEE International Conference on Systems Man and Cybernetics*, vol. 2. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1995, pp. 1285–1290.

[11] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[12] J. Gutierrez, N. A. Neogi, D. Kaeli, and E. T. Dill, "A High-Performance Computing GNSS-aware Path Planning Algorithm for Safe Urban Flight Operations," in *AIAA AVIATION 2022 Forum*, 2022, p. 3461.

[13] X. Hu, L. Chen, B. Tang, D. Cao, and H. He, "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles," *Mechanical systems and signal processing*, vol. 100, pp. 482–500, 2018.

[14] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[15] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[16] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a Dubins airplane," in *2007 46th IEEE conference on decision and control*. IEEE, 2007, pp. 2379–2384.

[17] S. Young, E. Ancel, A. Moore, E. Dill, C. Quach, J. Foster, K. Darafsheh, K. Smalling, S. Vazquez, E. Evans *et al.*, "Architecture and information requirements to assess and predict flight safety risks during highly autonomous urban flight operations," NASA Langley Research Center, Tech. Rep., 2020.

[18] E. T. Dill, S. D. Young, and K. J. Hayhurst, "SAFEGUARD: An assured safety net technology for UAS," in *2016 IEEE/AIAA 35th digital avionics systems conference (DASC)*. IEEE, 2016, pp. 1–10.

[19] K. Batonisashvili, "PO-RRT*: Pareto-optimal, safety-aware, multi-objective path planning," Master's thesis, Boston University, 2026.

[20] S. Baik, J. H. Bong, and S. Jeong, "Weight-Incorporating A* Algorithm with Multi-Factor Cost Function for Enhanced Mobile Robot Path Planning," in *Actuators*, vol. 14, no. 8. MDPI, 2025, p. 369.

[21] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2013, pp. 191–199.

[22] D. Devaurs, T. Siméon, and J. Cortés, "Optimal path planning in complex cost spaces with sampling-based algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 415–424, 2015.

[23] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 5628–5635.

[24] D. Kularatne, S. Bhattacharya, and M. A. Hsieh, "Optimal path planning in time-varying flows using adaptive discretization," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 458–465, 2017.

[25] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.

[26] D. Cohen, "Incremental methods for computer graphics," Tech. Rep., 1969.

[27] E. Conrad and A. Dutle, "Formalization of the bellman-ford algorithm for airspace applications," in *55th Southeastern International Conference on Combinatorics, Graph Theory and Computing*, 2024.