Multi-UAV 3D Urban Logistics Task, Path Planning, and Dynamic Obstacle Avoidance Simulation System

1. Overview of Implemented Functions

Competition Requirement		Completed Work
Basic Work	**Urban 3D Scene Setup**	**[Scene Construction]:** ✧ Integrated *OpenStreetMap* building data and *GMTED2010* terrain elevation data to generate a high-fidelity urban geographic model. ✧ Constructed a 3D occupancy grid map (*occupancyMap3D*) of Manhattan, USA, supporting simulation in complex urban environments.
	Single-UAV 3D Path Planning	**[Algorithm Implementation]:** ✧ Applied the *BiRRT* (Bidirectional Rapidly-exploring Random Tree) algorithm for global path planning in the 3D occupancy grid map (*occupancyMap3D*). ✧ Compared multiple algorithms, including *RRT, RRT*, PRM, A*, and BiRRT*, and finally adopted *BiRRT* for its balance between path length and computational efficiency. ✧ Performed polynomial trajectory smoothing based on the Minimum Snap method.。
	Multi-UAV 3D Path Planning	**[Algorithm Implementation]:** ✧ Supported flexible configuration of the number of *egoUAVs*, enabling multi-UAV parallel path planning and collision-free navigation.
Advanced Work	**Dynamic Obstacle State Estimation Based on Simulated Radar**	**[Algorithm Implementation]:** ✧ Integrated radar and *LiDAR* sensor models to perform environmental perception and obstacle detection. ✧ Employed *JPDA* (Joint Probabilistic Data Association) and *PHD* (Probability Hypothesis Density) filters for multi-target tracking, allowing real-time estimation of obstacle positions and velocities.
	Multi-UAV Logistics Task Allocation	**[Algorithm Implementation]:** ✧ Solved the *Multi-Traveling Salesman Problem (MTSP)* using an improved *Genetic Algorithm (GA)* to achieve intelligent multi-task–multi-UAV assignment. ✧ Incorporated multiple realistic constraints—obstacle perception, battery management, and time scheduling—to

Competition Requirement		Completed Work
		dynamically balance task completion time and UAV utilization efficiency. ✧ Introduced a *preventive return-to-base mechanism* and *range constraint strategy* to enhance mission reliability.
	Dynamic and Static Obstacle Avoidance via Decentralized Algorithms	**[Algorithm Implementation]:** ✧ Developed a *prediction model* based on multi-sensor fusion to generate future obstacle trajectories in real time. ✧ Employed multi-level trajectory screening (*kinematic constraints + collision checking + cost optimization*) to identify feasible avoidance paths. ✧ Adopted an *MPC-like framework* (short-horizon execution + frequent replanning) to achieve real-time obstacle response and continuous path correction in dynamic environments.
	Full-City Multi-UAV Logistics Simulation	**[Integrated Testing]:** ✧ Integrated the urban 3D environment with the multi-UAV system to build a collaborative simulation platform for Manhattan's urban district. ✧ Supported simultaneous multi-UAV mission execution, real-time obstacle avoidance, and dynamic visualization.

2. Technical Completeness and Innovation

2.1 Algorithmic Level

（1） High-Fidelity Urban Simulation Environment Construction

The system integrates OpenStreetMap building data and GMTED2010 terrain elevation data to construct a realistic 3D Manhattan simulation environment based on *occupancyMap3D*. This environment preserves authentic building layouts and terrain variations while maintaining computational tractability, providing a complex and reproducible experimental scene for multi-UAV path planning, task allocation, and obstacle avoidance algorithms.

（2） Comparative Analysis of Global Path Planning Algorithms

On the constructed Manhattan 3D occupancy grid map, five global path planning algorithms—**RRT, RRT*, PRM, A*, and BiRRT**—were systematically compared. Considering path length, planning time, and stability, **BiRRT** was selected as the primary algorithm. This method can efficiently search for feasible paths in complex urban environments, achieving a balance between planning accuracy and computational efficiency, and is well-suited for large-scale 3D multi-UAV mission scenarios.

Tab 1 Performance Comparative Analysis of Five Path Planning Algorithms

Algorithm	Avg Time (sec)	Time Range (sec)	Typical Waypoints	Best Feature
RRT	0.15	0.06 - 0.31	13 - 26	Balanced performance
BiRRT	**0.04**	0.02 - 0.05	7 - 14	**Fastest & Most efficient**
RRTStar	0.30	0.07 - 0.89	14 - 29	Moderate speed
PRM	0.03	0.01 - 0.12	26 - 101	Fast but long detours (2.45× - 4.47×)
3D A*	18.51	10.10 - 133.94	72 - 159	**Slowest but optimal paths**

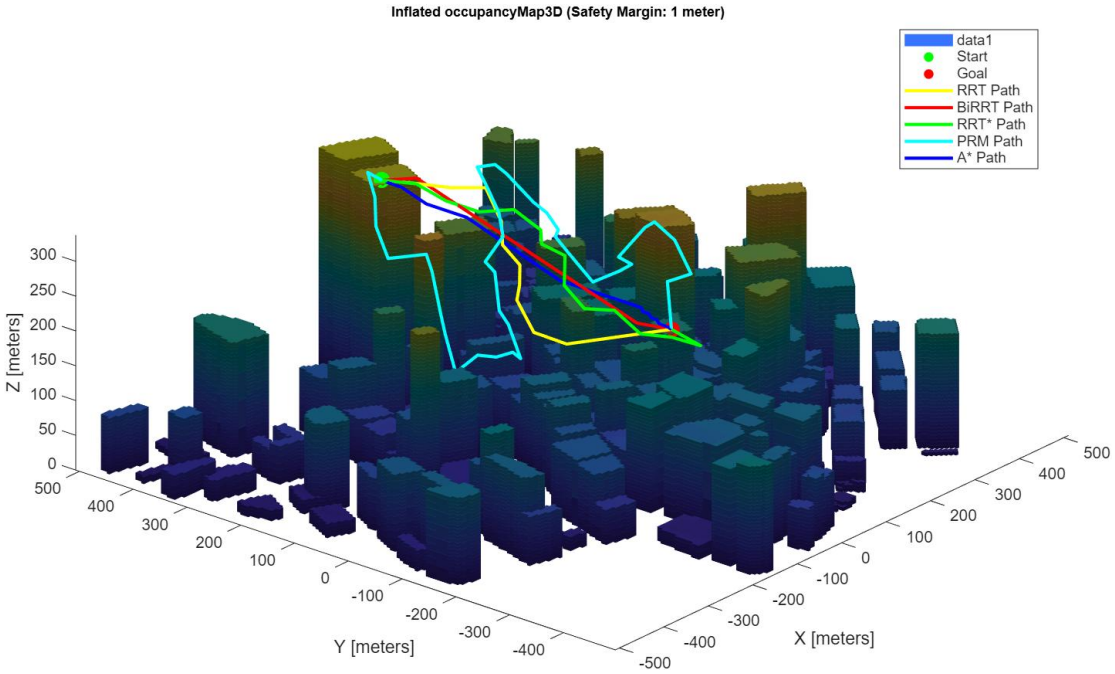


Fig 1. Path planning results of the five algorithms for the same start and end points.

（3） Multi-UAV Task Allocation Under Realistic Constraints

- **Task Planning with Endurance and Charging Constraints:** To address UAV battery limitations, multiple charging stations are preset in the urban scene to enable in-task recharging and path relays, ensuring feasibility for long-duration missions.
- **MTSP-Based Task Allocation Modeling:** Multi-task and multi-UAV scheduling is formulated as a **Multi-Traveling Salesman Problem (MTSP)**. The optimization objective minimizes total flight distance and task completion time while introducing constraints on energy limits, task reachability, and charging station locations.
- **Improved Genetic Algorithm (GA) Optimization:** An enhanced GA is used to search for the optimal allocation scheme. Through individual encoding and adaptive crossover–mutation strategies, global search efficiency and solution stability are improved.
- **Adaptive Weight Coefficient Mechanism:** The fitness function dynamically adjusts weight coefficients to balance flight distance, energy consumption, and task duration constraints, allowing flexible scheduling strategies under different operational scenarios.
- **Energy Prediction and Preventive Return Mechanism:** Before executing each task, the system predicts remaining battery capacity. If the predicted value falls below a 20% safety threshold, the UAV automatically returns to a charging station. This mechanism prevents mission interruptions and enhances system safety and reliability.

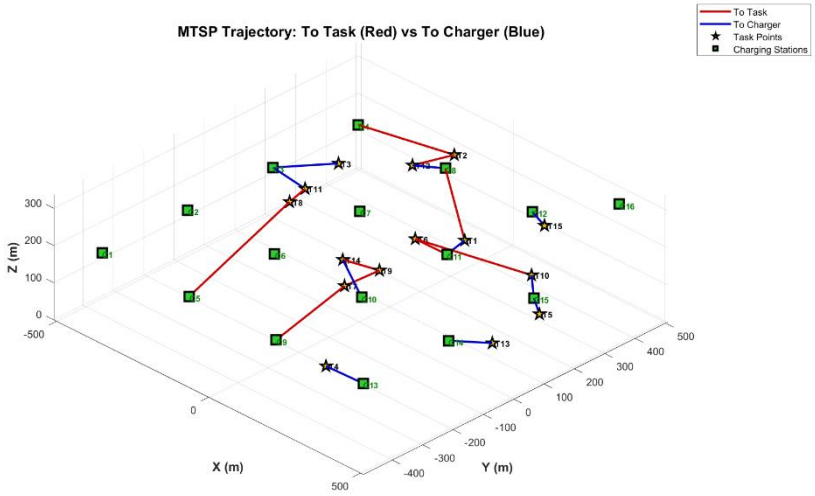


Fig 2. MTSP Task Planning Results for a Scenario with 16 UAVs and 15 Tasks

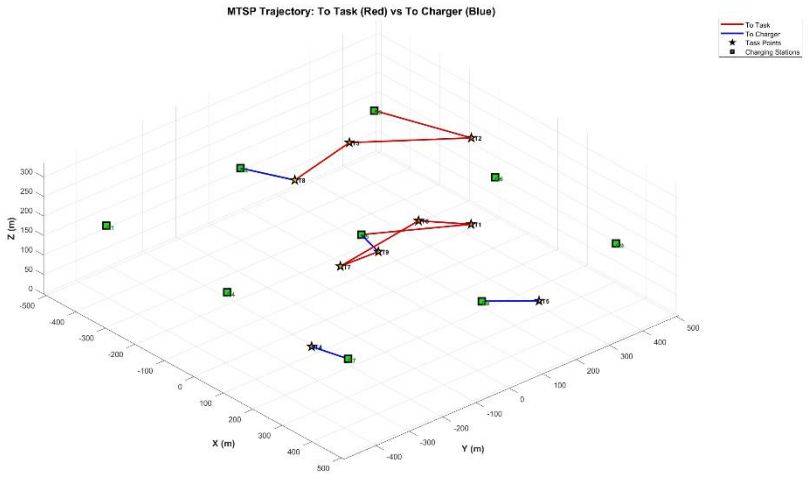


Fig 3. MTSP Task Planning Results for a Scenario with 9 UAVs and 9 Tasks

（4） Dynamically Feasible Trajectory Smoothing Algorithm

Based on the **Minimum Snap** method, the system smooths the segmented trajectories generated by *BiRRT*. A seventh-order polynomial model and quadratic programming optimization are used to produce time–space continuous trajectories. The optimization simultaneously constrains velocity and acceleration, ensuring dynamic feasibility while maintaining path smoothness under real urban conditions.

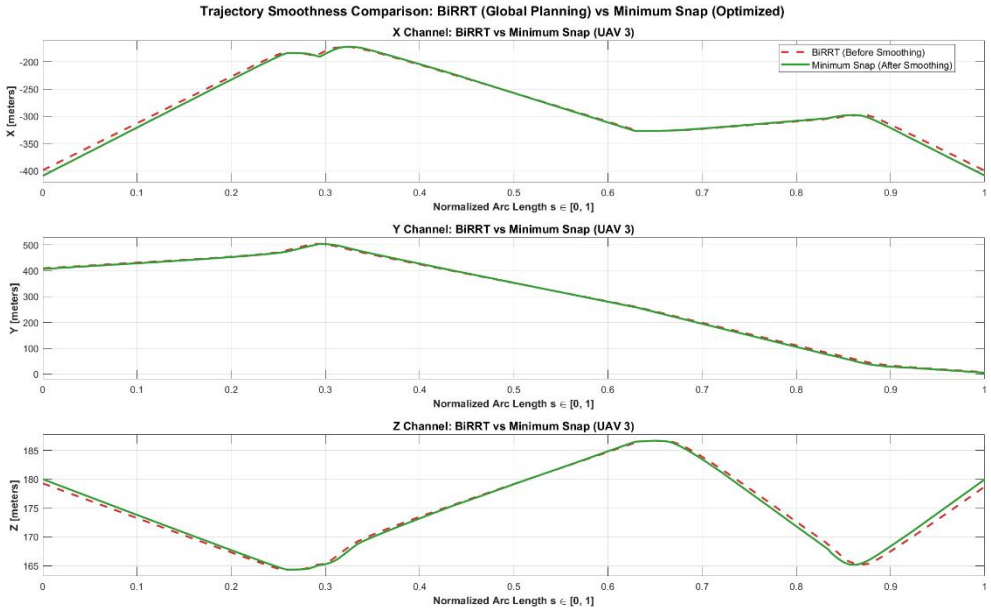


Fig 4. Comparison of Original BiRRT and Optimized Trajectories Across X, Y, and Z Axes

2.2 System-Level Design

（1） Efficient Local Dynamic Obstacle Avoidance System

A decentralized local obstacle avoidance framework is developed, enabling each UAV to independently perceive its environment and plan its own avoidance trajectory. After fusing radar and LiDAR data, the system predicts the motion states of surrounding obstacles using trackers. The avoidance planner conducts "**line–line**" collision detection directly between the UAV's own trajectory and the predicted trajectories of other agents, replacing the traditional "**trajectory–voxel map**" method. This approach significantly reduces computational complexity and improves real-time performance.

（2） MPC-like Short-Horizon Execution and Frequent Replanning Framework

The system adopts a "Perception–Prediction–Planning–Execution" loop structure. Each UAV executes only a short horizon of its current trajectory (*controlSteps*) rather than the entire selected *fullTrajectory*. Immediately afterward, a new planning cycle begins, enabling rapid response to dynamic obstacles and continuous path correction. This ensures stable operation in complex and changing environments.

（3） Smooth and Safe Trajectory Management Mechanism

Within the main simulation loop of *manhattan_3d_planning.m*, the system ensures continuity and physical realism of flight. When the UAV approaches the end of a current trajectory segment, the next trajectory is preloaded to maintain smooth motion. Before reaching the final segment, the UAV automatically decelerates and stops in accordance with physical dynamics. Once the UAV moves sufficiently away from old segments, historical data are cleared to release memory, achieving seamless trajectory transitions and efficient resource management.

（4） Highly Parameterized and Robust System Architecture

The main file *manhattan_3d_planning.m* adopts a highly parameterized architecture, allowing flexible adjustment of core parameters such as task scale, UAV configuration, and environmental resolution. This design minimizes code modification and enhances reusability and adaptability. During operation, the system provides interactive prompts and fallback mechanisms

for potential issues such as planning failure, path conflicts, or perception errors, ensuring stable and continuous performance under complex environmental conditions.

3. Detailed Implementation Process

The system, centered on the main program *manhattan_3d_planning.m*, integrates modules for **urban environment construction, task generation and allocation, global path planning, trajectory smoothing**, and **local dynamic obstacle avoidance simulation**, forming a complete workflow from environmental setup to execution.

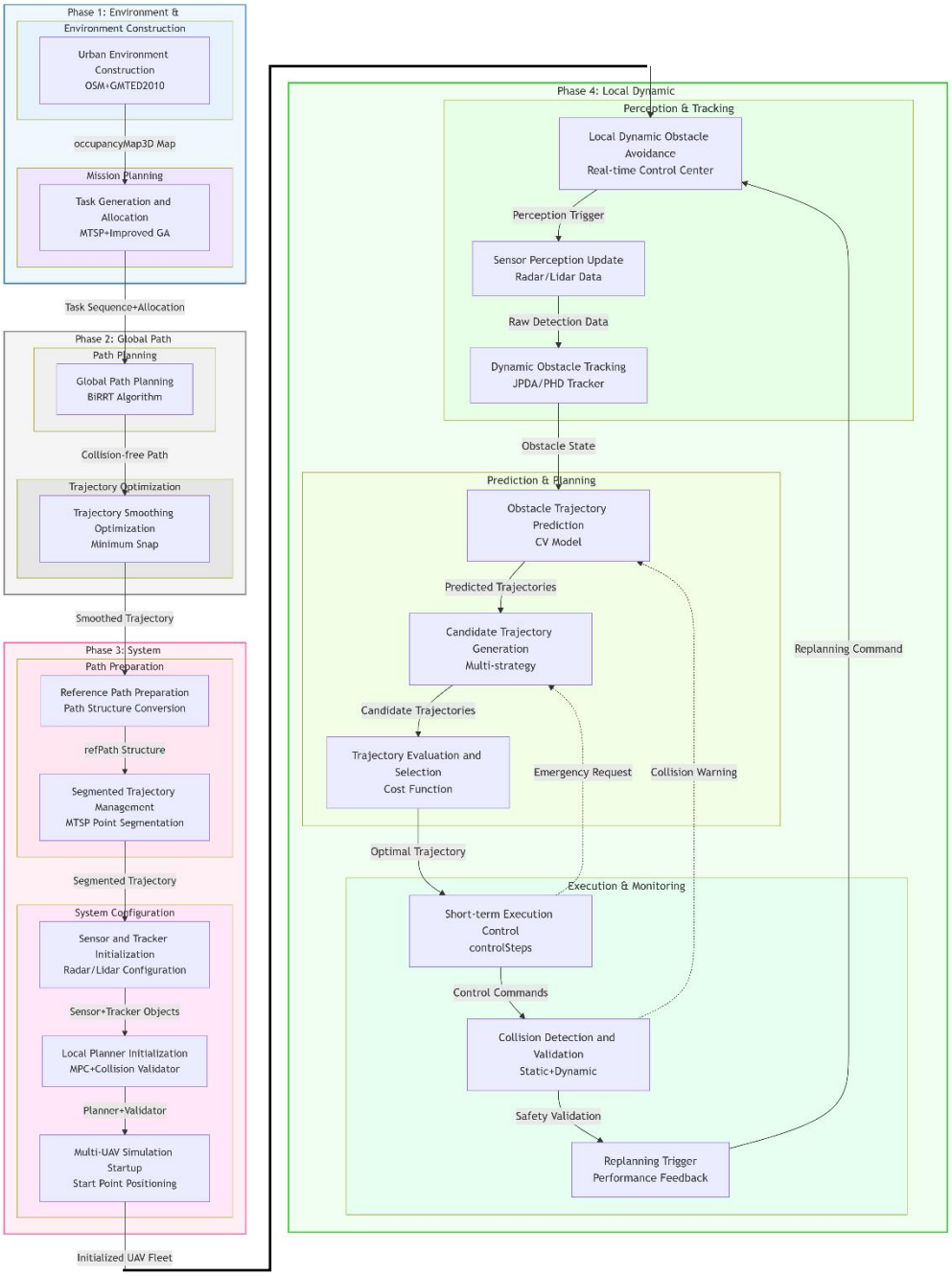


Fig 5. Functional Block Diagram of the System

3.1 Urban Environment Construction

The project builds a 3D path-planning simulation environment based on real Manhattan geographic data. The system integrates *OpenStreetMap* building data and *GMTED2010* terrain elevation data to generate the *occupancyMap3D*.

The process includes:

- Extracting building contours through *XY projection* (using *convhull*, *polygon*, or *rectangular* approximation).
- Assigning voxel values layer by layer across height levels using *inpolygon*, achieving voxel-based modeling.
- Applying *adaptive grid subdivision* and a *safety-margin inflation mechanism* to construct an obstacle map supporting complex 3D navigation and collision checking.

The final 3D urban scene satisfies simulation and planning requirements in terms of scale, structure, and resolution.

3.2 Task Generation and Allocation

3.2.1 Charging-Station Placement

The system employs an adaptive layout strategy to automatically determine optimal charging-station distributions based on the number of stations. When the number is fewer than four, stations are placed at the scene corners; for nine or sixteen, 3×3 and 4×4 grid layouts are used, respectively. Other quantities are arranged using a square-root-based adaptive uniform grid. All stations are set to the same altitude and distributed evenly within scene boundaries, reserving 50–100 m of clearance for safety. After generation, a collision-checking procedure verifies placement validity; if overlap with buildings or obstacles is detected, station altitudes are automatically raised near the top of the scene to ensure safe and valid positioning.

3.2.2 Task Definition and Generation

(1) Task Definition: A task is defined as a UAV departing from a charging station to deliver to a 3D spatial target. Data are stored in an N × 3 double matrix *tasks_3d*, where each row *[x, y, z]* represents the coordinates of an individual delivery target. This definition corresponds to real-world logistics scenarios where UAVs deliver packages from warehouses to distributed customer points.

(2) Task Generation: The module generates task points within a predefined 3D workspace according to configuration parameters. Candidate coordinates are first generated via random sampling, and each is checked with *checkOccupancy* for collisions. Only samples located in free space (occupancy < 0.5) are retained. To prevent infinite loops in dense environments, a parameter maxAttempts = 100 × number of tasks limits maximum sampling attempts, ensuring robustness and stability of task generation.

3.2.3 Task Allocation

(1) Function Definition and General Description: The file *MTSP.m* defines a custom function *MTSP*, which solves the *Multi-Traveling-Salesman Problem (MTSP)* to automatically allocate multiple task points to multiple UAVs. An improved *Genetic Algorithm (GA)* serves as the core optimization engine, jointly considering obstacle awareness, energy limits, and time constraints to produce feasible global task-allocation schemes.

(2) Obstacle-Aware 3D Distance Model: The system replaces conventional Euclidean distance with an obstacle-aware distance model combining *Manhattan distance* and a

vertical-flight penalty factor. Occupancy is sampled along the line between two points (*manhattan_obstacles mode*). A penalty coefficient proportional to the sampled occupancy ratio modifies the line distance, yielding a more realistic cost estimation under urban constraints.

(3) Fitness Function with Multiple Constraints: The GA fitness function *computeFitness_MTSP* evaluates each individual according to multiple real-world constraints:

- **Multi-Objective Optimization:** jointly optimizes UAV utilization, total flight distance, and mission completion time using weight coefficients *(q1, q2, q3)* to adjust optimization bias.
- **Energy and Range Management:** initializes each UAV with a maximum range *maxRange* and checks remaining range before every leg; if insufficient, triggers an emergency return.
- **Preventive Return Mechanism:** after each task, predicts post-next-leg battery; if expected remaining charge < 20 %, initiates proactive return for recharging.
- **Maximum Endurance Constraint:** enforces *max_endurance* to guarantee safe return even under battery degradation.

(4) Robustness Assurance: If the GA fails to find a feasible solution within the iteration limit, the system automatically falls back to random allocation to ensure continuity of the simulation.

3.3 Global Path Planning Based on BiRRT

(1) Planning Workflow: After task allocation, the system performs path planning sequentially for each UAV. Each path segment connects a pair of adjacent task points *(sp→gp)*; if their altitude difference exceeds 150 m, an intermediate waypoint is inserted for sub-segmented planning. The program calls *birrt_plan_segment* (defined in *birrt_plan_segment.m*) to generate a feasible polyline path for each segment and stores it as a BiRRT sub-path. All sub-paths are concatenated to form the UAV's full reference path *globalRefPaths{u}*.

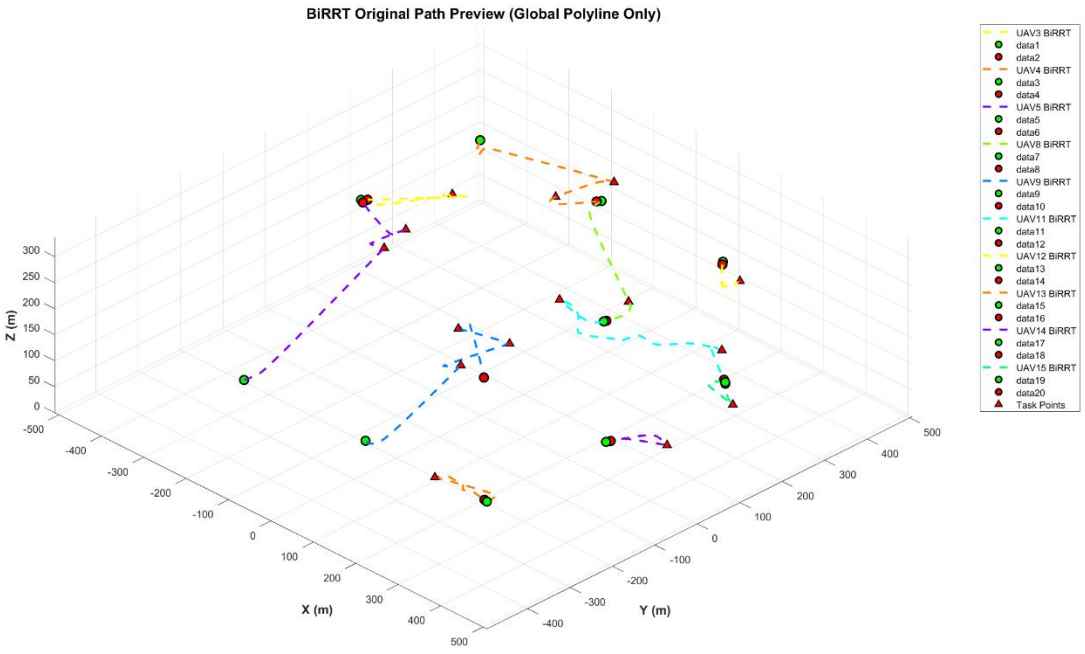


Fig 6. Global BiRRT Planned Piecewise Linear Path for 15 Tasks and 16 UAVs

(2) Multi-Level Safety Enhancements: In the custom *birrt_plan_segment* function, several additional safety checks and corrections enhance path feasibility and stability:

- **Loop Detection and Removal:** removeLoops(path) identifies redundant loops; if any segment within 30 points has < 20 m distance, it is skipped and connected to the loop's end.
- **Adaptive Density Interpolation:** re-samples at 8 m spacing via *interpPathPoints(path, 8)* to ensure uniform point density.
- **Smoothing with Detail Preservation:** applies a moving-average filter (window = 3) to smooth while preserving obstacle-avoidance detail.
- **Strict Collision Validation:** *validateAndFixPath(path, omap)* and *checkLineCollision(p1, p2, omap)* check each segment for occupancy > 0.5 at 2 m intervals to prevent wall penetration.
- **Bypass and Re-planning Strategy:** for detected collisions, *findBypassPoint(p1, p2, omap)* attempts local repair through a three-step hierarchy: XY-plane lateral shift → Z-axis elevation → segment re-planning, ensuring feasible routes even in dense obstacles.

3.4 Trajectory Smoothing via the Minimum Snap Method

(1) Function Overview: The custom function *buildSegmentedMinSnapPaths* (in *buildSegmentedMinSnapPaths.m*) performs Minimum Snap optimization on each BiRRT segment. For every segment, it executes key-point selection, time allocation, Minimum Snap solving, and trajectory merging, producing time- and space-continuous reference paths.

(2) Key-Point Selection: *selectKeyPointsByCurvature* extracts key points from each BiRRT polyline—adding points on long or highly curved paths to preserve geometry and reducing them on short straight paths for efficiency.

(3) Time Allocation and Minimum Snap Solution: *adaptiveTimeAllocation* assigns flight time to each segment according to distance, UAV velocity direction, and kinematic limits (velocity, acceleration, jerk). Then *solveMinimumSnapForUAV* optimizes a seventh-order polynomial trajectory:

- *computeA* builds the quadratic cost matrix A;
- *buildLocalConstraintData* and *computeConstraint* construct equality/inequality constraints for position and derivative continuity;
- *quadprog* solves polynomial coefficients;
- *extractTrajectoryFromSolution*_new samples the trajectory in time;
- *validateTrajectory* verifies velocity, acceleration, and continuity for dynamic feasibility.

(4) Post-Processing and Merging:

After optimization, *removeConsecutiveDuplicates* (in manhattan_3d_planning.m) removes consecutive duplicates and merges all smoothed segments into *optimizedRefPaths{u}*, serving as the final reference trajectory.

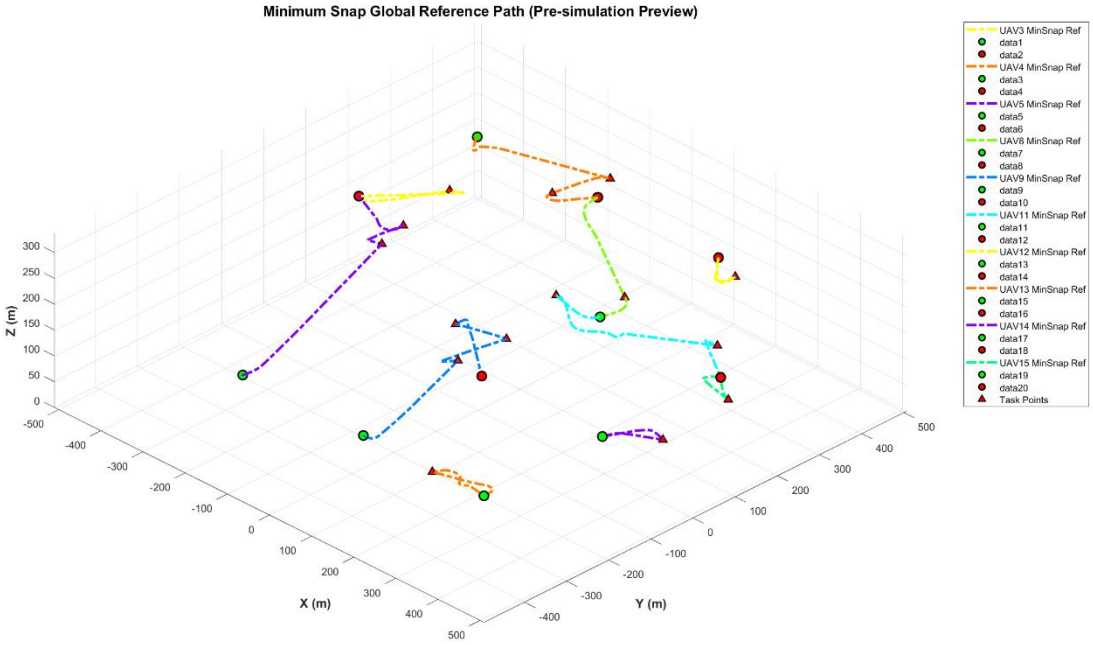


Fig 7. A Minimum-Snap Smoothed Global Reference Path for 15 Tasks and 16 UAVs

3.5 Local Dynamic Obstacle Avoidance

(1) Perception and Tracking: LiDAR and radar sensors periodically sense the environment, updating detections every SENSOR_UPDATE_INTERVAL. Detected point clouds and radar echoes are clustered and passed to their respective trackers (*lidarJPDA*, *radarPHD*). Trackers output dynamic-object arrays *objectTrack* (*lidarTracks*, *radarTracks*), implemented by *updateLidarTracker* and *updateRadarTracker*.

(2) Obstacle-Trajectory Prediction: The custom function *predictTracks* (in *predictTracks.m*) uses a Constant-Velocity (CV) model to predict obstacle motion from State *[x, y, z, vx, vy, vz]*, *StateCovariance* (6×6), and *TrackID*. *updateCollisionValidator* stores the predicted positions in *collisionValidator.predictedObstacles* for later collision assessment.

(3) Candidate-Trajectory Generation: The planner object planner (created by *helperCreateTrajectoryPlanner*) calls *generateCandidateTrajectories* to create multiple candidate trajectories *candidateTrajectories* around the reference path, sampling in speed and lateral offset dimensions to enhance avoidance flexibility.

(4) Trajectory Feasibility and Collision Checking: Generated candidates undergo multi-stage filtering:

- Kinematic-Constraint Filtering: *checkKinematicFeasibility* removes trajectories exceeding speed or acceleration limits.
- Static-Environment Collision Check: occupancy queries in occupancyMap3D eliminate paths intersecting buildings.
- Dynamic-Obstacle Conflict Check: if at any time the UAV trajectory approaches a predicted obstacle trajectory within *safetyMargin*, a collision (or collision probability) is recorded.

After filtering, the safe-trajectory set *safeTrajectories* remains, ensuring feasibility in both static and dynamic contexts.

(5) Cost Evaluation and Trajectory Selection: *calculateTrajectoryCosts* evaluates each *safeTrajectory*, considering progress along reference, deviation, comfort/energy cost, terminal velocity, continuity, and collision risk. The trajectory with the lowest cost is chosen as the execution path *fullTrajectory*.

(6) Short-Horizon Execution and Closed-Loop Re-Planning: The UAV executes only the first portion of *fullTrajectory* (*controlSteps = replanSteps + 1*) before re-entering the perception–prediction–planning–execution loop. This *MPC-like (Model Predictive Control)* mechanism enables continuous adjustment to new obstacles and environmental changes.

The avoidance planner directly performs line-to-line collision evaluation between self-trajectories and predicted others' trajectories, replacing traditional *trajectory-to-voxel (line-to-volume)* pointwise checking, thereby significantly improving real-time efficiency in local re-planning.

4. Results and Demonstrations

4.1 Task Execution Process Demonstration

This section corresponds to the simulation workflow shown in the overall framework diagram of Section 3.

4.1.1 Case 1: 15 Task Points, 16 Charging Stations, 16 UAVs, Simulation Time = 80 s

- Comparison between Reference and Actual Trajectories (Length and Number of Waypoints):

```
[1/2] Reference Path vs Actual Flight Statistics
  UAV1: No reference path
  UAV2: No reference path
  UAV3:
    Reference Path: 364.2m (773 points)
    Actual Flight: 363.4m (400 points)
    Total Length Deviation: 0.8m (0.2%)
  UAV4:
    Reference Path: 531.8m (1098 points)
    Actual Flight: 531.5m (400 points)
    Total Length Deviation: 0.3m (0.1%)
  UAV5:
    Reference Path: 784.9m (1625 points)
    Actual Flight: 783.6m (400 points)
    Total Length Deviation: 1.3m (0.2%)
  UAV6: No reference path
  UAV7: No reference path
  UAV8:
    Reference Path: 433.6m (876 points)
    Actual Flight: 433.5m (400 points)
    Total Length Deviation: 0.0m (0.0%)
  UAV9:
    Reference Path: 759.1m (1597 points)
    Actual Flight: 757.2m (400 points)
    Total Length Deviation: 1.9m (0.2%)
  UAV10: No reference path
  UAV11:
    Reference Path: 618.3m (1275 points)
    Actual Flight: 618.2m (400 points)
    Total Length Deviation: 0.1m (0.0%)
  UAV12:
    Reference Path: 228.3m (461 points)
    Actual Flight: 225.4m (400 points)
    Total Length Deviation: 3.0m (1.3%)
  UAV13:
    Reference Path: 356.6m (717 points)
    Actual Flight: 356.2m (400 points)
    Total Length Deviation: 0.3m (0.1%)
  UAV14:
    Reference Path: 237.2m (504 points)
    Actual Flight: 237.1m (400 points)
    Total Length Deviation: 0.1m (0.0%)
  UAV15:
    Reference Path: 152.2m (340 points)
    Actual Flight: 151.4m (400 points)
    Total Length Deviation: 0.9m (0.6%)
  UAV16: No reference path
```

Fig 8. Reference vs Actual Trajectories — Length Statistics

- Visualization of Reference and Actual Trajectories: (The reference trajectories are shown as dashed lines, and the actual UAV trajectories as solid lines.):

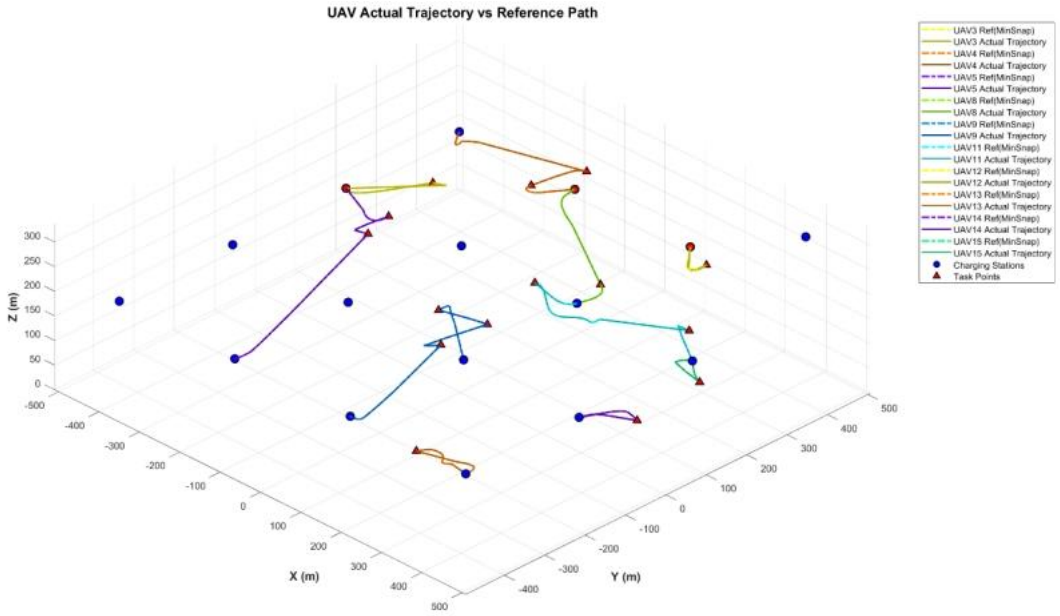


Fig 9. Reference and Actual Trajectories for 15 Tasks and 16 UAVs

- Display of Reference and Actual Trajectories in the Manhattan OccupancyMap3D Scene:

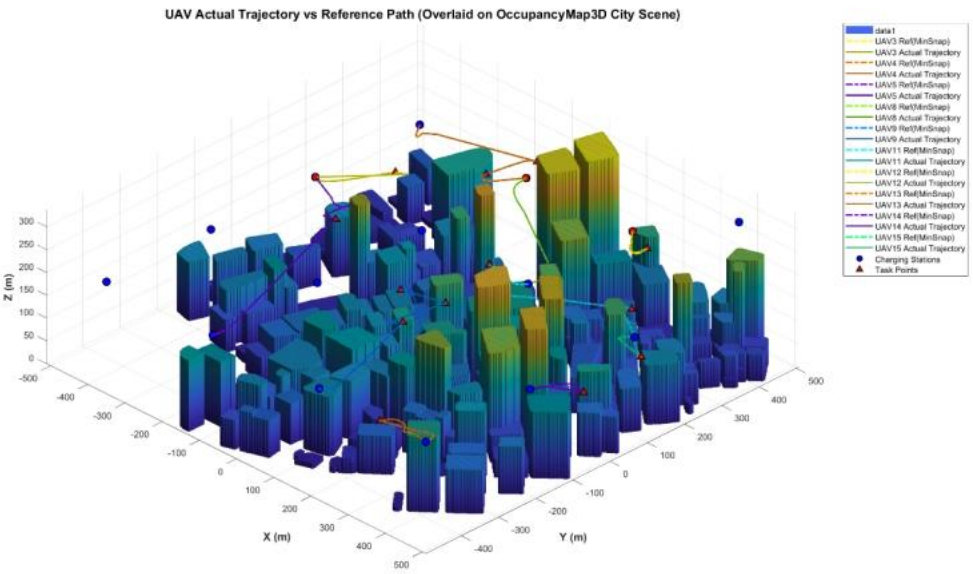
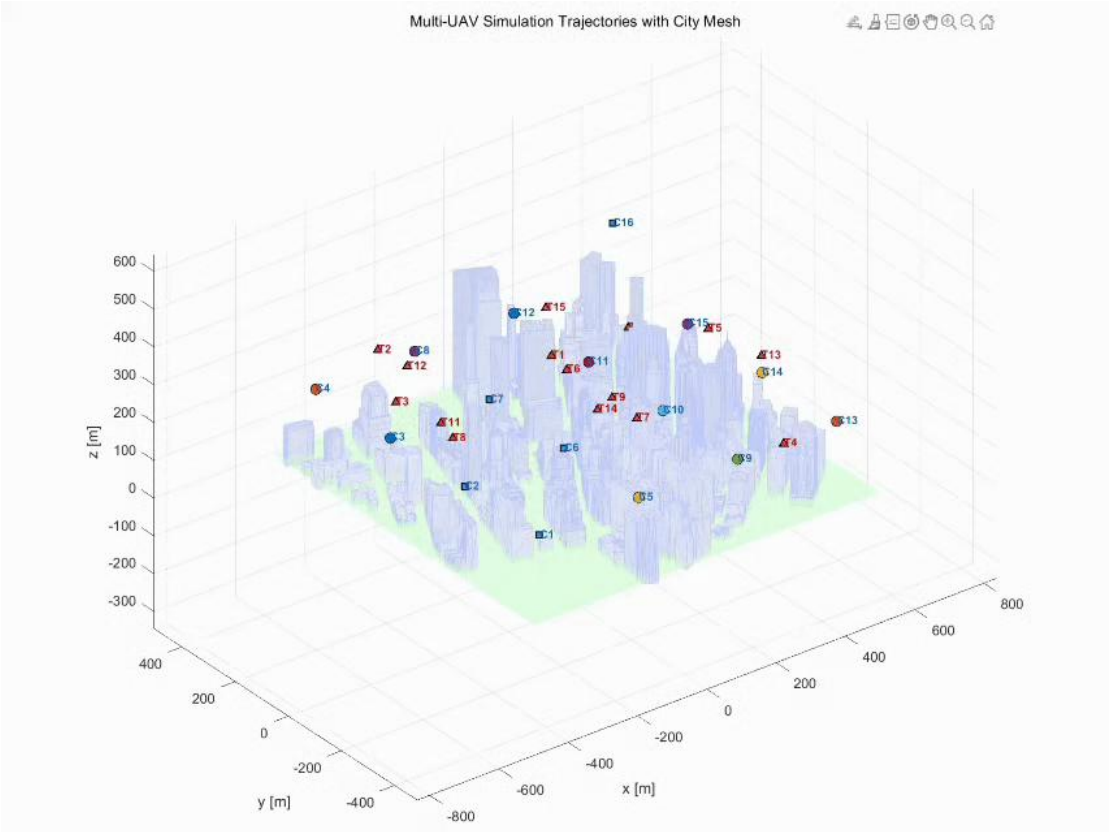
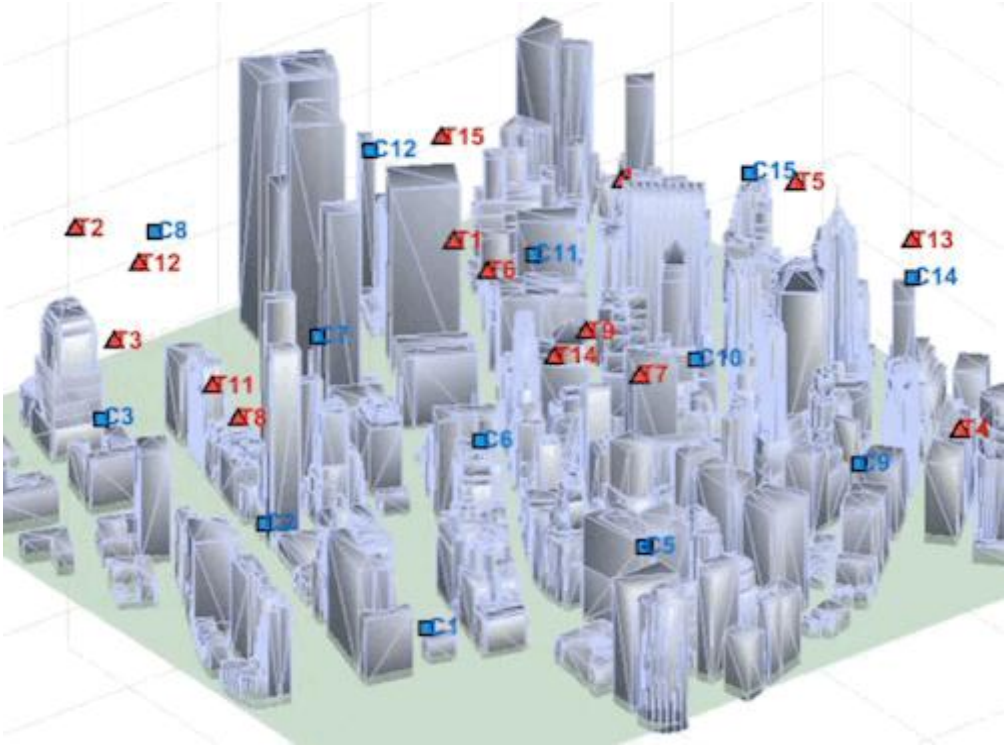


Fig 10. Reference and Actual UAV Trajectories in OccupancyMap3D (15 Tasks, 16 UAVs)

- Animated Demonstrations:

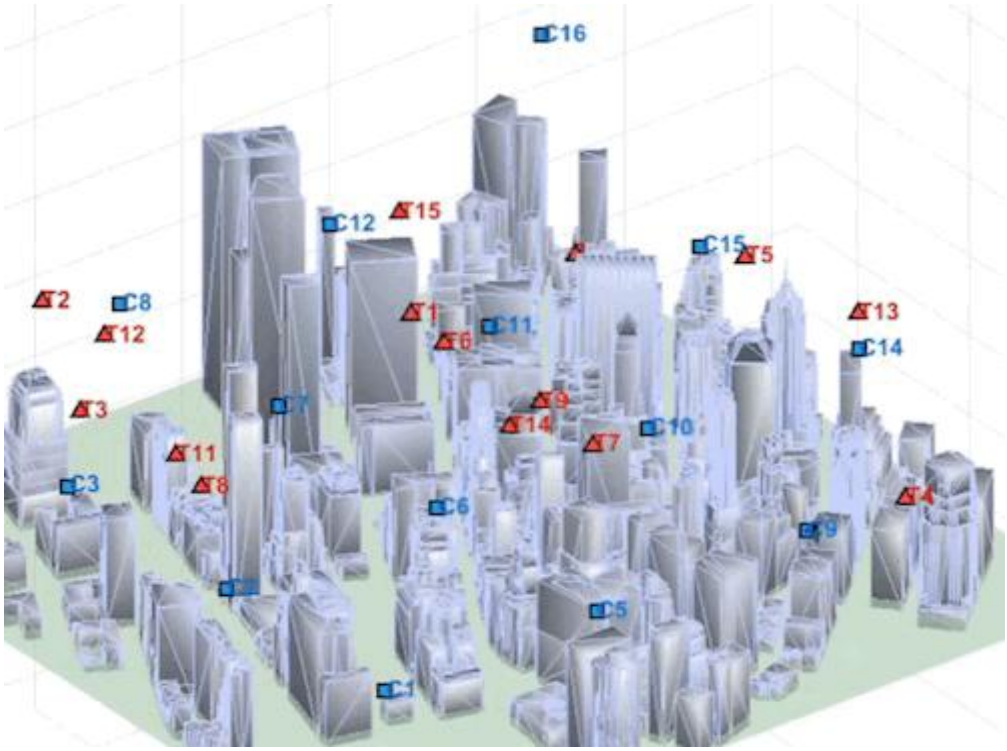


Animation 1: All UAVs executing 15 tasks with 16 UAVs in total.

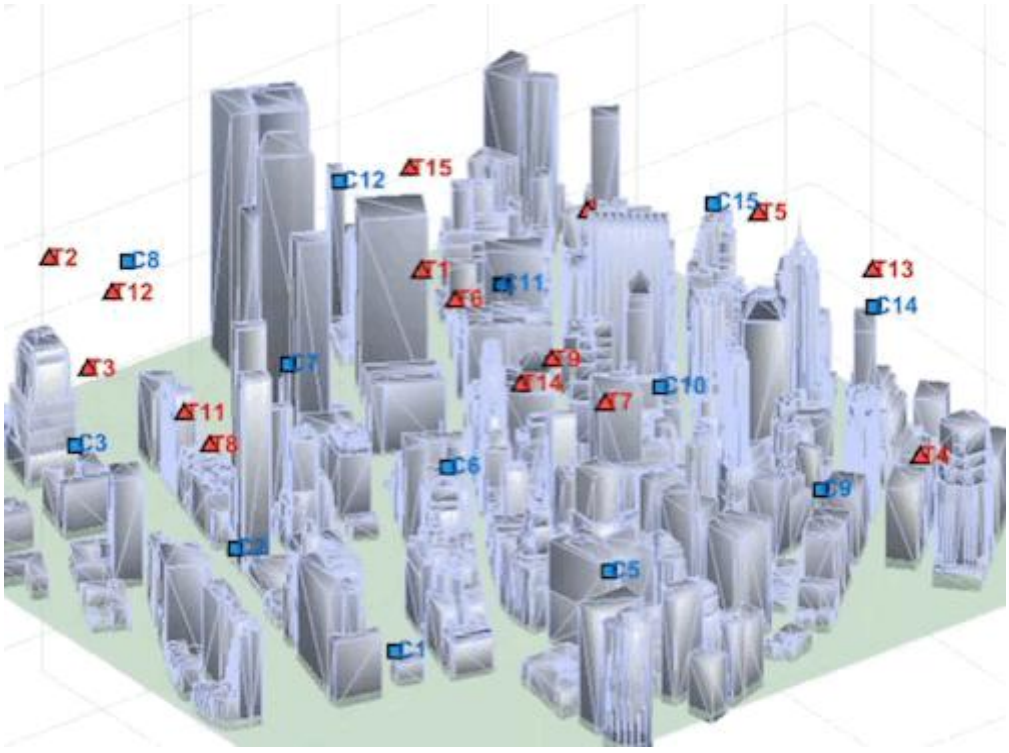


Animation 2: Single UAV (UAV 5) performing its assigned task in the case of 15 task points

and 16 UAVs.



Animation 3: Single UAV (UAV 9) performing its assigned task in the case of 15 task points and 16 UAVs.



Animation 4: Single UAV (UAV 11) performing its assigned task in the case of 15 task points and 16 UAVs.

4.1.2 Case 2: 9 Task Points, 9 Charging Stations, 9 UAVs, Simulation Time = 100 s

- Comparison between Reference and Actual Trajectories (Length and Number of Waypoints):

```
UAV1: No reference path
UAV2: No reference path
UAV3:
  Reference Path: 913.4m (1887 points)
  Actual Flight: 911.6m (500 points)
  Total Length Deviation: 1.8m (0.2%)
UAV4: No reference path
UAV5:
  Reference Path: 1036.6m (2146 points)
  Actual Flight: 1037.5m (500 points)
  Total Length Deviation: 0.9m (0.1%)
UAV6: No reference path
UAV7:
  Reference Path: 317.5m (684 points)
  Actual Flight: 317.2m (500 points)
  Total Length Deviation: 0.3m (0.1%)
UAV8:
  Reference Path: 319.8m (676 points)
  Actual Flight: 319.3m (500 points)
  Total Length Deviation: 0.5m (0.1%)
UAV9: No reference path
```

Fig 11. Reference vs Actual Trajectories — Length Statistics

- Visualization of Reference and Actual Trajectories: (The reference trajectories are shown as dashed lines, and the actual UAV trajectories as solid lines.)

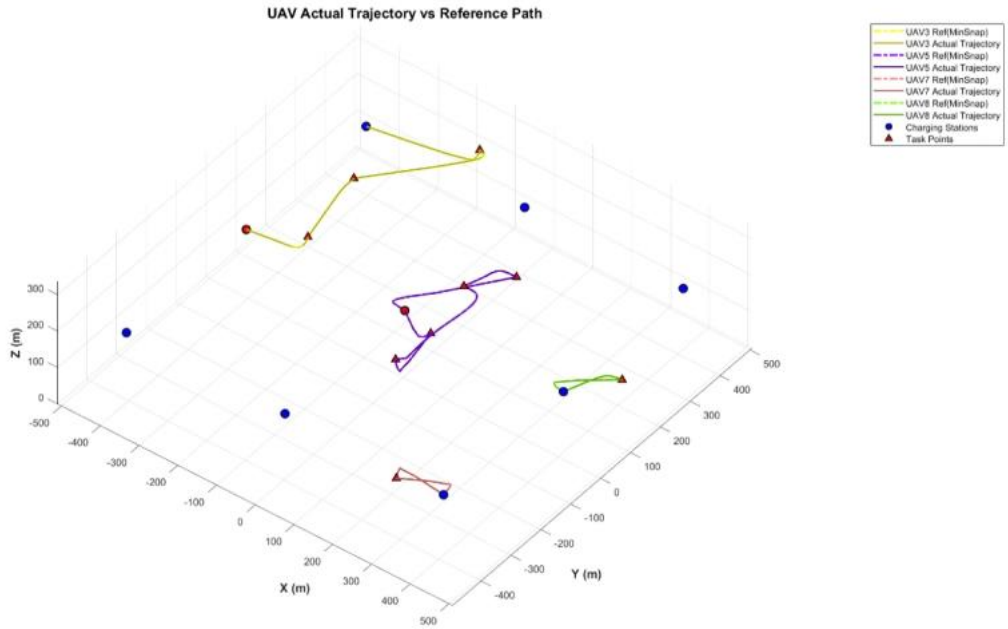


Fig 12. Reference and Actual Trajectories for 9 Tasks and 9 UAVs

- Display of Reference and Actual Trajectories in the Manhattan OccupancyMap3D Scene:

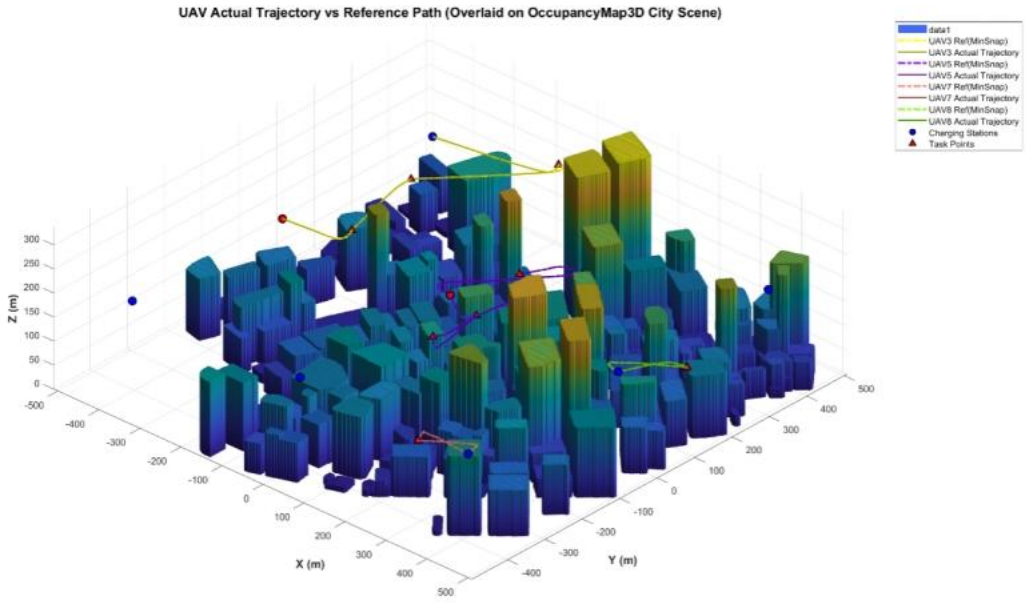


Fig 13. Reference and Actual UAV Trajectories in OccupancyMap3D (9 Tasks, 9 UAVs)

4.2 Local Dynamic Obstacle Avoidance Demonstration

This part corresponds to Section 3.5 Local Dynamic Obstacle Avoidance. In the large-scale Manhattan map used in Section 3.5, UAVs did not actually experience trajectory conflicts. To verify the local obstacle-avoidance capability, we manually created test scenarios where two UAV trajectories intersect (minimum safety distance = 6.00 m). The simulation results confirm the effectiveness of our local dynamic obstacle avoidance module.

Test Setup

- **Trigger Condition:** When the real-time distance between two UAVs becomes smaller than 6.00 m, the local dynamic obstacle avoidance module is triggered and begins decision-making intervention.
- **Verification Objective:** To evaluate whether the system can detect predicted conflicts within the look-ahead time horizon under various approach angles and generate avoidance trajectories that maintain safe separation, thereby preventing collisions.

Test Results

(1) Head-on Encounter (Trajectory Intersection Angle = 0°)

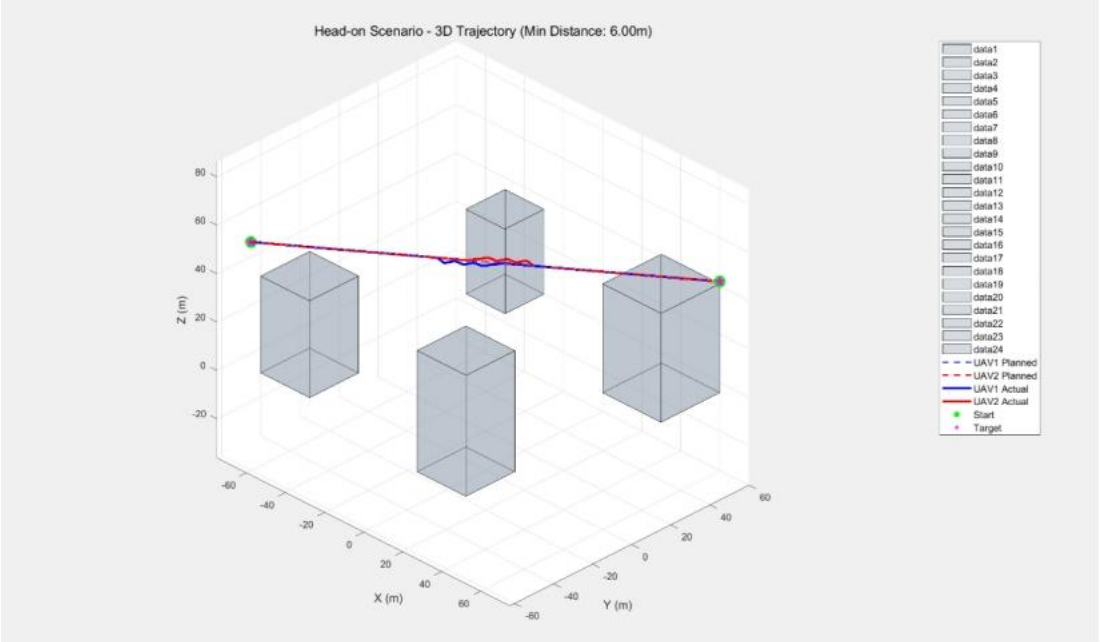


Figure 14. Avoidance after Collision Detection — 3D Trajectory View (0° Intersection)

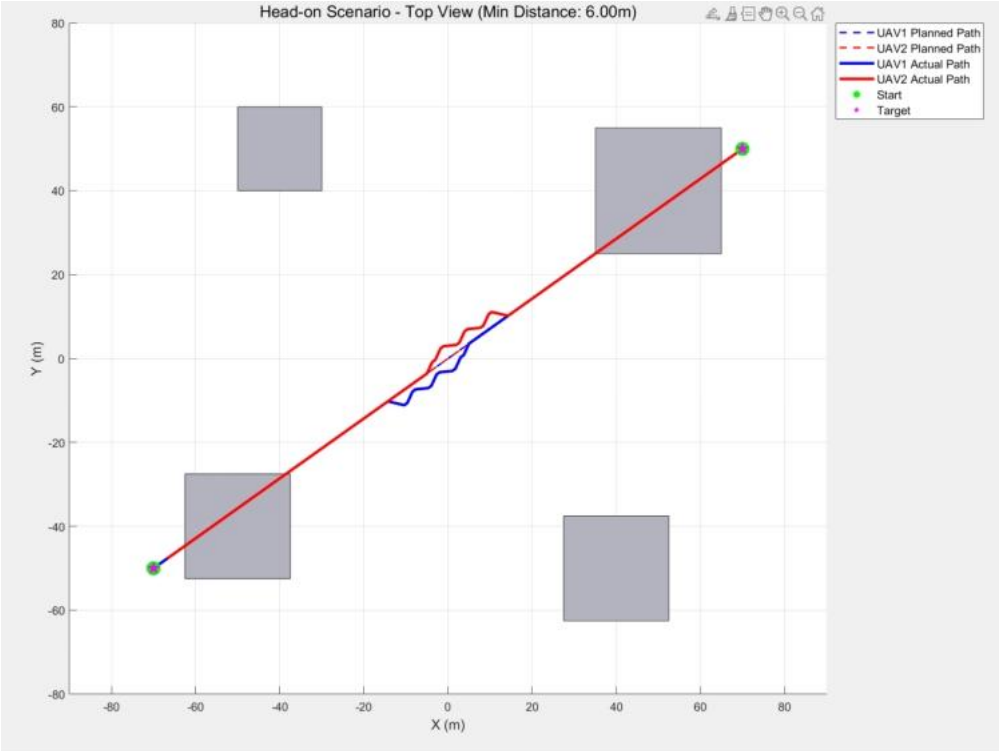


Figure 15. Avoidance after Collision Detection — XY-Plane View (0° Intersection)

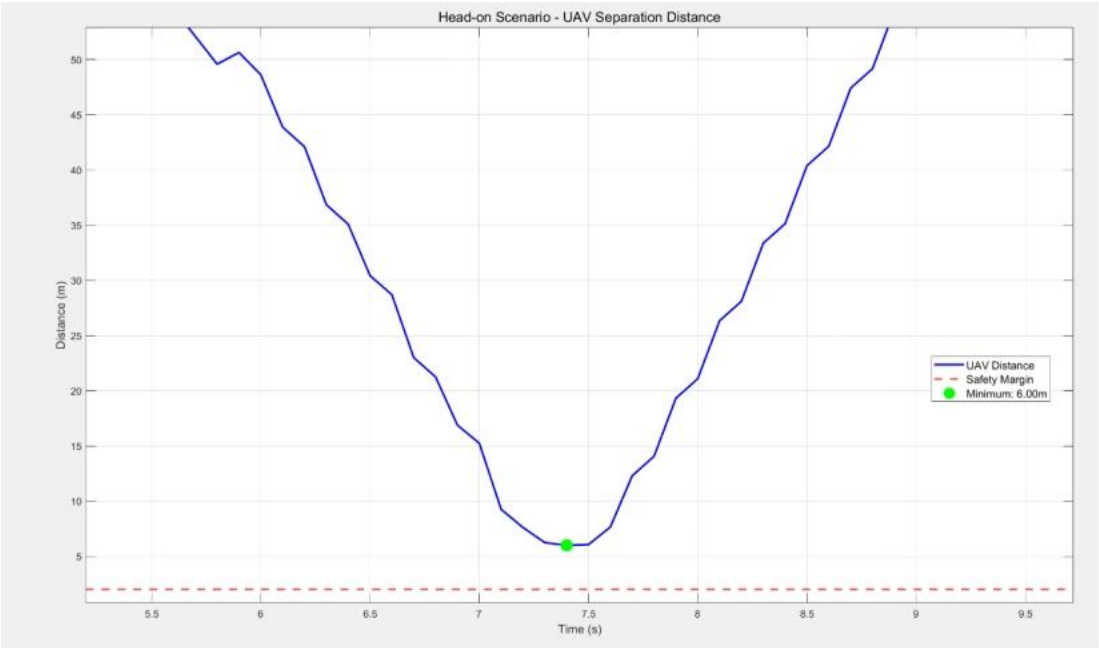


Figure 16. Distance Between Two UAVs vs. Time (0° Intersection)

(2) Oblique Crossing (Trajectory Intersection Angle = 30°)

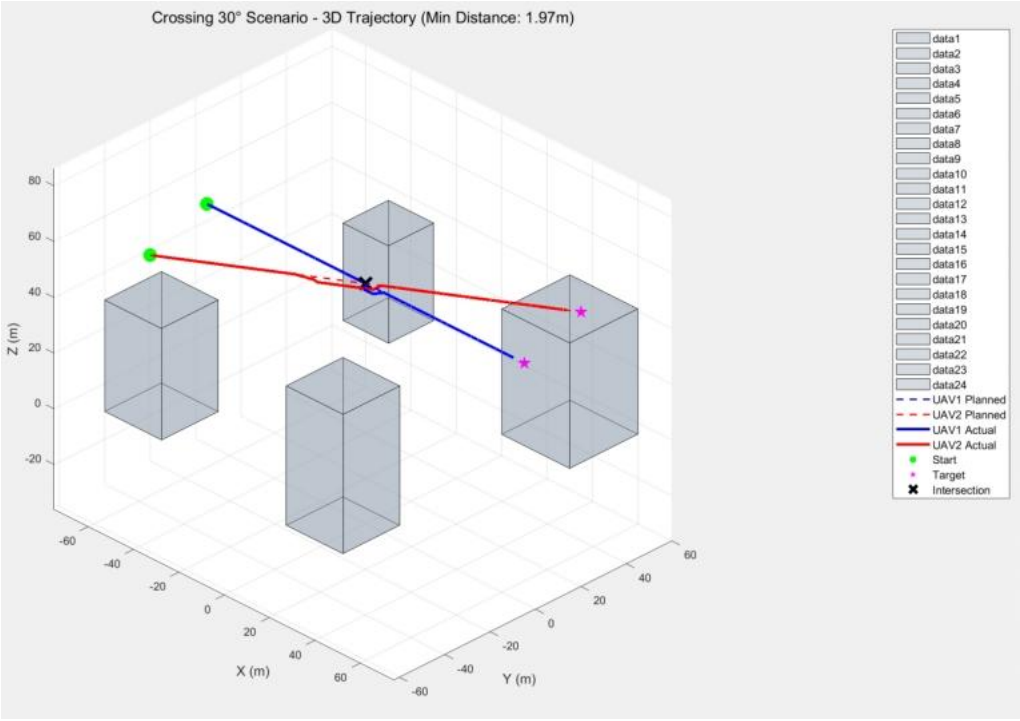


Figure 17. Avoidance after Collision Detection — 3D Trajectory View (30° Intersection)

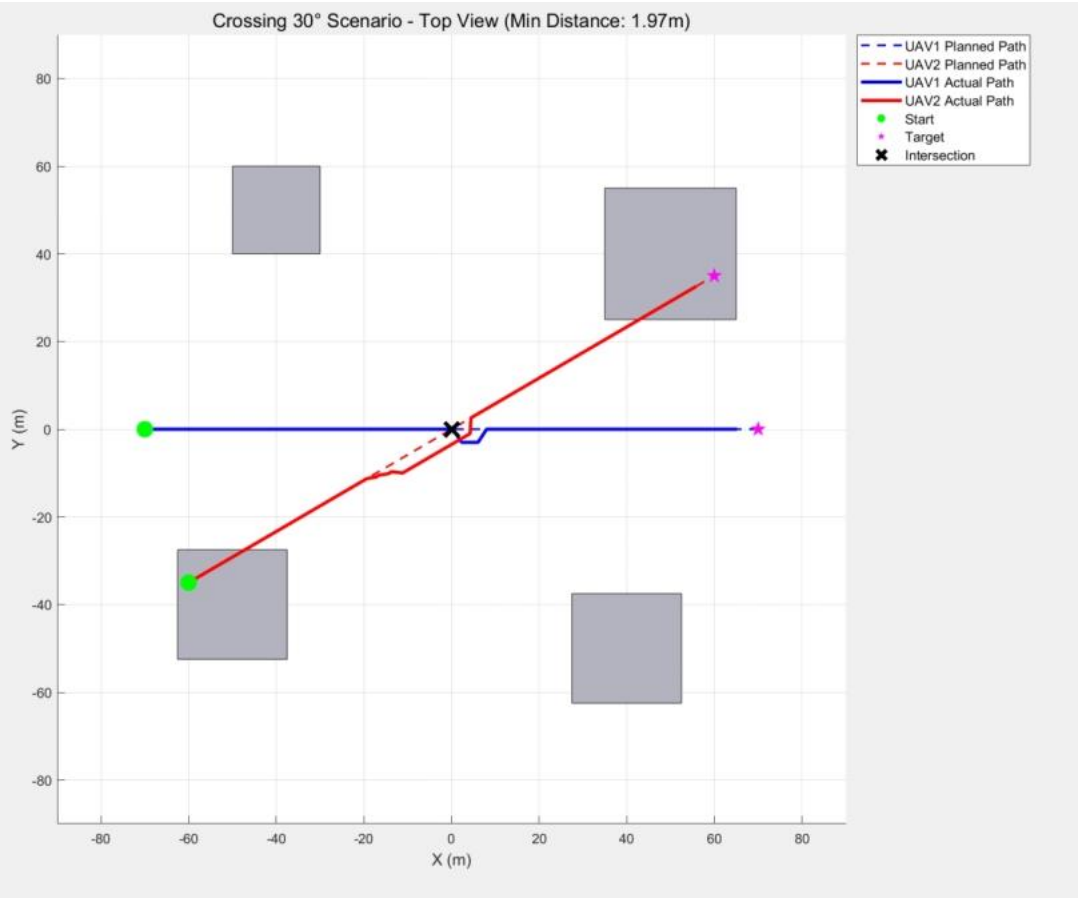


Figure 18. Avoidance after Collision Detection — XY-Plane View (30° Intersection)

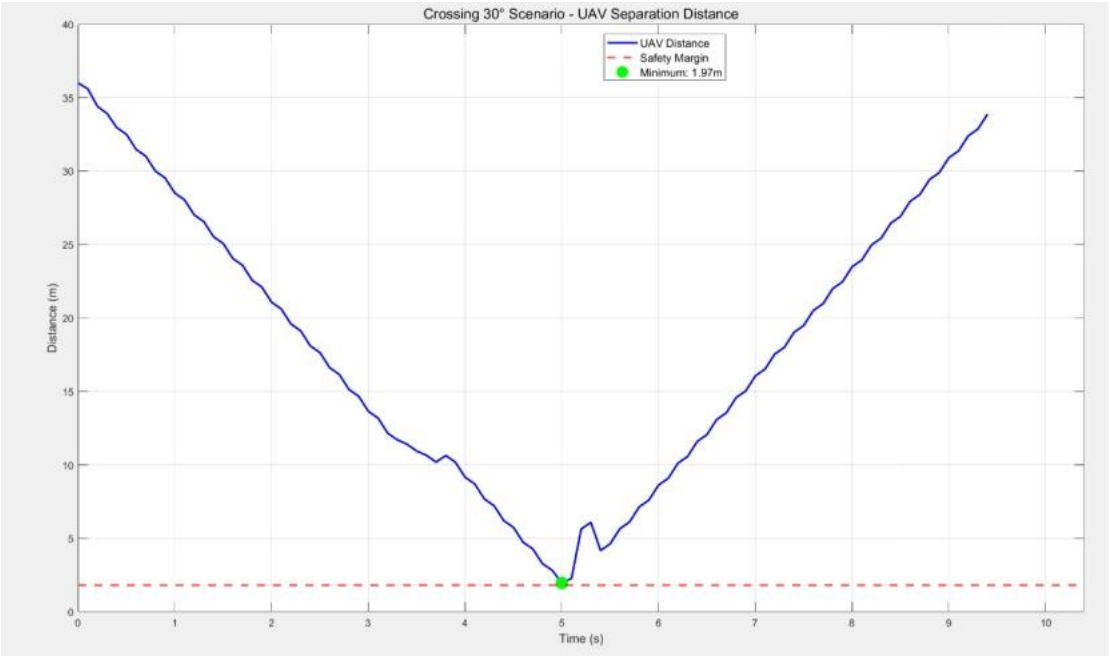


Figure 19. Distance Between Two UAVs vs. Time (30° Intersection)

(3) Perpendicular Crossing (Trajectory Intersection Angle = 90°)

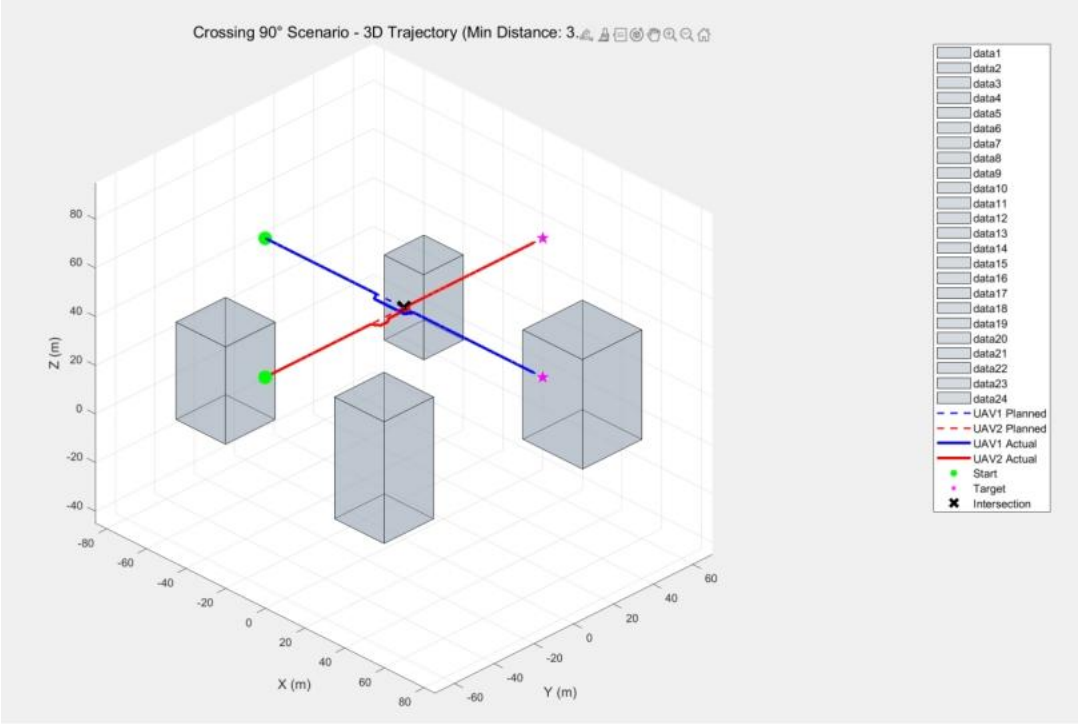


Figure 20. Avoidance after Collision Detection — 3D Trajectory View (90° Intersection)

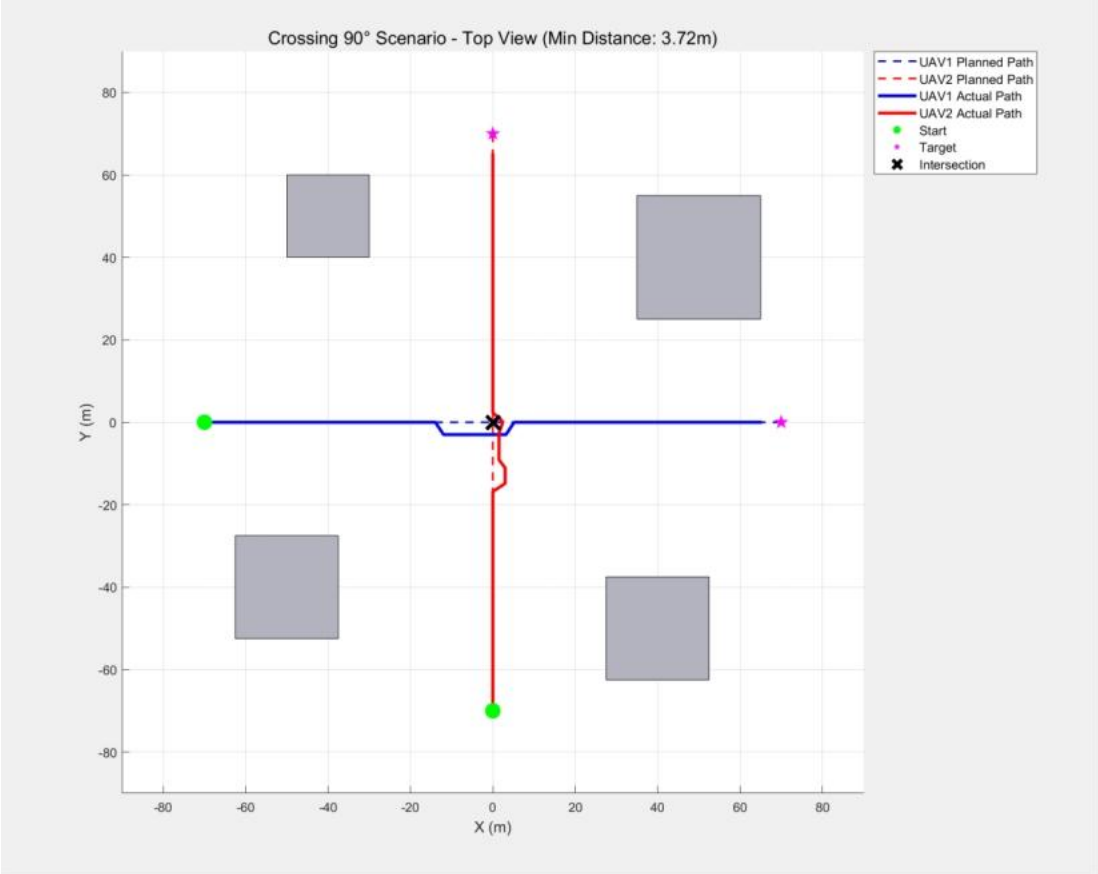


Figure 21. Avoidance after Collision Detection — XY-Plane View (90° Intersection)

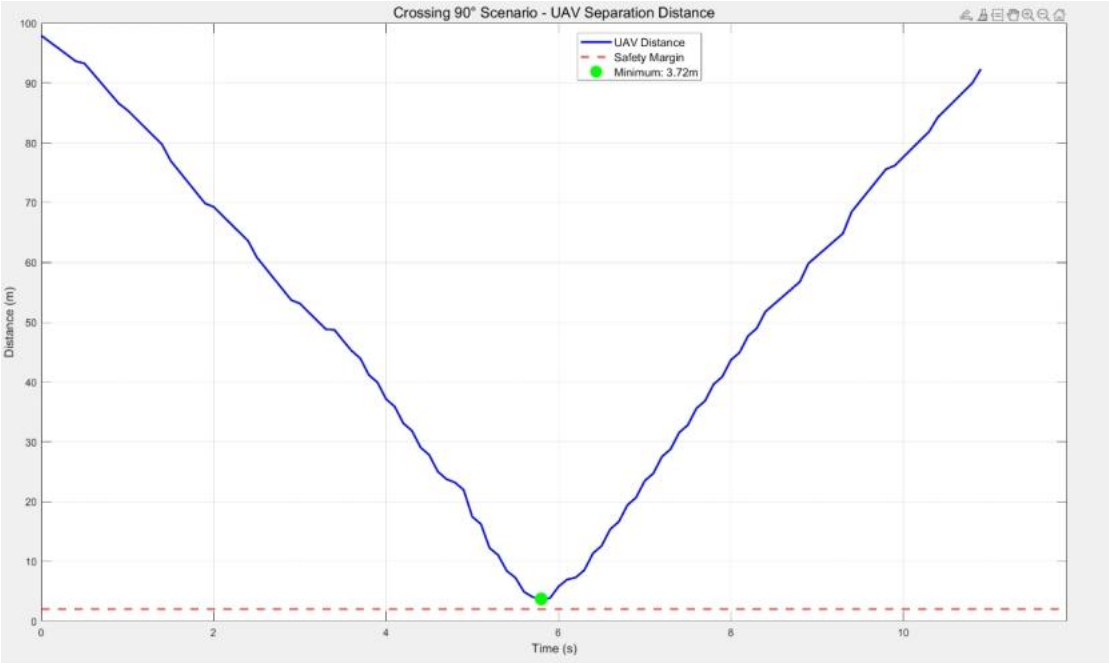
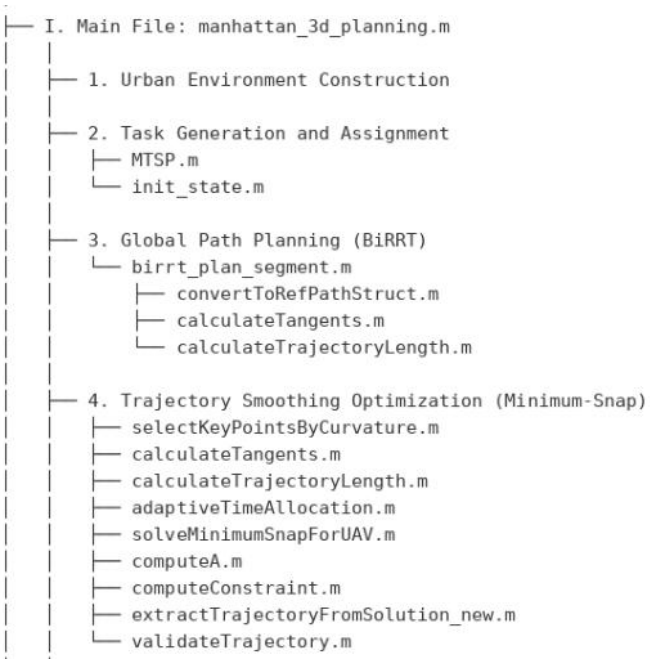


Figure 22. Distance Between Two UAVs vs. Time (90° Intersection)

Conclusion

Under artificially constructed collision scenarios—including head-on, oblique, and perpendicular encounters—the local dynamic obstacle avoidance module successfully detected potential conflicts in advance and generated effective avoidance trajectories. All UAVs maintained the required minimum safety distance throughout the test, and no collisions were observed, validating the robustness and effectiveness of the proposed local obstacle avoidance system.

5. Code File Dependency Diagram


I. Main File: manhattan_3d_planning.m
1. Urban Environment Construction
2. Task Generation and Assignment
MTSP.m
init_state.m
3. Global Path Planning (BiRRT)
birrt_plan_segment.m
convertToRefPathStruct.m
calculateTangents.m
calculateTrajectoryLength.m
4. Trajectory Smoothing Optimization (Minimum-Snap)
selectKeyPointsByCurvature.m
calculateTangents.m
calculateTrajectoryLength.m
adaptiveTimeAllocation.m
solveMinimumSnapForUAV.m
computeA.m
computeConstraint.m
extractTrajectoryFromSolution_new.m
validateTrajectory.m

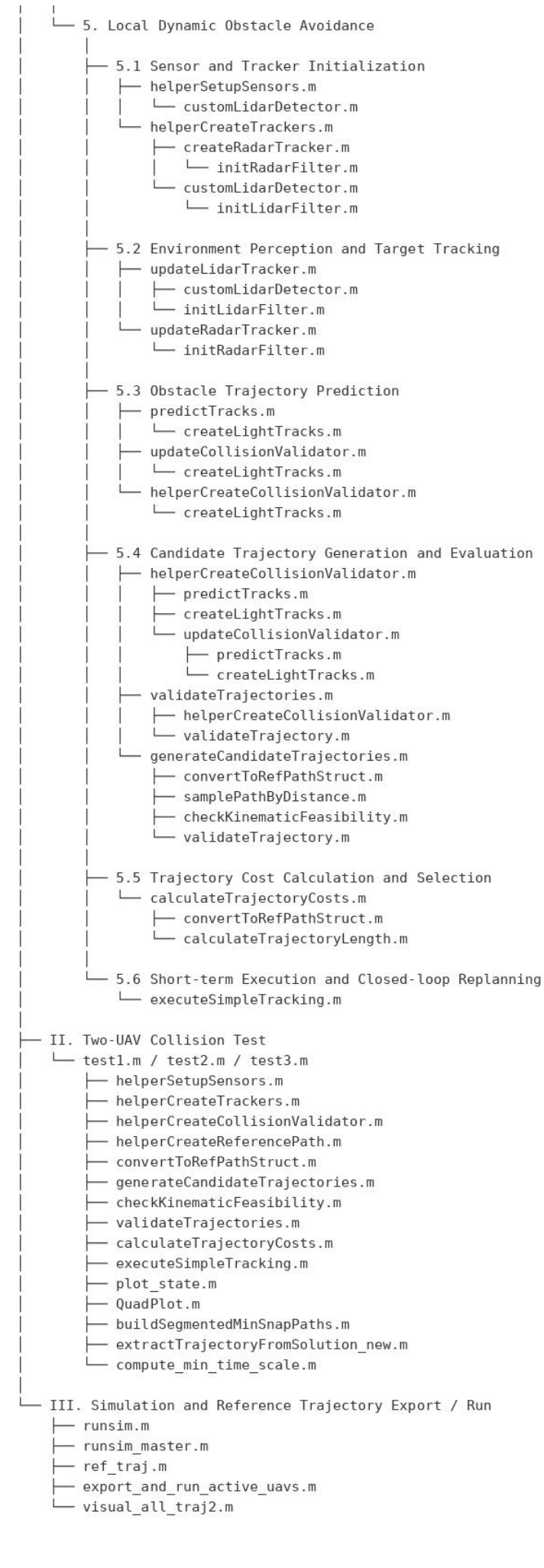

5. Local Dynamic Obstacle Avoidance
5.1 Sensor and Tracker Initialization
helperSetupSensors.m
customLidarDetector.m
helperCreateTrackers.m
createRadarTracker.m
initRadarFilter.m
customLidarDetector.m
initLidarFilter.m
5.2 Environment Perception and Target Tracking
updateLidarTracker.m
customLidarDetector.m
initLidarFilter.m
updateRadarTracker.m
initRadarFilter.m
5.3 Obstacle Trajectory Prediction
predictTracks.m
createLightTracks.m
updateCollisionValidator.m
createLightTracks.m
helperCreateCollisionValidator.m
createLightTracks.m
5.4 Candidate Trajectory Generation and Evaluation
helperCreateCollisionValidator.m
predictTracks.m
createLightTracks.m
updateCollisionValidator.m
predictTracks.m
createLightTracks.m
validateTrajectories.m
helperCreateCollisionValidator.m
validateTrajectory.m
generateCandidateTrajectories.m
convertToRefPathStruct.m
samplePathByDistance.m
checkKinematicFeasibility.m
validateTrajectory.m
5.5 Trajectory Cost Calculation and Selection
calculateTrajectoryCosts.m
convertToRefPathStruct.m
calculateTrajectoryLength.m
5.6 Short-term Execution and Closed-loop Replanning
executeSimpleTracking.m
II. Two-UAV Collision Test
test1.m / test2.m / test3.m
helperSetupSensors.m
helperCreateTrackers.m
helperCreateCollisionValidator.m
helperCreateReferencePath.m
convertToRefPathStruct.m
generateCandidateTrajectories.m
checkKinematicFeasibility.m
validateTrajectories.m
calculateTrajectoryCosts.m
executeSimpleTracking.m
plot_state.m
QuadPlot.m
buildSegmentedMinSnapPaths.m
extractTrajectoryFromSolution_new.m
compute_min_time_scale.m
III. Simulation and Reference Trajectory Export / Run
runsim.m
runsim_master.m
ref_traj.m
export_and_run_active_uavs.m
visual_all_traj2.m

Figure 23. Code File Dependency Diagram

6. References

[1] 2025 MATLAB & Simulink Intelligent Robot Challenge — WeChat Official Account Announcement

[2] The Official GitHub Repository Provided for the 'Multi-UAV Path Planning for Urban Air Mobility' Competition

[3] Ait Saadi, A., et al. (2022). UAV path planning using optimization approaches: A survey. Archives of Computational Methods in Engineering, 29(6), 4233-4284.

[4] Wang, K. P. (2021). UAV trajectory planning method and application research in complex urban spatial environments [Doctoral dissertation, Tsinghua University]. China National Knowledge Infrastructure (CNKI)

[5] Lidar and Radar Fusion in Urban Air Mobility Scenario (UAV) This example implements the perception and trajectory estimation of the three red UAVs below by the ego UAV (the top blue UAV in the simulation).

[6] Motion Planning in Urban Environments Using Dynamic Occupancy Grid Map (Vehicle) Based on a local dynamic occupancy grid map, the local planner performs local obstacle-avoidance replanning at the same frequency as the simulation time step. It specifically employs a sampling-based trajectory planning method in the Frenet frame, then filters and selects the trajectory that is comprehensively optimal in terms of acceleration cost, collision probability cost, and velocity deviation cost, continuously guiding the vehicle towards the curve fitted by the global waypoints.

[7] Object Tracking and Motion Planning Using Frenet Reference Path (Vehicle) In this example, the trajectories and their evolution of other vehicles around the ego vehicle are estimated by a JPDA tracker, which aligns better with real-world scenarios.

[8] Github MTSP-for-Flight-Path-Planning-using-MILP

[9] Github AerialRobotics

[10] Github multi-agent-path-finding

[11] GitHub Desktop Documentation