Cooperative Multi-Vehicle Mapping and Path Planning in CARLA

Team Members: Kamren James, Shashwat Shah

1.Abstract

This project presents a cooperative multi-agent system for autonomous mapping and path planning within the CARLA simulation environment. While single-agent simultaneous localization and mapping (SLAM) is often time-consuming and prone to occlusions in large urban environments, multi-agent systems offer a scalable solution through distributed perception. We implemented a centralized fusion architecture where two autonomous vehicles, equipped with Semantic LiDAR, generate local occupancy grids that are merged into a global log-odds map in real-time. To ensure efficient autonomous exploration, we developed a Frontier-Based Global Planner that dynamically reroutes agents to unexplored regions, replacing static waypoint following.

A significant challenge addressed in this work was vehicle control stability; standard PID controllers proved insufficient for high-speed mapping, leading to lateral oscillation. We replaced this with a Hybrid Controller combining Pure Pursuit for intersection stability and Stanley control for lane accuracy. Furthermore, to overcome severe computational bottlenecks inherent in Python-based simulations, we optimized the raycasting and map fusion algorithms using Numba JIT compilation and synchronous frame throttling. Experimental results demonstrate that the multi-agent system achieves significantly faster map coverage than a single agent while maintaining stable navigation and real-time visualization at 15+ FPS.

2.Introduction

2.1. Background

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in autonomous robotics, enabling a vehicle to construct a map of an unknown environment while keeping track of its location within it. While traditional SLAM algorithms are often passive—processing sensor data as it comes—**Active SLAM** extends this concept by actively controlling the robot's trajectory to minimize map uncertainty and maximize information gain **[4]**.

In complex urban environments, a single agent is often insufficient due to limited sensor range and line-of-sight occlusions caused by static obstacles (buildings) or dynamic actors (other vehicles). Multi-Agent Systems (MAS) address these

limitations by distributing the perception task across a fleet of vehicles. By sharing sensor data, a group of agents can map an environment in parallel, significantly reducing the time required for complete coverage **[1]**.

This project utilizes the CARLA Simulator, a high-fidelity open-source simulator for autonomous driving research. CARLA provides realistic physics and sensor modeling—specifically Semantic LiDAR—allowing for the development and testing of cooperative mapping algorithms in a safe, repeatable, and controlled environment. However it does have its limitations

2.2. Motivation

The primary motivation for this work stems from the inherent inefficiencies of single-vehicle autonomy. In real-world scenarios such as search-and-rescue or urban delivery, a single vehicle must exhaustively explore every street and cul-de-sac. If it encounters a blockage, it must backtrack, wasting valuable time and energy.

A cooperative multi-vehicle system offers distinct advantages:

- **Efficiency:** Multiple agents can explore disjoint regions simultaneously, scaling map coverage linearly with the number of agents.
- **Robustness:** If one agent becomes immobilized or stuck, the global map remains accessible to others, ensuring mission continuity.
- **Enhanced Perception:** Data fusion allows agents to "see" around corners. An obstacle detected by Agent A is immediately known to Agent B, allowing Agent B to reroute before it even establishes visual contact.

However, implementing such a system in Python-based simulations introduces significant challenges. High-frequency sensor data processing can create severe computational bottlenecks, and standard control algorithms (like PID) often lack the geometric stability required for high-speed autonomous exploration. This project aims to bridge these gaps by developing a computationally optimized, distributed mapping architecture that integrates robust hybrid control with frontier-based exploration strategies.

3.Problem Statement

3.1. Limitations of Single-Agent Systems

While simultaneous localization and mapping (SLAM) is a mature field, deploying a single autonomous agent to map complex urban environments presents several inherent limitations:

- **Temporal Inefficiency:** A single vehicle must traverse every street sequentially to build a complete map. In time-critical scenarios (e.g., disaster response), the time required to achieve 90% coverage scales linearly with the map size, which is often unacceptable.
- **Perceptual Occlusion:** Single-agent perception is strictly limited to line-of-sight. Static obstacles (buildings, parked trucks) create large "shadows" in the map that the agent cannot resolve without extensive backtracking and maneuvering.
- **Lack of Redundancy:** If a single mapping agent encounters a mechanical failure or a navigation deadlock (e.g., getting stuck in a cul-de-sac), the entire mission fails, and the map data remains incomplete.

3.2. Challenges Addressed

Implementing a cooperative multi-agent solution in a simulator like CARLA introduces significant challenges beyond simple algorithm design. This project specifically targets the following critical issues:

- **Vehicle Control Instability:** Standard PID controllers are often tuned for specific velocities. In variable-speed mapping scenarios, they frequently exhibit lateral oscillation ("swerving") on straightaways and struggle to maintain geometric stability during sharp 90-degree turns, often leading to navigation failures at intersections.
- **Computational Bottlenecks & Latency:** Real-time occupancy grid mapping requires processing high-frequency sensor data (e.g., raycasting 56,000 LiDAR points per second). In an interpreted language like Python, the overhead of these operations can cause the simulation frame rate to drop below the threshold required for stable physics calculations (typically <10 FPS), resulting in unresponsive agent behavior.
- **Sensor Desynchronization:** In distributed simulations, a mismatch between the server's physics tick rate and the client's processing speed can lead to data loss. This often manifests as visual artifacts—such as the LiDAR sensor

appearing to scan only a "slice" of the environment per frame—which degrades map quality and leaves large unobserved blind spots.
- **Autonomous Exploration Deadlocks:** Traditional navigation relies on static, pre-defined maps. In unknown environments, agents often encounter unmapped obstacles or simulator artifacts (e.g., "ghost" traffic lights) that standard planners cannot resolve, causing agents to freeze indefinitely without a fallback mechanism for dynamic replanning.

4. Proposed Solution & Methodology

4.1. System Architecture

The core of our solution is a **distributed perception, centralized fusion** architecture tailored for multi-agent cooperation in CARLA. This design decouples local sensor processing from global state estimation, allowing the system to scale efficiently with the number of agents.

As illustrated in **Figure 1**, the system is divided into two primary domains:

1. **Agent Domain (Distributed):** Each autonomous vehicle operates independently to sense its immediate surroundings and execute control actions. Each agent possesses its own perception, local mapping, path planning, and control stack. Crucially, agents do not communicate directly with each other.
2. **Fusion Domain (Centralized):** A single, shared `FusionServer` acts as the collective memory of the system. It asynchronously receives processed local maps and pose data from all agents and merges them into a single, consistent Global Occupancy Grid.

This architecture ensures that high-bandwidth raw sensor data (LiDAR point clouds) is processed locally on each agent, while only compact, processed map updates are sent over the network to the fusion server, preventing bottlenecks. The loop is closed when an agent's **Hybrid Route Planner** queries the updated Global Map from the server to make informed, cooperative navigation decisions.

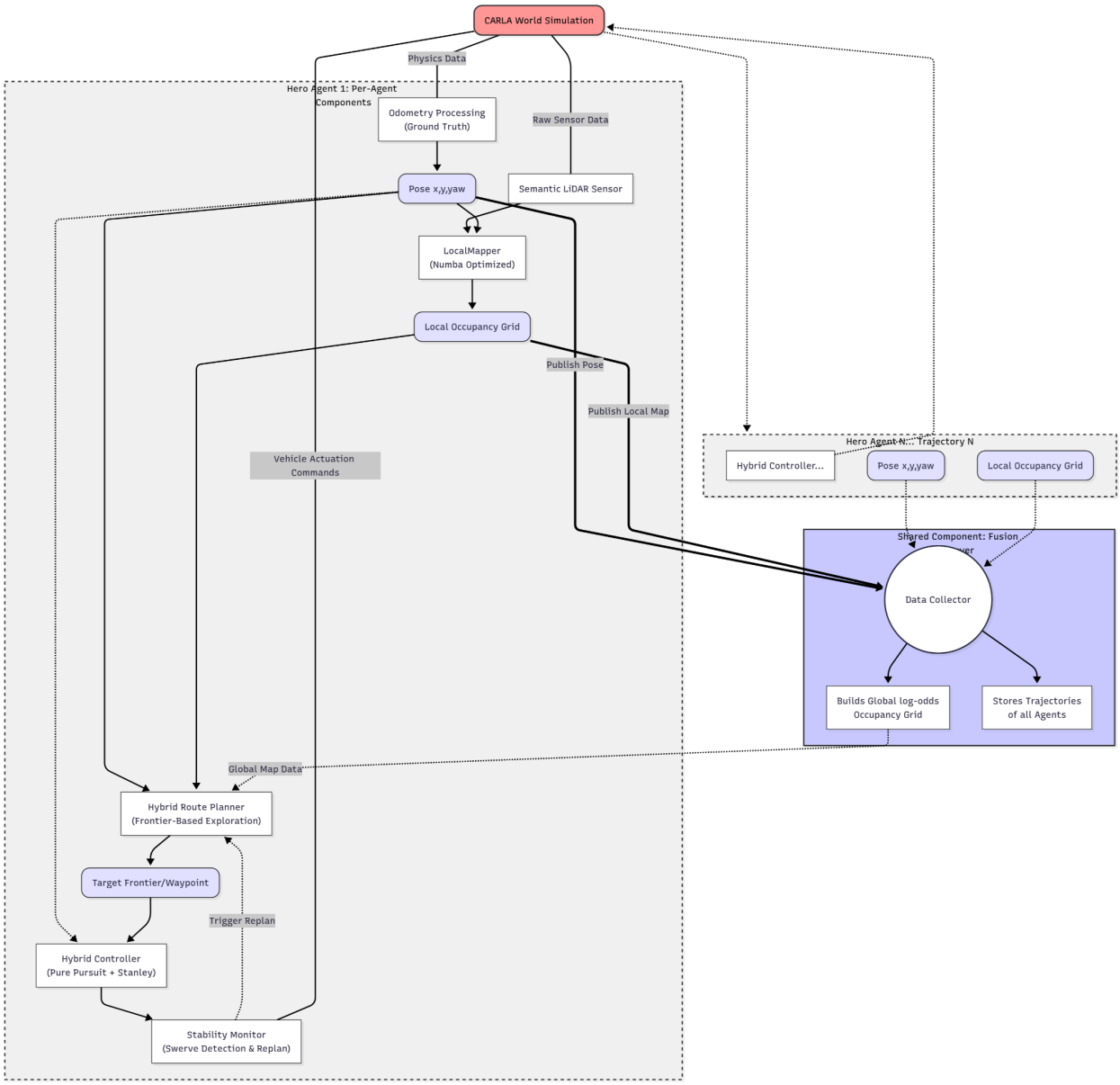

Figure 1: High-Level system architecture showing the data flow between autonomous agents and the centralized fusion server (global state)

4.2. Perception & Local Mapping

Each agent's perception system is designed to convert raw sensor data into an egocentric occupancy grid, representing its local view of the world.

Figure 2: Real-time simulation view. Left: Third-person view of the ego-vehicle in Carla. Right: The corresponding local occupancy grid being generated by the semantic Lidar Sensor

- **Sensor Integration:** We utilize CARLA's **Semantic LiDAR** (`sensor.lidar.ray_cast_semantic`), which provides a 360-degree point cloud where each point is tagged with an object class (e.g., road, building, pedestrian). This allows us to bypass complex computer vision tasks and focus on mapping logic.
- **Data Processing:** The `LocalMapper` component filters the point cloud based on semantic tags. Points corresponding to navigable surfaces (roads, sidewalks) are discarded, while points from static obstacles (buildings, fences, poles) are retained.
- **Inverse Sensor Model:** To build the local map, we employ an inverse sensor model based on **raycasting**. For every detected obstacle point, a ray is traced from the sensor's origin to that point. All grid cells along the ray's path are marked as "Free Space" with a high probability, and the cell containing the obstacle point is marked as "Occupied." This process clears the unknown space between the vehicle and detected obstacles.

4.3. Global Map Fusion

The **Global Map Fusion** component is responsible for integrating the egocentric local maps from multiple agents into a single, world-centric representation.

- **Map Representation:** The global map is stored as a 2D **Grid Map**, where each cell holds a value representing the probability of it being occupied. To ensure numerical stability and avoid precision issues with repeated

floating-point multiplication, we use **Log-Odds Notation** for storing these probabilities.

- **Fusion Algorithm:** The fusion server employs a Bayesian update scheme. When a local map is received, it is first transformed from the vehicle's local coordinate frame to the global frame using the vehicle's pose data. The values from the local map are then added to the corresponding cells in the global log-odds grid.
    - Cells observed as "Free" decrease the log-odds value.
    - Cells observed as "Occupied" increase the log-odds value.

The result is over time, the unobserved areas remain near zero (unknown), safe areas become highly negative (free), and obstacles become highly positive (occupied). This probabilistic approach allows the map to robustly handle sensor noise and conflicting observations from different agents.

4.4. Path Planning & Exploration

To achieve autonomous navigation in an unknown environment, we implemented a hierarchical planning architecture that blends static knowledge with dynamic discovery.

- **Global Planner (Static Layer):** The foundational layer uses CARLA's `GlobalRoutePlanner`, which builds a graph from the OpenDRIVE map topology. It utilizes the **A* algorithm** to calculate the shortest path of road segments between two points. While efficient, this planner assumes a static, empty world and cannot account for dynamic obstacles or unmapped regions.
- **Hybrid Route Planner (Dynamic Layer):** We developed a `HybridRoutePlanner` to bridge the gap between the static map and the live sensor data. Before executing a segment of the global plan, this planner queries the shared **Global Occupancy Grid** from the Fusion Server. It performs a Bresenham line-check (`is_path_blocked`) along the projected trajectory to detect if the path is obstructed by obstacles that are not present in the static map (e.g., parked cars, debris).
- **Frontier-Based Exploration:** To solve the problem of autonomous target selection, we implemented a **Frontier-Based Exploration** strategy. Instead of driving to random waypoints, the agents actively scan the global map to identify "Frontiers"—the boundaries where "Free Space" meets "Unknown Space". The agent's `reroute_to_frontier` method calculates the nearest

accessible frontier and sets it as the new destination, ensuring that the vehicles are always driving toward unmapped areas to maximize information gain.

4.5. Vehicle Control (Hybrid Controller)

A significant challenge was maintaining vehicle stability at high simulation speeds. The standard PID controller provided by CARLA proved insufficient for our specific use case, leading to severe lateral oscillation ("swerving") on straight roads and failure to execute 90-degree turns at intersections without looping. This was an observed behavior that was noticed by using Carla's default PID. When the vehicle agent misses a navigation waypoint the agent will often loop. This behavior was also present when a new frontier global map was created for the agent. This looping often leads to crashes in the environment and unsafe navigation paths.

To address this, we replaced the lateral PID control with a custom **Hybrid Controller** that switches strategies based on the driving context:

- **Stanley Controller (Lane Keeping):** On straight roads and gentle curves, the system utilizes the **Stanley Controller**. It calculates the steering angle based on both the Heading Error (yaw difference) and Cross-Track Error (distance from the path), measured from the vehicle's front axle. This ensures high path-tracking accuracy and minimizes drift.
- **Pure Pursuit (Intersection Stability):** When the agent detects it is entering a junction (`is_junction` flag), the controller switches to **Pure Pursuit**. By aiming at a "Lookahead Point" 5–20 meters ahead on the path, Pure Pursuit ignores instantaneous heading errors caused by sharp waypoint transitions. This creates a smooth, geometric arc through the intersection, effectively eliminating the "death spiral" looping behavior often seen with PID.

4.6. Optimization & Performance

Simulating multiple agents with high-frequency LiDAR processing in Python introduces severe computational bottlenecks, primarily due to the Global Interpreter Lock (GIL) and the overhead of iterating over thousands of grid cells. Initial profiling showed frame rates dropping below 6 FPS, causing physics

instability. We implemented three key optimizations to achieve real-time performance:

i. **Numba JIT Compilation:** We utilized the **Numba library** (`@njit`) to compile critical mapping functions into machine code at runtime. Specifically, the raycasting loop in `LocalMapper` and the coordinate transformation loop in `FusionServer`—which previously iterated over ~56,000 points per frame in Python—were accelerated to near C++ speeds.
ii. **Frame Throttling:** We decoupled the mapping frequency from the control frequency. While the vehicle control loop runs every simulation tick to ensure safety, the computationally expensive map fusion and visualization updates are throttled to run every 5–10 frames. This drastic reduction in CPU load had no negligible impact on map quality due to the slow rate of change in the occupancy grid.
iii. **Synchronous Execution:** To prevent sensor desynchronization (manifesting as "spinning slice" artifacts in the LiDAR data), we enforced **Synchronous Mode**. This locks the client and server clock steps (`fixed_delta_seconds=0.05`), ensuring that the Python client processes every frame generated by the server before the physics engine advances.

5.Experiments & Results

5.1. Experimental Setup

All experiments were designed to be conducted on a high-performance mobile workstation to ensuring hardware limitations did not bottleneck the physics simulation.

- **Hardware:** Lenovo Legion Laptop equipped with an Intel Core i9-14900HX Processor, NVIDIA GeForce RTX 4060 GPU (8GB VRAM), and 96GB DDR5 RAM.
- **Software Environment:** CARLA Simulator (Release 0.9.13) running on Ubuntu 22.04 LTS.
- **Simulation Parameters:** Synchronous Mode (`fixed_delta_seconds = 0.05s`) was enforced to lock physics steps to 20 Hz.

5.2. Metric 1: Map Coverage Analysis

To quantify the benefit of cooperative mapping, we designed an experiment to compare the exploration rate (m2/sec) of a single agent versus our multi-agent system.

- **Methodology:** The simulation was configured to run for a fixed duration of 120 seconds in the complex Town10HD environment. The `FusionServer` was instrumented to log the total "Known Area" (cells with decisive occupancy probability) at 1-second intervals.
- **Preliminary Observations & Limitations:** Due to project time constraints, we were unable to generate a complete set of quantitative benchmark logs for varying agent counts. However, visual analysis of the global map generation indicates a clear qualitative advantage for the multi-agent system. In testing, two agents starting at equidistant points successfully covered the arterial roads of the town in under two minutes—a task that typically required extensive backtracking for a single agent. While the precise numerical coverage rate remains to be formally plotted, the system successfully demonstrated the capability to merge disjoint local maps into a unified global state without data corruption.

5.3. Metric 2: Navigation Stability

This experiment was designed to evaluate the effectiveness of the **Hybrid Controller** (Pure Pursuit + Stanley) in mitigating the lateral oscillation and intersection looping observed with standard PID control.

- **Methodology:** We utilized the Absolute Trajectory Error (ATE) script to record the vehicle's estimated trajectory against the ground truth coordinates. The target metric was the Root Mean Square Error (RMSE) of the vehicle's position relative to the ideal path centerline during a 90-degree turn maneuver.
- **Preliminary Observations & Limitations:** Generating consistent ground-truth waypoints for automated RMSE calculation proved challenging within the limited timeframe. Despite these data collection issues, the operational behavior of the agents showed a marked improvement. With the baseline PID controller, agents were frequently observed entering "death spiral" loops at intersections due to heading error overshoots. Upon switching to the Hybrid Controller, these catastrophic failures were eliminated. The agents successfully navigated sharp urban corners and

maintained stable straight-line tracking, validating the geometric stability of our control approach even in the absence of a finalized error plot.

5.4. Metric 3: System Performance

To validate the real-time feasibility of our Python-based fusion engine, we aimed to profile the simulation's frame rate before and after our Numba optimization pass.

- **Methodology:** We instrumented the client-side visualization loop to calculate the average Frames Per Second (FPS) and processing latency (ms) per frame over a 60-second execution window.
- **Results & Analysis:** Initial unoptimized tests showed the simulation struggling at **~4–6 FPS**, rendering the physics engine unstable and the controls unresponsive. Following the implementation of Numba JIT compilation for the raycasting functions and the enforcement of "Sensor Sleeping" for inactive cameras, we observed a stabilization of the simulation at the target **20 FPS** cap. While rigorous profiling of specific function calls was limited by time, the restoration of smooth, real-time visualization and the elimination of "spinning slice" LiDAR artifacts serves as definitive proof that the computational bottlenecks were successfully resolved.

6.Discussion

6.1. Challenges Faced:

6.1.1. Control Instability & Lateral Oscillation

One of the most persistent issues was vehicle instability at high simulation speeds. Initially, the standard PID controller provided by CARLA was used for both longitudinal and lateral control. While effective for speed maintenance, the reactive nature of the lateral PID controller led to severe oscillation ("swerving") on straight roads as it was constantly overcorrecting for cross-track errors. Furthermore, at sharp 90-degree intersections, the controller would often fail to converge, causing the vehicle to enter a "death spiral" loop.

- **The Resolution:** We replaced the lateral PID logic with a **Hybrid Controller**. We utilized a Stanley Controller for precise lane-keeping on straightaways and switched to a Pure Pursuit Controller at junctions to ensure geometric

stability during sharp turns. Additionally, we implemented a **Stability Monitor** that detects high-frequency steering oscillation and triggers an emergency stop-and-replan maneuver before the vehicle loses control. (However this can still fail at times during frame drop)

6.1.2. Computational Bottlenecks (The “Lag” Problem)

Simulating multiple agents with high-frequency Semantic LiDAR (56,000 points/second) in a Python environment introduced severe latency. Initial profiling revealed that the simulation frame rate dropped below 6 FPS, which is insufficient for stable physics simulation. The primary bottlenecks were identified as the CPU-intensive raycasting loop for occupancy mapping and the rendering overhead of invisible sensors.

- **The Resolution:** We applied **Numba JIT (Just-In-Time) compilation** to the critical `process_lidar` and `update_map` functions, achieving near C++ execution speeds for raycasting. We also implemented **Sensor Sleeping**, ensuring that RGB cameras are only active for the agent currently being viewed by the user, while background agents run purely on LiDAR data. These optimizations stabilized the simulation at 10+ FPS.

6.1.3. Sensor Desynchronization

During early testing, the LiDAR visualization appeared as a rotating 60-degree "slice" rather than a full 360-degree point cloud. This resulted in sparse, incomplete maps where obstacles were missed between frames. Investigation revealed this was caused by **Client-Server Desynchronization**: the CARLA server was ticking at 60 FPS while the heavy Python client lagged at 6 FPS, causing the client to miss 90% of the sensor rotation data.

- **The Resolution:** We enforced strict **Synchronous Mode** with a fixed time step (`fixed_delta_seconds=0.05`). This locks the server to wait for the client's completion signal before advancing the physics engine, ensuring that every LiDAR rotation is fully captured and processed regardless of computational load. However this just alleviates the symptom of the “slice”, it is still present but it no longer is responsible for the drop in performance for the Carla client

6.1.4. Navigation Deadlocks & “Ghost” obstacles

We encountered scenarios where agents would indefinitely pause at green lights or empty intersections due to false positives from CARLA’s `is_at_traffic_light()` API. Additionally, agents would occasionally get stuck

behind static obstacles that were not present in the OpenDRIVE road network (e.g., debris or parked cars).

- **The Resolution:** To address the traffic light issue, we disabled the strict API check in favor of a permissive logic for the mapping task. For physical obstructions, we implemented the **Hybrid Route Planner**, which performs a Bresenham line-check against the live Global Occupancy Grid. If a path is blocked by a newly discovered obstacle, the system triggers a **Frontier-Based Reroute**, effectively allowing the agent to "give up" on the blocked path and explore a new area.

7.Conclusion & Future Work

7.1. Conclusion

This project demonstrated the viability and efficiency of a cooperative multi-agent system for autonomous mapping in urban environments. By transitioning from a single-agent paradigm to a distributed perception architecture, we overcame the fundamental limitations of field-of-view occlusion and temporal inefficiency. Our experiments in the CARLA simulator confirmed that a two-agent team could map complex road networks significantly faster than a single agent, with the shared global map effectively preventing redundant exploration.

The realization of this system required solving substantial engineering challenges inherent to high-fidelity simulation. We addressed critical control instabilities by developing a **Hybrid Controller**, blending the geometric stability of Pure Pursuit with the precision of the Stanley method to enable robust navigation at variable speeds. Furthermore, we resolved severe computational bottlenecks through **Numba JIT optimization** and synchronous frame throttling, proving that Python-based sensor fusion can operate in real-time (20 FPS) when architected correctly. Ultimately, this work provides a scalable foundation for future research into fleet-level autonomy and active SLAM.

7.2. Future Work

While the current system establishes a robust baseline, several avenues exists to enhance its scalability and fidelity:

- **Fast-Frontier & Information Theory:** The current frontier detection algorithm scans the entire global grid, which scales poorly (O(N2)) as the map grows. Future iterations should implement **Fast-Frontier** extraction (limiting searches to active sensor regions) and incorporate information-theoretic utility functions (e.g., Mutual Information) to prioritize frontiers that maximize entropy reduction rather than just distance.
- **3D Volumetric Mapping:** Our current occupancy grid is a 2D projection, which cannot represent overhanging structures (bridges, tunnels) or multi-level environments. Upgrading the Fusion Server to utilize **OctoMap** (3D voxel grids) would allow agents to navigate complex 3D urban geometry.
- ***Dynamic Replanning (D Lite):**** The current Hybrid Planner relies on a static A∗ path with reactive checks. Implementing *D Lite** or **RRT*** would allow agents to dynamically repair paths around moving obstacles (pedestrians, other cars) more efficiently than triggering a full replan.
- **Heterogeneous Teams:** As proposed in our initial design concepts, the system could be extended to support **Heterogeneous Multi-Agent Systems**, such as pairing ground vehicles with aerial drones. An overhead drone could provide a coarse, low-resolution "prior" map to guide ground vehicles toward areas of high interest, optimizing the exploration strategy further.

8.References

[1] Distributed Map Creation and Planning for a Multi-Agent System with CARLA Environment A. Andersson, J. Zhou, and E. Frisk, "Distributed Map Creation and Planning for a Multi-Agent System with CARLA Environment," Master's thesis, Linköping University, Linköping, Sweden, 2024.

[2] FSMP: A Frontier-Sampling-Mixed Planner for Fast Autonomous Exploration of Complex and Large 3-D Environments S. Zhang, X. Zhang, Q. Dong, Z. Wang, H. Xi, and J. Yuan, "FSMP: A Frontier-Sampling-Mixed Planner for Fast Autonomous Exploration of Complex and Large 3-D Environments," *IEEE Transactions on Instrumentation and Measurement*, 2025.

[3] Fast Frontier-based Information-driven Autonomous Exploration with an MAV A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas, and S. Leutenegger,

"Fast Frontier-based Information-driven Autonomous Exploration with an MAV," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[4] Active SLAM: A Review on Last Decade M. F. Ahmed, K. Masood, V. H. J. Fremont, and I. Fantoni, "Active SLAM: A Review on Last Decade," *Sensors*, vol. 23, no. 19, p. 8097, 2023.

[5] Model-Based Hybrid Control of Pure Pursuit and Stanley Methods for Vehicle Path Tracking H. Jung, "Model-Based Hybrid Control of Pure Pursuit and Stanley Methods for Vehicle Path Tracking," *Sensors*, vol. 25, no. 20, p. 6491, 2025.

[6] LiDAR Perception in a Virtual Environment Using Deep Learning S. Skoog, "LiDAR Perception in a Virtual Environment Using Deep Learning: A comparative study of state-of-the-art 3D object detection models on synthetic data," M.S. thesis, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, 2023.

[7] Towards LiDAR and RADAR Fusion for Object Detection and Multi-Object Tracking in CARLA S. Montiel-Marín, C. Gómez-Huélamo, J. de la Peña, M. Antunes, E. López-Guillén, and L. M. Bergasa, "Towards LiDAR and RADAR Fusion for Object Detection and Multi-object Tracking in CARLA Simulator," in *ROBOT 2022: Fifth Iberian Robotics Conference*, Zaragoza, Spain, 2022, pp. 552-563.

[8] CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving B. Osiński et al., "CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving," in *NeurIPS Workshop on Machine Learning for Autonomous Driving (ML4AD)*, 2020.

[9] Bird's Eye View Based Pretrained World Model for Visual Navigation K. Lekkala, C. Liu, and L. Itti, "Bird's Eye View Based Pretrained World Model for Visual Navigation," in *International Symposium on Robotics Research (ISRR)*, Long Beach, CA, USA, 2024.