Towards Modernization of Long-Range Image-Space Planning for Off-Road Navigation

Shubhra Aich

CMU-RI-TR-25-96

November 26, 2025

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Wenshan Wang, *Chair*
Sebastian Scherer, *Co-Chair*
Maxim Likhachev
Samuel Triest, *RI PhD*

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To Fernando de La Torre (Faculty, CMU RI)

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

Abstract

This thesis revisits long-range, image-space planning for off-road navigation and modernizes the classical first-person view (FPV) paradigm by building upon recent advances in perception. It introduces a lightweight depth calibration scheme, analytic configuration-space (C-space) transforms, interpretable frontier selection, and a pixel-space A* planner with validated heuristic soundness. Concretely, we (i) make monocular depth metrically usable at test time via an affine, log-domain calibration with sparse LiDAR; (ii) derive and implement a depth-aware FPV C-space inflation that projects vehicle width/length analytically and realizes it with separable row/column sliding-maximum filters augmented by per-pixel depth consistency checks; (iii) propose transparent, angular-sector frontiering that reasons jointly about traversability cost and minimal lethal depth, alongside goal-aware revalidation; and (iv) preserve A* admissibility/consistency in image space through a simple cost renormalization that avoids silent suboptimality in low-cost free space.

We evaluate the resulting, modular sub-stack in the high-fidelity Falcon simulator [1] under a shared ROS graph. Using a common perception front-end and planning back-end, we compare three frontiering strategies – (1) a LAGR-style row-wise baseline, (2) an LRN-inspired openness heuristic adapted to operate with explicit depth and cost, and (3) an Angular Cost & Depth (ACD) variant that couples average cost with a minimum lethal-depth constraint. Across diverse courses (e.g., farm, desert, mixed terrain), the calibrated monocular depth reduces error versus raw monocular predictions, and both LRN-inspired and ACD frontiering tend to outperform the purely row-wise baseline on longer traverses. We view these as encouraging indications rather than definitive claims: all results are in simulation, with performance subject to perception quality, calibration coverage, and environment diversity.

Scope and limitations are explicit. The work was conducted over a short project window (May 2025–present) and prioritized stabilizing the proposed sub-stack in conjunction with a core FieldAI [2] stack through the high-fidelity Falcon simulator [2] and ROS1 into a reliable, end-to-end testing framework. No real-world deployments were performed. Nevertheless, the design is intentionally modular and auditable to make it relatively straightforward to integrate the sub-stack into existing off-road autonomy stacks lacking explicit long-range planning. Such integration might enable better autonomy by offering a practical bridge between classical image-space efficiency and metric-world robustness.

Acknowledgments

I gratefully acknowledge my advisors at the Robotics Institute, Carnegie Mellon University, for providing full financial support throughout my time in the program. The foundational ideas behind this thesis were deeply influenced by Prof. Maxim Likhachev's Planning course, which laid the conceptual groundwork for much of the research presented here.

I am indebted to the exceptional members of the AirLab off-road autonomy team – Samuel Triest, Matthew Sivaprakasam, Mateo Gauman Castro, Parv Archana Maheshwari, Micah Nye, Cherie Ho, and Yifei Liu. This thesis truly stands on the shoulders of these off-road giants. In particular, I owe special thanks to Samuel Triest – the frontiering and planning ideas explored here originated from our early one-on-one discussions at the start of the project. I am equally grateful to Matthew Sivaprakasam, whose costmap prediction model enabled me to focus on the core aspects of this work.

I would like to express my sincere appreciation to Dr. Shehryar Khattak and Adrian Gomez at Field AI, and to Syed Muhammad Maaz and David Pepley at Duality, for their guidance and collaboration. Their technical insights and assistance in designing the stack using the Falcon simulator were instrumental throughout this research.

Beyond the technical scope, I am profoundly grateful to Prof. Fernando de la Torre for bringing me to Carnegie Mellon University. Although our research directions diverged over time, his mentorship and timely advice have guided me through many difficult junctures. I owe much of my academic and professional growth, including the best opportunities I have received to date, to his support and trust.

I would also like to thank Alvaro Fernandez Garcia for helping me get started during my early days at CMU and for being a reliable friend ever since. My sincere appreciation goes to Vishnu Mani Hema, whose calm composure and thoughtful suggestions have helped me navigate several challenging situations.

Special thanks are due to Hadley Pratt, one of the kindest and most patient individuals at the Robotics Institute, for her tremendous help in dealing with administrative matters during my first year. On a similar note, I would like to acknowledge Ashley Hong for her prompt and efficient assistance, which made many logistical challenges much smoother.

My gratitude also extends to Janice Phillips and Devin Frank at the RI SQH office – their clarity, responsiveness, and genuine care are a quiet but essential force behind the success of countless RI students.

Last but not least, I was also fortunate enough to serve as a TA for Prof. William "Red" Whittaker, often regarded as the godfather of field robotics. Observing his lectures and hearing his reflections from decades of pioneering experience was a rare privilege that fundamentally reshaped my thought process. I believe those lessons will come full circle one day – perhaps when I will find myself competing against other CMU teams, armed with the very instincts he helped instill.

Funding

This work was funded by Field AI [2], whose support made this research possible. Access to the high-fidelity Falcon [1] simulator was instrumental in developing and validating the proposed long-range navigation stack.

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Autonomous navigation in unstructured, off-road environments remains a demanding problem at the intersection of perception, planning, and control. Long-range decision making is especially challenging when the robot must reason beyond the near field. In these far-field regions, sparse geometry and viewpoint changes reduce the reliability of map-centric pipelines.

A complementary line of work, image-space planning [11, 20], seeks to plan directly on a fixed-size first-person view (FPV), pixel-grid derived from the camera view. By keeping the planning domain constant in size (i.e. a fixed-resolution FPV pixel grid regardless of the physical extent of the world), and tightly coupled to perceptual measurements-based direct predictions without hallucination, such methods can be efficient and responsive at far range. However, they require careful handling of geometry, footprint inflation, and frontier selection to be dependable in practice. Thus, we adopt this perspective and modernize it with lightweight depth calibration, analytic FPV configuration-space (C-space) transforms, interpretable frontiering, and a validation of pixel-space A^* heuristic soundness via a simple cost renormalization that preserves admissibility and consistency. The objective is a practical, modular navigation sub-system (henceforth, the FPV sub-stack) for long-range, off-road navigation.

Motivation and Perspective

This work aims to describe every component from the first principles, tracing the full reasoning chain from perception to long-range frontiering and planning, except for the use of off-the-shelf deep learning models in the perception stage. In spirit, the thesis represents a rejuvenation of the image-space planning paradigm first explored in the DARPA LAGR [11, 12] program, now reinterpreted through the lens of modern advances in perception algorithms and compute capabilities. Whereas early systems faced limited resolution, sparse computation, and rigid geometric models, current platforms benefit from high-resolution cameras, learned depth priors, and real-time dense computation on inexpensive hardware. The nomenclature of this thesis reflects this continuity – a modern take on long-range navigation grounded in image-space reasoning, but freed from the heavy constraints that once limited its reach.

Problem Setting and Goals

We consider a robot operating off road with a forward-facing RGB camera, access to sparse LiDAR, and a high-level waypoint or goal. Our objective is to choose long-range image-space frontiers and generate FPV plans that respect traversability while remaining simple enough for real-time use. The design emphasizes (i) modularity: each block can be replaced without retraining the rest of the stack; and (ii) interpretability: frontier choices and footprint inflation are explicit and auditable. We focus on consistency with established planning guarantees wherever possible (e.g., heuristic soundness for A* after cost renormalization) while keeping the overall approach lightweight.

Throughout the thesis, we use a few recurring system-level properties. By *efficient*, we mean that computation scales primarily with the fixed FPV image size (rather than world size) and that all core modules – depth calibration, C-space inflation, frontiering, and planning can run in real time in Falcon. By *modular*, we mean that perception, configuration-space transformation, frontier selection, and planning are encapsulated as ROS nodes with clear interfaces, so that any one component can be swapped or upgraded without retraining or rederiving the others. By *interpretable* or *auditable*, we mean that the decisions of the system (e.g., sector statistics, frontier selection, heuristic values) are expressed in terms of explicit geometric quantities and

scalar summaries that can be inspected and debugged offline.

Notations and Conventions

We consider a robot state $x \in \mathcal{X}$ evolving in a workspace $\mathcal{W} \subset \mathbb{R}^3$ with a navigation goal $g \in \mathcal{W}$ (typically specified as a high-level waypoint). At each time step, the robot acquires an undistorted, forward-facing RGB image $I \in \mathbb{R}^{H \times W \times 3}$ and a corresponding depth map $D \in \mathbb{R}^{H \times W}$, where H and W denote the image height and width, respectively. We use (u, v) (or equivalently (c, r)) to denote pixel coordinates, with u/c increasing to the right, v/r increasing downward, and the origin at the top-left of the image.
From (I, D) we construct an FPV traversability costmap $C \in \mathbb{R}^{H \times W}$ in image space and a configuration-space costmap $C^{\text{csp}} \in \mathbb{R}^{H \times W}$ after footprint inflation. Unless otherwise stated, for simplicity of notation, we use C to refer to both the FPV and C-space costmaps, the intended meaning should be evident from context. Goals projected into the image are written as pixel coordinates (u_g, v_g), and frontiers are selected as pixels (u_f, v_f) on C^{csp}. The navigation problem, in this notation, is to compute a collision-free path γ in the FPV image (and its corresponding 3D reprojection) that connects the current state x to a neighborhood of g while respecting traversability encoded by C^{csp}.

Approach Overview

The system processes undistorted RGB and sparse LiDAR to predict an FPV traversability costmap, projects a goal into the image, inflates obstacles in FPV to approximate the robot's footprint, selects a frontier using angular sector statistics, and runs A* on the pixel grid. The resulting image-space plan is reprojected into 3D for the downstream local planner and controller. All components run within a ROS graph and were evaluated in the high-fidelity Falcon simulator [1].

More concretely, we combine four ingredients (Figure 3.1, described later in detail):

1. **Test-time depth calibration**: An affine, log-domain correction that makes monocular depth metrically usable using only sparse LiDAR, with no network updates at inference.

2. **Depth-aware FPV C-space**: An analytic projection of vehicle width/length onto FPV, realized via separable row/column sliding-max filters with per-pixel depth consistency checks for efficiency and robustness.
3. **Interpretable frontiering**: Reasoning jointly about cost and minimal lethal depth based on angular sector statistics, yielding transparent choices that can be revalidated against the goal sector.
4. **Pixel-space A***: Search on the FPV grid with a simple cost renormalization $C'(p) = 1 + C(p)$ to preserve admissibility/consistency of the Euclidean heuristic.

These choices aim to bridge the efficiency of classical image-space planning with the robustness benefits of explicit geometry, while keeping complexity modest.

Positioning with Respect to Prior Work

Classical image-space planning (e.g., LAGR [11]) demonstrated that planning on a fixed FPV grid can reduce compute and avoid far-range 3D fusion issues, but highlighted sensitivity to projection error and the need for robust C-space inflation. We retain the FPV planning benefits while addressing footprint handling analytically and enforcing depth consistency in inflation.

More recent frontier learning approaches (e.g., LRN [20]) argue that long-range navigation is chiefly about selecting a good direction early. In contrast to the LRN formulations that assume no reliable depth and thus operate only in angular coordinates, we make monocular depth reliably usable via sparse-LiDAR calibration, and incorporate it directly into frontier selection and footprint handling. The result keeps frontiering simple and explainable while enabling metric reasoning when it matters.

Complementary to these FPV frontier methods, FITAM [6] learns from historical navigation data to predict metric costs for distant terrain directly from RGB images. It uses far-field visual features to guide navigation through a costmap defined in the metric workspace, without relying on dense range sensing or explicit depth calibration at inference. Our work shares FITAM's goal of exploiting far-field visual cues for metric decision making, but takes a more geometric and depth-centric route. We retain a single FPV grid tied to the current camera view, and rely on calibrated monocular depth and analytic C-space transforms instead of training a

far-field traversability network. In environments where sparse LiDAR is available, this calibration yields more metrically consistent depth in the mid/far field, which in turn simplifies footprint inflation and frontier evaluation while keeping the overall stack lightweight and auditable.

Contributions

This thesis makes the following contributions:

- A lightweight, test-time mono–LiDAR depth calibration that improves mid/far-range depth without retraining, enabling metric use of monocular depth in FPV.
- An analytic, depth-aware FPV C-space transform using separable sliding-max with depth gating to approximate footprint inflation in real time.
- Angular frontier selection with cost and depth statistics, including goal-aware revalidation, that remains interpretable and modular.
- Heuristic-consistent FPV A^* via cost renormalization, preserving the usual guarantees while keeping implementation simple.

Scope and Caution

Experiments are conducted in the Falcon simulator under a shared ROS-Noetic graph, comparing three frontiering strategies (LAGR-style row-wise, LRN-inspired openness, and an Angular Cost & Depth (ACD) variant) with the same perception and planning stack. The ACD strategy is generally competitive, especially on earlier segments, and both ACD and LRN improve over the purely row-wise LAGR baseline on longer traverses. We view these as encouraging indications rather than definitive claims. The results quantify the behavior only in simulation.

Due to the limited project duration (May 2025–present), the Falcon [1] simulator had to be brought into a stable configuration for end-to-end testing, leaving insufficient time to perform real-world deployments. In future work, we envision this long-range image-space sub-stack augmenting a physical off-road stack, such as Velociraptor [23], to provide valuable insight into the system's robustness under varying traction, lighting, and perceptual conditions.

Organization of the Thesis

Chapter 2 (Related Works) positions the approach relative to image-space planning, frontier learning, and traversability estimation. Chapter 3 (Pipeline) details the ROS nodes and data flow. Chapter 4 (Test-Time Depth Calibration) presents the formulation and algorithm for test-time mono depth calibration with sparse LiDAR. Chapter 5 (Frontier Selection) derives the C-space transform and angular sector based frontier selection strategies. Chapter 6 (Planning) describes FPV A* and path simplification. Chapter 7 (Experiments) reports quantitative and qualitative results. Chapter 8 concludes with limitations and directions for further work.

Chapter 2

Related Works

Long-range navigation in unstructured, off-road environments sits at the intersection of vision-based traversability estimation, frontier-driven long-horizon reasoning, and planners that operate either in the metric map or directly in the image (pixel) plane. We review the evolution from classical image-space planning (LAGR) [11] to recent frontier affordance learning (LRN) [20], and situate our contributions within this landscape – depth-calibrated FPV C-space inflation, angular frontier selection with depth-and-cost criteria, and A* with heuristic-normalized costs.

Image-Space Planning Lineage

A central line of work relevant to this thesis is the decision to plan directly in image space using a cost image and a projection of the goal into the camera frame. The DARPA LAGR [11] introduced a practical pipeline that (i) learns a color-to-cost mapping from near-range labels; (ii) applies a pseudo C-space transform in the image plane to account for robot width (i.e. row-wise dilation that grows toward the bottom of the image); (iii) projects the GPS goal into the image (with robust handling near the frame boundary); and (iv) runs A* on the pixel grid with admissible heuristics to produce a path, which is then handed to the local Cartesian stack as a subgoal just beyond near-range perception. The appeal is twofold: (1) constant state size (fixed number of pixels) decouples runtime from world size, and (2) bypassing explicit 3D fusion avoids misalignment and sparsity pitfalls of far-range mapping. Trade-offs

include lack of true metric guarantees in the far field, sensitivity to projection errors, and the need for robust pixel-space C-space inflation.

The LAGR rationale motivates our own FPV-first design, but we depart in several key ways. First, our FPV C-space transform is depth-aware via analytic projection of vehicle width and length as a function of image row. We implement it efficiently using separable, deque-optimized sliding maxima with a per-pixel depth-consistency test. Second, we modernize the perception front-end (foundation-model features and learned monocular depth with test-time LiDAR calibration). Third, we introduce angular frontier selection with explicit lethal-depth checks and average-cost statistics before A* planning in the pixel C-space.

Learning Frontiers for Long-Range Decisions

While LAGR shows that pixel-space planning can break myopia without building large metric maps, Long Range Navigator (LRN) [20] reframes the problem. Instead of building a full long-range map, it learns affordable frontiers from egocentric RGB and pick the heading that aligns with the distant goal. LRN trains a goal-agnostic affordance backbone to produce image-space hotspot heatmaps, projects them into angular bins, then uses a goal-conditioned head with temporal smoothing to choose the best heading. Labels are obtained largely from unlabeled videos via point tracking and hotspot mining, thereby scaling supervision. Empirically LRN reduces interventions in off-road courses by helping the stack commit to a good direction earlier than short-horizon heuristics.

This thesis shares LRN's thesis that long-range planning chiefly requires choosing a good frontier. However, while LRN demonstrates impressive long-horizon reasoning in image space, it explicitly assumes that no reliable depth estimate is available, projecting frontiers along camera rays using intrinsics only. This design choice simplifies perception allowing monocular RGB inputs and learned affordance maps but also restricts geometric reasoning – all decisions are made in angular image coordinates, and distances along rays remain unknown. Consequently, LRN can reason about which direction to travel but not how far obstacles actually lie, which limits precise footprint handling or configuration-space inflation.

Our approach departs fundamentally from this depth-free paradigm. We make monocular depth reliable and metrically usable by performing a test-time calibration with sparse LiDARs. This enables us to perform depth-aware configuration-space inflation and lethal-depth frontier evaluation. In contrast to LRN's assumption of unreliable depth, our pipeline leverages calibrated monocular depth to combine long-range visual reasoning with explicit geometric safety margins, effectively bridging image-space planning with metric-world robustness.

Hence, we bridge LAGR's pixel-planning machinery with LRN's long-horizon frontier insight by injecting explicit depth and cost statistics into frontier choice, rather than learning depth-free affordances end-to-end.

Traversability Estimation

Classic outdoor traversability pipelines use near-range geometry (stereo, LiDAR) to self-supervise a color/texture classifier for far range [3, 8, 14]. LAGR variants explored superpixels and learned color models in dynamic lighting [13], and polar or nonuniform map parameterizations to bias angular resolution toward horizons [3].

Early deep learning based methods, such as TerrainNet [16], presents a camera-only BEV semantic and geometric terrain model for high-speed off-road driving, using stereo self-supervised depth completion and multi-view fusion. Its multi-headed representation highlights the need for terrain semantics and geometry for navigation. RoadRunner M&M [19] predicts multi-range, multi-resolution traversability and elevation maps through multi-modal fusion (RGB + LiDAR voxel grids) and dense supervision from satellite DEMs and hindsight traversability estimators. Although BEV-based and fully supervised, RoadRunner emphasizes the increasing importance of far-range map prediction for high-speed off-road navigation.

FROLL [22] is a major precursor to modern learning-based traversability estimation based on self-supervision. Their scoped-learning model integrates reliable near-range LADAR features with sparse far-range overhead imagery or low-density sensor returns to predict terrain cost at extended ranges. This demonstrated that far-field cues could be exploited through self-supervised regression, anticipating later appearance-based long-range estimators.

HDIF [4] learns to predict traversability costmaps by combining exteroceptive visual and geometric data with proprioceptive IMU feedback. It defines a continuous traversability cost as the bandpower of the vertical acceleration signal, capturing how rough the terrain feels to the vehicle. Unlike occupancy or semantic methods, HDIF learns the nuanced relationship between appearance, geometry, and vehicle velocity, producing smooth continuous costmaps suitable for control optimization. Its velocity-conditioned learning architecture allows a single network to capture dynamic interactions between vehicle speed and perceived roughness, outperforming hand-engineered geometric roughness metrics.

SALON [21] extends this paradigm towards adaptive perception. It introduces a self-supervised online adaptation loop that fuses visual foundation model (VFM) features, proprioceptive roughness signals, and Gaussian process regression to learn risk-aware cost and speed maps. Unlike offline-trained systems, SALON updates its traversability predictions within seconds of new experience, avoiding out-of-distribution terrain through uncertainty reasoning. By mapping VFM features into a Bird's-Eye View (BEV) grid, it generalizes across multiple robot platforms and sensing modalities.

Beyond SALON, other contemporary approaches, such as Velociraptor [23] and WVN [15] demonstrate the value of using pretrained visual foundation models for dense terrain reasoning. These methods motivate our integration of off-the-shelf traversability estimation methods based on such generalizable foundation model features, replacing brittle semantic segmentation with generalizable visual embeddings.

Configuration-Space Transformation in FPV

Inflating obstacles by the robot footprint directly in the image is nontrivial because the pixel footprint is depth-varying and nonaxis-aligned. LAGR proposed a practical surrogate – row-wise max filtering with row-dependent widths (larger near the bottom), optionally combined with goal-row heuristics in A* [11]. We derive closed-form expressions for projected width and the horizon row from calibrated intrinsics, camera pitch, and height; then implement separable row/column sliding maxima with deque optimizations and a per-pixel depth-consistency gate. This preserves LAGR's computational benefits while increasing geometric fidelity for turning maneuvers and mixed-depth scenes.

Frontier-Based Navigation and Angular Sectoring

Classic frontier-based exploration selects boundaries between known and unknown regions to guide expansion [25]. In off-road long-range planning, a similar idea appears as choosing goal-aligned or affordable headings. LRN discretizes headings and scores them by affordance and goal proximity [20]. We instead compute angular sector statistics over the FPV C-space: mean cost, minimum lethal depth, and validity; then select either (i) the safest (max lethal depth) sector or (ii) the lowest-average-cost sector that satisfies minimal lethal depth. This keeps the frontier selection interpretable, geometry-aware, and explainable.

Planning on Pixel Grids

Search-based planning in image space uses grid neighborhoods and admissible heuristics. LAGR's A* introduced a simple admissible heuristic tied to a baseline per-step cost, plus move-direction restrictions that encouraged monotone progress [11]. Our planner normalizes the per-step cost as $C'(p) = 1 + C(p)$ so that it upper-bounds Euclidean distance, restoring admissibility/consistency of the Euclidean heuristic and avoiding silent suboptimality in low-cost free space. We also implement staged motion primitives to regulate curvature near the vehicle (dynamic rollouts) and a greedy line-of-sight simplifier to reduce pixel zig-zag before reprojection.

From LAGR and LRN to This Thesis

Why image space? Following LAGR, constant-size FPV grids amortize compute and improve far-range responsiveness.

Why frontier selection? Following LRN, long-range decisions are mostly about choosing where to aim, not mapping everything.

Our stance We replace learned affordances with explicit, calibrated FPV geometry, foundation-model traversability, analytic FPV footprint inflation, and interpretable angular-sector frontiers. This yields a modular stack that can be audited and tuned per site while retaining LRN's horizon benefits.

Overall, we identify three gaps this thesis targets as follows:

1. **Depth-aware FPV C-space:** Prior row-wise dilations lacked explicit per-row length handling and per-pixel depth consistency; we provide closed-form projection and separable, real-time inflation.
2. **Interpretable long-range frontiering:** Between LAGR (row sweep) and LRN (learned affordances), we add angular depth & cost angular statistics with goal-aware revalidation.
3. **Heuristic soundness for A*:** We explain why conventional cost range ($[0, 1]$) breaks consistency and provide a simple renormalization that restores A* guarantees in pixel space.

Chapter 3

Pipeline

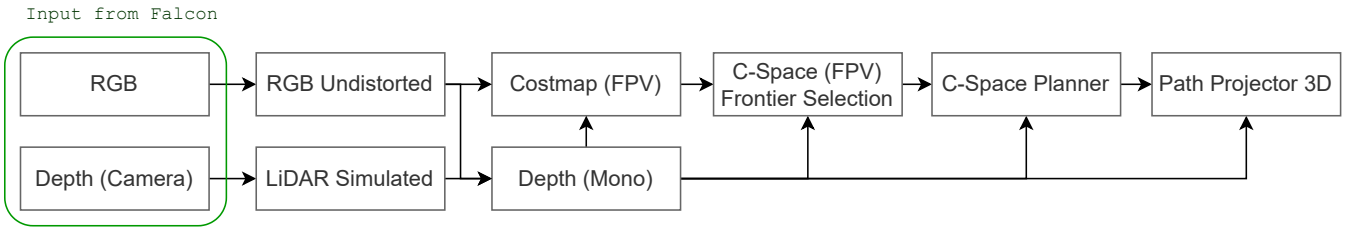

Figure 3.1: **ROS pipeline for long-range image-space planning in Falcon [1]:** The pipeline integrates simulated perception and planning components for off-road navigation. Raw RGB and sparse LiDAR inputs are processed through an undistortion and depth-estimation sequence to form a traversability-aware costmap in first-person view (FPV). The `frontier_selector` identifies a frontier in configuration space (C-space), while the `planner` computes a feasible image-space trajectory guided by curvature and traversability costs. The `path_projector` reprojects the resulting pixel-space plan into the 3D inertial frame for execution by the local planner.

Figure 3.1 illustrates the overall ROS-based perception–planning pipeline designed for long-range image-space navigation within the Falcon [1] simulation environment. The architecture modularizes depth perception, costmap generation, frontier selection, planning, and path projection to operate cohesively in real time. Each node in the pipeline fulfills a distinct role in synthesizing information from the simulated sensor suite and generating navigable trajectories in both image and 3-D inertial coordinates. Below we briefly describe the role of each node, excluding the standard Falcon base stack provided by our sponsor FieldAI [2]. The nodes developed and an-

alyzed in detail as part of this thesis are also discussed extensively in the subsequent chapters.

`lidar_sim` — Lightweight LiDAR Simulation

This node emulates LiDAR measurements by applying a precomputed LiDAR mask derived from the projection statistics of the LiDAR sensors in the Falcon simulator using classical image-processing techniques. This approach bypasses the computationally expensive ray-casting process of the Falcon LiDAR scanners, improving simulation speed by over 15× while maintaining a comparable level of measurement sparsity. Further details can be found in Appendix B.

`undistorter` — Camera Model Rectification and Intrinsics Update

The `undistorter` node standardizes the input imagery to match the physical camera parameters used in real deployments. It crops and rescales the RGB feed to align with the real sensor's aspect ratio and field of view. The node also republishes the adjusted *camera intrinsics* (focal length, principal point) to downstream components to ensure consistent geometric projection between simulated and real image coordinates. In the real vehicle configuration, the node also performs additional undistortion to compensate for lens distortions in the raw camera feed.

`mono_depth` — Monocular Depth Prediction and Sparse Correction

This node produces a dense depth map from the undistorted RGB image using a learned monocular depth estimation model. We employ the off-the-shelf MoGe-2 [24] to predict metric depth from a single monocular image. At test time, sparse LiDAR samples from `lidar_sim` are used to correct scale drift and local inconsistencies, further refining depth accuracy. Details of the test-time calibration with sparse LiDAR are elaborated in Chapter 4.

`costmap` — First-Person View Costmap Prediction

The `costmap` node fuses RGB imagery with LiDAR-corrected depth to predict dense per-pixel costmap in the image (FPV) plane. To convert visual inputs into a traversability-

aware costmap, we leverage SALON [21], which projects visual foundation model features into a bird's-eye-view (BEV) map and learns to associate these features with IMU-based roughness experienced by the vehicle. This pairing enables the system to infer local traversability and to identify objects or regions that are significantly out of distribution. This BEV traversability map (costmap) is then projected into the FPV space, which is subsequently used for downstream long-range planning tasks. The visual features are extracted using DINOv2 [18], whose output is originally 14× lower in spatial resolution. They are additionally upsampled via LoftUp [10] to recover high-resolution feature maps, allowing finer geometric detail at greater distances.

`frontier_selector` — Goal Projection, C-Space Transform, and Frontier Selection

The `frontier_selector` node first projects the navigation goal from the inertial 3-D coordinate frame into the image coordinate frame to align with perception outputs. Simultaneously, it converts the costmap produced by the `costmap` node into a configuration-space (C-space) cost representation. Using the projected goal as a spatial reference, the node then identifies a frontier pixel on the C-space costmap. Frontier selection is guided by heuristics based on local cost statistics and global objectives such as reachability and clearance. The selected frontier is subsequently published to the mid-range, image-space planner, which uses it to generate a reference plan for the local planner inside the FieldAI stack. A more detailed description is presented in Chapter 5.

`planner` — Image-Space Path Generation

The `planner` node computes a candidate path directly in the C-space FPV costmap produced by the `frontier_selector`. Internally, it employs an A*-based search algorithm operating over the C-space costmap, where each pixel encodes traversability and curvature constraints, and constructs feasible paths by minimizing a composite cost that accounts for both geometric smoothness and terrain safety. The resulting image-space trajectory captures the globally optimal route toward the selected frontier within the available field of view. A detailed discussion of the planner's internal formulation, including its cost term composition, curvature constraints, and

heuristic, is provided separately in Chapter 6.

`path_projector` — Back-Projection to 3D Inertial Frame

The `path_projector` maps the 2-D image-space trajectory back into the 3-D inertial coordinate frame using the predicted monocular depth. Each waypoint in the image plane is re-projected into metric world coordinates. The resulting 3D path is subscribed by the local planner to be used as a reference.

`meters` — Odometry-Based Benchmarking and Runtime Metrics

This node continuously monitors the vehicle's odometry and goal updates to compute both cumulative and per-goal travel distances in real time. Each time a new navigation goal is issued, the node logs the distance covered toward the previous goal and resets its local distance accumulator. The measurements provided by this metric are used for benchmarking in the Experiments section (Section 7.2).

Chapter 4

Test-Time Depth Calibration

Monocular depth estimation networks suffer from range-dependent bias and scale inconsistency, particularly in outdoor or off-road environments where the depth range spans several orders of magnitude. To mitigate this, we apply a lightweight test-time depth calibration that aligns the dense monocular depth predictions with sparse LiDAR measurements. This calibration is non-parametric at training time, and performed entirely during inference, requiring no network retraining or gradient updates.

Formulation

Let $\hat{D}_{\text{mono}}(u, v)$ denote the raw monocular depth prediction at pixel (u, v), and $D_{\text{lidar}}(u, v)$ the corresponding projected LiDAR depth. Denote their validity masks as $\mathcal{M}_{\text{mono}}$ and $\mathcal{M}_{\text{lidar}}$, respectively. We consider only the intersection region

$$\mathcal{M} = \mathcal{M}_{\text{mono}} \wedge \mathcal{M}_{\text{lidar}},$$

where both modalities have valid projections.

The calibration seeks a function $f_\theta(\cdot)$ that best maps the monocular predictions to the LiDAR depth:

$$D_{\text{lidar}}(u, v) \approx f_\theta\Big(\hat{D}_{\text{mono}}(u, v)\Big), \quad (u, v) \in \mathcal{M}.$$

We model $f_\theta(\cdot)$ as an n-th order polynomial:

$$f_\theta(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Empirically, we find that a first-order (affine) model suffices for calibration in our deployment setting, since most of the systematic bias manifests as a global scale and offset rather than higher-order nonlinearity.

Log-Domain Equalization

To ensure that both short- and long-range depths contribute comparably to the regression, e.g., 1 m and 200 m regions, we perform the least-squares fit in the `log-depth` domain. This logarithmic transformation compresses the dynamic range of depth values and prevents the regression from being dominated by large-distance pixels. In other words, the log transform enforces approximately uniform weighting across distance scales.

$$x'_i = \log \hat{D}_{\text{mono}}(u_i, v_i), \qquad y'_i = \log D_{\text{lidar}}(u_i, v_i), \quad (u_i, v_i) \in \mathcal{M}.$$

Least-Squares Estimation

We construct the polynomial design matrix

$$\mathbf{X} = \begin{bmatrix} x'_1 & (x'_1)^2 & \cdots & (x'_1)^n & 1 \\ x'_2 & (x'_2)^2 & \cdots & (x'_2)^n & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x'_m & (x'_m)^2 & \cdots & (x'_m)^n & 1 \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}, \quad \mathbf{Y} = \begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_m \end{bmatrix} \in \mathbb{R}^m$$

The least-squares solution minimizing $\|\mathbf{Y} - \mathbf{X}\mathbf{A}\|_2^2$ is given in closed form by

$$\mathbf{A} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

This coefficient vector is then used to predict calibrated depths for all valid monocular pixels. Finally, the outputs are exponentiated to revert to metric depth:

$$\hat{D}_{\text{calib}}(u, v) = \exp\Big(f_\theta(\log \hat{D}_{\text{mono}}(u, v))\Big), \quad (u, v) \in \mathcal{M}_{\text{mono}}.$$

Algorithm 1 Least-Squares Mono–LiDAR Calibration

Require: Monocular depth $\hat{D}_{\text{mono}}$, LiDAR depth D_{lidar}, masks $\mathcal{M}_{\text{mono}}, \mathcal{M}_{\text{lidar}}$, polynomial order n (default 1)
Ensure: Calibrated depth map $\hat{D}_{\text{calib}}$

$\mathcal{M} \leftarrow \mathcal{M}_{\text{mono}} \wedge \mathcal{M}_{\text{lidar}}$
$X \leftarrow [\, \hat{D}_{\text{mono}}(u, v) : (u, v) \in \mathcal{M} \,]$
$Y \leftarrow [\, D_{\text{lidar}}(u, v) : (u, v) \in \mathcal{M} \,]$
$\mathbf{X} \leftarrow [\log(X), (\log(X))^2, \ldots, (\log(X))^n, 1]$
$\mathbf{Y} \leftarrow \log(Y)$
$\mathbf{A} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$ ▷ least-squares coefficients
Build $\mathbf{X}_{\text{test}}$ from $\hat{D}_{\text{mono}}$ on $\mathcal{M}_{\text{mono}}$
$\mathbf{X}_{\text{test}} \leftarrow [\log(\hat{D}_{\text{mono}}), \ldots, 1]$
$\hat{Y} \leftarrow \mathbf{A}\mathbf{X}_{\text{test}}$
$\hat{D}_{\text{calib}}(u, v) \leftarrow \exp(\hat{Y})$ for $(u, v) \in \mathcal{M}_{\text{mono}}$, else 0
return $\hat{D}_{\text{calib}}$

Algorithmic Summary

Algorithm 1 summarizes the calibration procedure. It constructs polynomial features, performs least-squares fitting in the log-depth domain, and applies the fitted transformation to obtain a globally calibrated depth map.

As shown later in Section 7.1, the proposed log-linear calibration substantially reduces depth bias for different depth ranges across diverse off-road environments.

Note that the calibration itself is performed using only sparse simulated LiDAR measurements, whereas the evaluation is carried out using the dense ground truth depth maps from the depth camera, which are never seen by the calibration procedure. This is to further validate the generalization capability of the proposed test-time calibration beyond the sparse calibration samples.

Chapter 5

Frontier Selection

5.1 Goal Projection

To enable reasoning in FPV, the high-level navigation goal originally expressed in the global coordinate frame is first projected into the FPV frame of the camera.

Transformation and Projection

Each goal pose is first transformed into the camera coordinate frame using the corresponding TF transform. Let the goal position in the camera frame be (X_c, Y_c, Z_c). Using the camera intrinsic matrix K, the homogeneous image-plane projection is written compactly as

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = K \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \\ 1 \end{bmatrix}, \qquad (u, v) = \left(\frac{u'}{w'}, \frac{v'}{w'} \right)$$

If $Z_c \leq 0$, the goal lies behind the optical plane, and direct projection becomes invalid; such cases are handled separately as discussed later.

Clipping to the Visible Region

When the projected pixel (u, v) falls outside the image boundaries $[0, W) \times [0, H)$ due to large range or off-axis placement, we compute the intersection of the goal line with the image border. The line joining the start point (x_0, y_0) and the raw projected goal (x_1, y_1) is parameterized as $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$, and the smallest positive intersection with the border rectangle defines the clipped goal. This step guarantees that the visual target remains inside the camera view, as illustrated in Fig. 5.1, where the red segment shows the original out-of-view projection and the green segment shows the final clipped line within the valid FPV region.

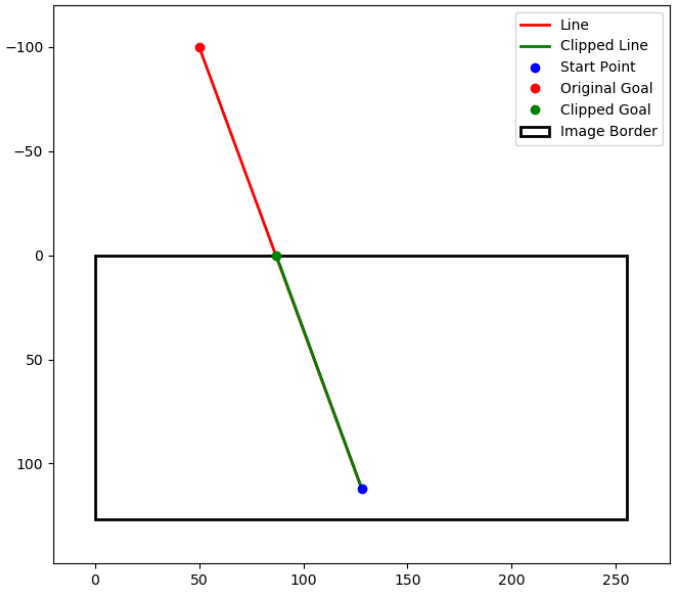

Figure 5.1: **Goal projection and clipping.** The red line indicates the original projected goal extending beyond the image border, while the green line (superimposed on red) shows the clipped goal within the visible FPV region. The blue marker denotes the starting pixel, and the green marker the adjusted in-view goal. The black border represents the boundary of the FPV image.

Handling Goals Behind the Camera

If $Z_c \leq 0$, the goal is geometrically behind the camera. Rather than discarding such goals, we intentionally assign a directional pseudo-goal on the visible border of the

image. This is to ensure that the vehicle first reorients itself by turning toward the appropriate side to bring the original goal into the visible FPV region.

The pseudo-goal is chosen along the border corresponding to the lateral direction of the true goal (left/right) with a fixed fraction r_f forward along the vertical axis:

$$y_{\text{goal}} = \max(y_0 - r_f H, 0), \qquad x_{\text{goal}} = \begin{cases} 0, & x_1 \leq x_0, \\ W - 1, & \text{otherwise.} \end{cases} \tag{5.1}$$

Here, $r_f \in (0, 1]$. We choose $r_f = 0.5$ to place the pseudogoal approximately halfway up the visible region, encouraging a forward-turning maneuver that realigns the camera towards the original behind-the-vehicle direction.

Algorithmic Overview

Algorithm 2 summarizes the complete procedure, combining TF transformation, projection, clipping, and pseudo-goal handling. All outputs are guaranteed to lie within the FPV frame. The projected goal becomes the anchor for C-space frontier selection.

Algorithm 2 Goal Projection and Clipping in FPV Image Space

Require: Goal pose $\mathbf{p}_g$ in global frame; intrinsics K; image size (W, H); transform buffer $\mathcal{T}$; start pixel $\mathbf{p}_0 = (x_0, y_0)$
Ensure: Projected or pseudo-goal pixel $\mathbf{p}_{\text{goal}}$

$\mathbf{p}_c = \mathbf{T}_{cg}\mathbf{p}_g$ ▷ Transform to camera frame
if $Z_c \leq 0$ **then** ▷ Goal behind camera
    $x_{\text{goal}} = 0$ or $W-1$ (from sign of X_c)
    $y_{\text{goal}} = \max(y_0 - r_f H, 0)$
    return $(x_{\text{goal}}, y_{\text{goal}})$
end if
Homogeneous projection: $(u', v', w') = K(X_c/Z_c, Y_c/Z_c, 1), \quad (u, v) = \left(\frac{u'}{w'}, \frac{v'}{w'}\right)$
if $0 \leq u < W$ and $0 \leq v < H$ **then**
    return (u, v)
end if
Parameterize line $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$, $\mathbf{p}_1 = (u, v)$
Compute intersections with $\{x = 0, x = W - 1, y = 0, y = H - 1\}$
$t^* = \min\{t > 0\}$; $\mathbf{p}_{\text{goal}} = \mathbf{p}(t^*)$
return $\mathbf{p}_{\text{goal}}$

5.2 Configuration Space Transform

The costmap generated from a first-person view (FPV) camera captures obstacle information at the pixel level. However, to enable effective motion planning and collision avoidance, it is necessary to account for the physical footprint of the robot. This is accomplished by generating a *configuration space* (C-space) costmap, where the robot is abstracted as a point, and each cell is inflated to reflect the area that the robot's body would occupy if it were centered on that cell.

5.2.1 Derivation

We derive a method to project the real-world vehicle width and length onto image coordinates, based on the geometry of a forward-facing camera mounted on a vehicle. Given:

- Vehicle width in real-world coordinates: w_r (in meters)
- Camera intrinsics: focal lengths f_x, f_y and principal point (c_x, c_y)
- Camera height above ground: h (in meters)
- Camera pitch angle downward from the horizontal: θ (in radians)
- Target pixel row in the image: y

We derive:

1. A mapping from image row y to real-world ground-plane depth $Z(y)$
2. The projected width $w_p(y)$ and length $l_p(y)$ of the vehicle in pixels at row y.

Pinhole Camera Model

The standard pinhole camera projection for a 3D point $\mathbf{P}_c = [X, Y, Z]^T$ in the camera coordinate frame is:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \Rightarrow \qquad \begin{cases} y = f_y \cdot \frac{Y}{Z} + c_y \\ x = f_x \cdot \frac{X}{Z} + c_x \end{cases} \tag{5.2}$$

Coordinate System and Camera Pose

The world coordinate system is defined such that:

- The positive (X, Y, Z) is (right, bottom, forward).
- The ground plane lies at $Y = h$.
- The camera is mounted at height h above the ground, thus camera is at $Y = 0$.
- The camera is facing downward by angle θ

Let a point on the ground plane in world coordinates be $\mathbf{P}_w = (X_w, 0, Z_w)^T$.
The camera translation vector is $T = (0, -h, 0)^T$, and the rotation matrix for a pitch angle θ about the X-axis is:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Thus, the point $\mathbf{P}_w$ in camera coordinates is given by

$$\mathbf{P}_c = R(\mathbf{P}_w - T) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_w \\ h \\ Z_w \end{bmatrix} = \begin{bmatrix} X_w \\ h\cos\theta - Z_w\sin\theta \\ h\sin\theta + Z_w\cos\theta \end{bmatrix} \tag{5.3}$$

Ground Plane Depth Estimation

Substituting Y and Z from Equation 5.3 into the pinhole vertical coordinate in Equation 5.2, we get

$$\begin{aligned}
& y = f_y \cdot \frac{h\cos\theta - Z_w\sin\theta}{h\sin\theta + Z_w\cos\theta} + c_y \\
\Rightarrow\quad & y - c_y = f_y \cdot \frac{h\cos\theta - Z_w\sin\theta}{h\sin\theta + Z_w\cos\theta} \\
\Rightarrow\quad & (y - c_y)(h\sin\theta + Z_w\cos\theta) = f_y(h\cos\theta - Z_w\sin\theta) \\
\Rightarrow\quad & (y - c_y)h\sin\theta + (y - c_y)Z_w\cos\theta = f_y h\cos\theta - f_y Z_w\sin\theta \\
\Rightarrow\quad & (y - c_y)Z_w\cos\theta + f_y Z_w\sin\theta = f_y h\cos\theta - (y - c_y)h\sin\theta \\
\Rightarrow\quad & Z_w\left[(y - c_y)\cos\theta + f_y\sin\theta\right] = h\left[f_y\cos\theta - (y - c_y)\sin\theta\right]
\end{aligned}$$

Thus, the ground depth $Z(y)$ corresponding to image row y is:

$$Z(y) = \frac{h\,[f_y \cos\theta - (y - c_y)\sin\theta]}{(y - c_y)\cos\theta + f_y \sin\theta} \tag{5.4}$$

Small-Angle Approximation For small pitch angles ($\theta \ll 1$), $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. This leads to

$$Z(y) \approx \frac{h f_y}{(y - c_y) + f_y \theta}$$

Since the resolution of the FPV image and camera parameters remain fixed, we can precompute the ground depth once at the start of the mission. Therefore, the small-angle approximation is not needed in our case.

Projected Vehicle Width

At any depth Z, a vehicle of width w_r (in meters) spans $X = \pm w_r/2$ in the real world. These 3D points in camera coordinates project to image u coordinates as:

$$u = f_x \cdot \frac{X}{Z} + c_x$$

Here, the left and right edges are given by

$$u_L = f_x \cdot \frac{-w_r/2}{Z} + c_x, \quad u_R = f_x \cdot \frac{w_r/2}{Z} + c_x$$

Thus, the projected pixel width at row y based on ground depth estimate $Z(y)$ is:

$$w_p(y) = u_R - u_L = f_x \cdot \frac{w_r}{Z(y)} = \frac{f_x \cdot w_r}{h} \cdot \frac{(y - c_y)\cos\theta + f_y \sin\theta}{-(y - c_y)\sin\theta + f_y \cos\theta} \tag{5.5}$$

Note that $w_p(y)$ is a nonlinear rational function of the form $w_p(y) = \dfrac{A(y - c_y) + B}{C(y - c_y) + D}$ for $\theta \neq 0$. Only when the camera is level ($\theta = 0$), it becomes linear, i.e. $w_p(y) = \dfrac{f_x\, w_r}{h\, f_y}\,(y - c_y)$, meaning the projected width increases steadily with image row without any nonlinear bending. As the camera tilts more up or down, the envelope of the vehicle width bends more sharply.

Horizon Row

In the camera frame, the optical axis (aligned with the Z-axis) is rotated downward by pitch θ, resulting in a unit direction vector $\mathbf{d} = (0,\ -\sin\theta,\ \cos\theta)^T$. This vector points in the direction the camera is looking, i.e. parallel to the ground plane. Substituting $\mathbf{d}$ into the pinhole model for vertical projection in Equation 5.2, we get

$$y_{\text{horizon}} = f_y \cdot \frac{-\sin\theta}{\cos\theta} + c_y$$

$$\Rightarrow\ y_{\text{horizon}} = c_y - f_y \cdot \tan(\theta) \tag{5.6}$$

In general, the vertical position of the horizon row as a function of the camera pitch angle θ can be summarized as follows:

- $(\theta = 0)$: the camera looks straight ahead and $y_{\text{horizon}} = c_y$
- $(\theta > 0)$: the camera is pitched downward, and the horizon appears above the principal point: $y_{\text{horizon}} < c_y$. This is the case for us.
- $(\theta < 0)$: the camera is pitched upward, and the horizon moves below the principal point.

Projected Vehicle Length

Estimating the projected length of the vehicle in pixels along the vertical axis requires calculating the vertical pixel distance corresponding to the real-world vehicle length l_r at depth $Z(y)$ as follows:

1. For a given image row y_{rear} corresponding to the rear of the vehicle, the depth is:

$$Z_{\text{rear}} = Z(y_{\text{rear}})$$

2. Assuming the vehicle lies along the depth axis of the camera, the depth of the front of the vehicle along the optical axis is:

$$Z_{\text{front}} = Z_{\text{rear}} + l_r$$

3. To find the image row y_{front} corresponding to Z_{front}, we invert the ground depth

function $Z(y)$ in Equation 5.4 to obtain:

$$y(Z) = \frac{f_y \left(h \cos\theta - Z \sin\theta\right)}{Z \cos\theta + h \sin\theta} + c_y$$

4. The projected length in pixels at the current row $l_p(y = y_{\text{rear}})$ is then the vertical pixel distance between $y = y_{\text{rear}}$ and y_{front}:

$$l_p(y) = y - y(Z_{\text{front}}) \tag{5.7}$$

Figure 5.2 shows projected vehicle width, half-length, and horizon row along the image row based on the parameters mentioned in the caption of the figure.

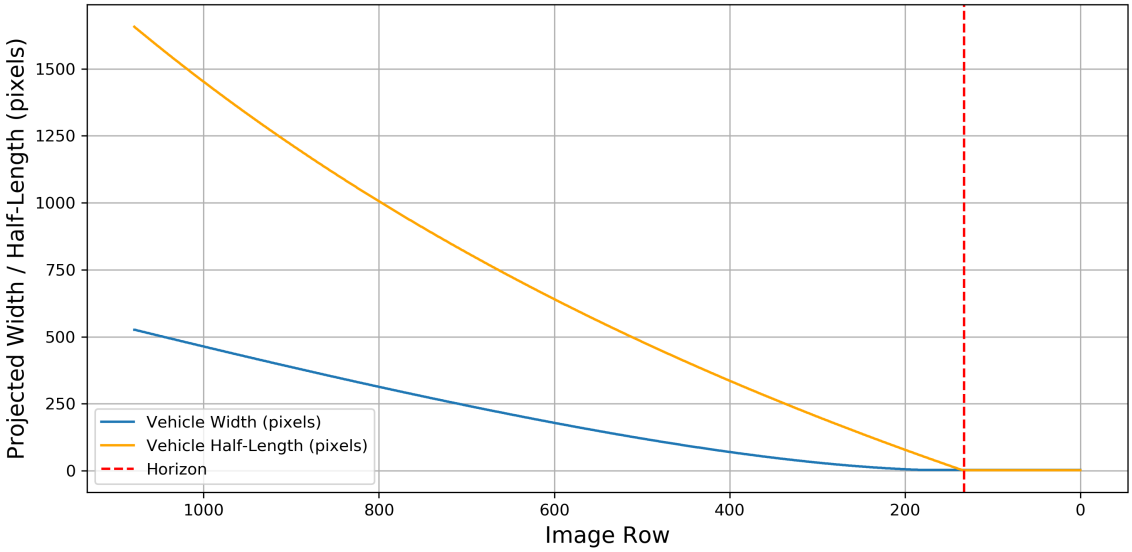

Figure 5.2: Projected vehicle width and half-length, along with the horizon row (Equations 5.5, 5.7, 5.6), calculated $f_x = f_y = 960$, $c_x = 960$, $c_y = 540$, image resolution of 1080×1980, pitch angle of 23°, and camera height, vehicle width, and vehicle length of 1.5, 2.0, and 4.5 meters, respectively.

5.2.2 Implementation

Having established all the necessary components for the C-space transformation, we will now turn to the question of efficient implementation. As derived in Section 5.2.1, under a pinhole projection model, a 3D point (X, Y, Z) maps to the image

coordinates (x, y) via $x = fX/Z$ and $y = fY/Z$, where f is the focal length. The same physical width w varies nonlinearly appearing larger near the bottom of the image (small Z), and smaller higher up (large Z), resulting in a slanted quadrilateral footprint in image space.

To achieve approximate collision avoidance via projected vehicle-dimension inflation in the FPV costmap, first, we employ a row-wise maximum filter inspired by LAGR [11]. The key insight underlying this approach is that the robot width corresponding to the horizontal dimension in image space is the critical factor for collision avoidance. Forward motion of the robot maps to vertical displacement in the image, where the inaccuracies due to row-wise filtering effects compress the robot length. Consequently, we apply horizontal filtering exclusively, using row-dependent kernel sizes that approximate the projected robot width.

However, we empirically find horizontal filtering alone to be insufficient. In particular, during turning maneuvers in front of obstacles, the robot may still collide if no vertical or column-wise inflation is applied. To address this, we additionally perform column-wise inflation using a smaller fraction of the projected vehicle length, thereby accounting for potential collisions during rotational motion and ensuring a more robust obstacle avoidance strategy.

Variable-Sized Sliding Window Maximum

Let $I \in \mathbb{R}^{H \times W}$ denote the FPV costmap of height H and width W. For each row i and each column j, we compute $I'_{i,j} = \max_{v \in [j-k_i,\, j+k_i] \cap [0, W-1]} I_{i,v}$, where k_i is a row-dependent half-window size based on the estimated ground-plane depth $Z(i)$ following Equation 5.5. Thus, our row-wise C-space transformation essentially reduces to solving a sliding window maximum problem efficiently for each row.

Efficient C-Space Transformation

We realize an efficient deque-based implementation for C-space inflation here. Further details on the deque and its necessity for efficient C-space transform are given in Appendix A.

In addition to the row-wise sliding maximum operation based on the projected ve-

hicle width, a per-pixel depth consistency check is introduced before adding a new candidate into the deque. Pixels whose depth values differ by more than ΔZ from the current center depth are ignored:

$$\text{Include } v \text{ only if } |Z_{r,v} - Z_{r,c}| < \Delta Z.$$

This effectively removes outliers corresponding to obstacles at different depth layers or sudden terrain discontinuities, resulting in a geometrically valid inflation.

Algorithm 3 Row-Wise Sliding Maximum with Deque Optimization

Require: Input row $I_r[0{:}W{-}1]$, depth row Z_r (optional), window size w_r, depth tolerance ΔZ
Ensure: Output row $I'_r[0{:}W{-}1]$

if $w_r \leq 1$ **then**
  $I'_r \leftarrow I_r$ ▷ No filtering needed
  return
end if
$w_r \leftarrow w_r + 1$ if w_r is even ▷ Ensure odd window size
$k_r \leftarrow \lfloor w_r/2 \rfloor$
Initialize empty deque $\mathcal{D}$
for $c = 0$ to $W - 1$ **do**
  $L \leftarrow \max(0,\, c - k_r)$; $R \leftarrow \min(W - 1,\, c + k_r)$ ▷ Current window bounds
  Remove elements from $\mathcal{D}$ where $\mathcal{D}_{\text{front}} < L$

  Extend window:
  $s \leftarrow (\mathcal{D} \text{ is empty})?L : \mathcal{D}_{\text{back}} + 1$
  for $j = s$ to R **do**
    if $|Z_r[j] - Z_r[c]| > \Delta Z$ **then**
      continue ▷ Exclude inconsistent depth
    end if
    while $\mathcal{D}$ not empty and $I_r[j] \geq I_r[\mathcal{D}_{\text{back}}]$ **do**
      Pop back of $\mathcal{D}$ ▷ Remove smaller values
    end while
    Push j to back of $\mathcal{D}$
  end for
  $I'_r[c] \leftarrow I_r[\mathcal{D}_{\text{front}}]$ ▷ Current window maximum
end for

Row-Wise Aggregation Across Image The full horizontal inflation for the entire costmap is obtained by applying Algorithm 3 independently to each image row:

$$I'(r, c) = \text{RowMax}\Big(I(r, :),\, w_r\Big), \quad \forall r \in [0, H-1].$$

The window size w_r varies per row according to the projected vehicle width in Equation 5.5, which expands toward the image bottom (closer to the vehicle) and contracts near the horizon.

Vertical (Column-Wise) Sliding Maximum Filtering The row-wise filtering above expands the costmap laterally to represent the robot's horizontal footprint at each row. This is generally sufficient only if the robot experiences translational forward motion. However, when the robot moves forward and rotate near obstacles, it may get into collision unless the vertical (image row) direction is not inflated at all. Thus, followed by the row-wise cost expansion, we also apply a column-wise maximum filter with a fraction of the projected robot length in Equation 5.7 on the row-filtered result I', producing the final C-space costmap I'':

$$I''_{r,c} = \max_{u \in [r-l_r,\, r+l_r]} I'_{u,c},$$

where $l_r = \lfloor \alpha L_r / 2 \rfloor$, where L_r represents the projected vehicle length (in pixels) at image row r, as derived in Equation 5.7, and $\alpha \in (0, 1]$.

Conceptual Overview The column-wise sliding maximum filtering process is summarized in Algorithm 4. It takes as input the horizontally inflated costmap and produces a vertically expanded version, completing the separable C-space transformation. Each image column is processed independently. For each column index c, the algorithm maintains a deque storing row indices corresponding to potential maxima within the current vertical window. As the window slides downward along the column:

- The front of the deque always holds the index of the current maximum.
- Out-of-window indices are removed from the front.
- New indices are appended to the back while removing smaller values to preserve

a monotonically decreasing sequence of pixel intensities.

This ensures that each row's maximum can be computed in constant amortized time.

Algorithm 4 Column-Wise Sliding Maximum with Deque Optimization

Require: Input column $I'_c[0{:}H{-}1]$, vector of window lengths $\{L_r\}_{r=0}^{H-1}$
Ensure: Output column $I''_c[0{:}H{-}1]$

Initialize empty deque $\mathcal{D}$
for $r = 0$ to $H - 1$ **do**
    $l_r \leftarrow L_r$
    if $l_r \leq 1$ **then**
        $I''_c[r] \leftarrow I'_c[r]$
        continue
    end if
    $l_r \leftarrow l_r + 1$ if l_r is even ▷ Ensure symmetric window
    $k_r \leftarrow \lfloor l_r/2 \rfloor$
    $T \leftarrow \max(0,\, r - k_r)$; $B \leftarrow \min(H - 1,\, r + k_r)$ ▷ Vertical window bounds
    Remove all indices from $\mathcal{D}$ where $\mathcal{D}_{\text{front}} < T$

    Extend window:
    $s \leftarrow (\mathcal{D}$ is empty$)?\ T : \mathcal{D}_{\text{back}} + 1$
    for $i = s$ to B **do**
        while $\mathcal{D}$ not empty and $I'_c[i] \geq I'_c[\mathcal{D}_{\text{back}}]$ **do**
            Pop back of $\mathcal{D}$
        end while
        Push i to back of $\mathcal{D}$
    end for
    $I''_c[r] \leftarrow I'_c[\mathcal{D}_{\text{front}}]$ ▷ Current maximum in window
end for

Two-Stage Separable Transformation By first performing the row-wise (horizontal) filtering followed by the column-wise (vertical) filtering, we achieve a *separable approximation* of the full 2D configuration-space dilation:

$$I'' = \mathcal{M}_{\text{col}}\Big(\mathcal{M}_{\text{row}}(I)\Big),$$

where $\mathcal{M}_{\text{row}}$ and $\mathcal{M}_{\text{col}}$ denote the row- and column-wise maximum filters, respectively. This separable formulation allows the algorithm to approximate the nonlinear vehicle

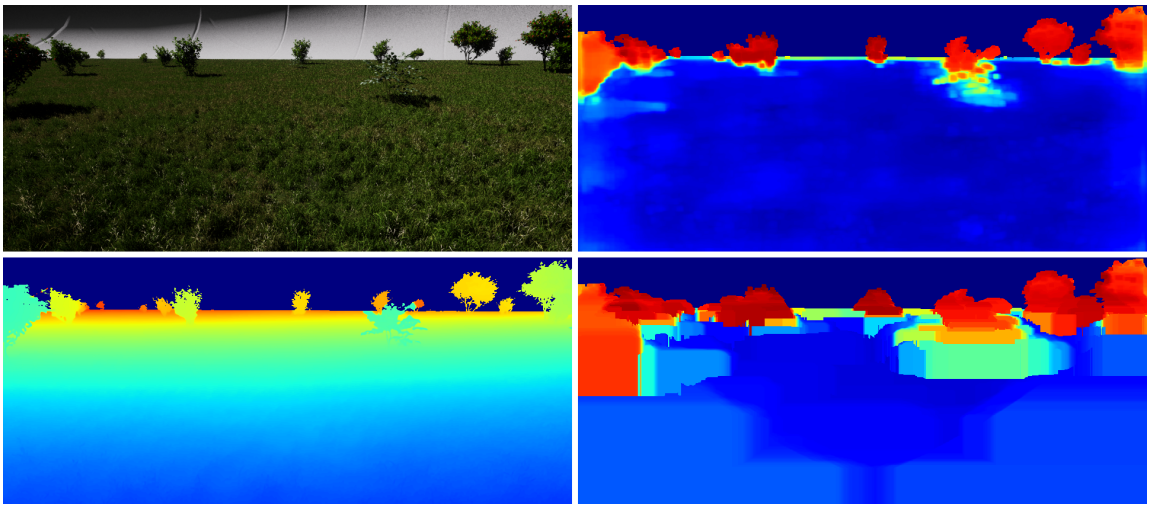

Figure 5.3: Sample illustration of **C-space transformation**. (Top-left) RGB image, (bottom-left) LiDAR-calibrated depth map, (top-right) FPV predicted costmap, and (bottom-right) C-space costmap.

footprint efficiently while retaining real-time performance characteristics. Figure 5.3 shows a sample C-space transformation.

Runtime The amortized runtime of our implementation is $O(HWk)$ for the FPV image of shape $H \times W$ and $k \ll H$. See Appendix A for details.

5.3 Frontier Selection Strategies

Autonomous navigation in cluttered environments requires the robot to select suitable frontier points that balance goal-directed progress, obstacle avoidance, and path length. Based on the goal projected into the FPV image following Section 5.1, in this section, we present three progressively refined FPV frontier selection strategies. They are ordered from simpler, row-wise selection strategies to more sophisticated angular cost and depth-aware approaches.

1. **LAGR:** Inspired by the DARPA LAGR paper [11], this method performs a row-wise frontier search from the goal row toward the origin row, explicitly encouraging forward (upward in the FPV image) motion towards the goal. We had to customize it for our work, as LAGR does not explicitly describe any long-range planner that repeatedly selects frontiers in the image space.
2. **LRN:** Following the LRN approach [20], this algorithm uses angular sector-wise statistics to select the farthest valid frontier in the sector with the largest minimum lethal depth, prioritizing generally more open directions. In our implementation, we adapted it because the original LRN learns affordance and frontier jointly without depth information, assuming no reliable depth is available. Instead, we proxy affordance with our explicit dense costmap prediction model, and include a reasonable, calibrated depth estimate as already mentioned in Chapter 4.
3. **ACD:** This strategy leverages angular sector-wise cost statistics to choose a sector with lower average cost that also satisfies a minimum lethal depth. The farthest valid point within the selected sector is then chosen as the frontier.

Note that we refer to the LAGR [11] and LRN [20] baselines using the shorthand "LAGR" and "LRN". However, these labels should be understood as LAGR-style and LRN-inspired strategies rather than exact implementations of the original methods. This is because they are adapted to our problem setting, and in several cases extended beyond what the original papers explicitly describe (e.g., incorporation of explicit depth estimates or repeated long-range frontier selection). We use the shorter names purely for brevity and to clearly indicate the conceptual lineage of each strategy.

5.3.1 LAGR

The LAGR algorithm, shown in Algorithm 5, identifies a valid frontier point along the line of sight from the goal to the robot's current position. First, the algorithm checks if the goal itself is valid according to the costmap and collision threshold; if so, it returns the goal as the frontier. Otherwise, a set of points along the line connecting the origin and goal is sampled, and a row-wise search is performed for valid frontier candidates. For each sampled row, the algorithm scans left and right along the row to locate the nearest valid pixels whose cost is below the threshold. If multiple candidates exist, the pixel closest to the sampled goal is selected; if only one valid pixel exists, it is chosen. If no valid pixels are found across all sampled rows, the algorithm returns a failure indicator. This row-wise strategy encourages forward motion toward the goal while respecting collision constraints.

Algorithm 5 LAGR-Style, Row-Wise Frontier Selection

Require: Costmap C, mask M, depth map D, origin (r_o, c_o), goal (r_g, c_g), maximum cost $C_{\max}$, number of rows to sample n, column-wise step size Δc

Ensure: Frontier (r_f, c_f)

Initialize $(r_f, c_f) \leftarrow (-1, -1)$
if $C(r_g, c_g) < C_{\max}$ **then**
    return (r_g, c_g)
end if
Sample n points along the line from (r_g, c_g) to (r_o, c_o) excluding (r_o, c_o): $\{(r_i, c_i)\}$
Row-wise search for valid candidates relative to sampled goal point:
for each sampled point (r, c) **do**
    $c_l \leftarrow$ Immediate left column c_i from c with step Δc with $C(r, c_i) < C_{\max}$
    $c_r \leftarrow$ Immediate right column c_i from c with step Δc with $C(r, c_i) < C_{\max}$
    if both (r, c_r) and (r_i, c_l) exist **then**
        return $((r_f, c_f) =$ closest to sampled goal $(r, c))$
    else if only one valid pixel exists **then**
        return $((r_f, c_f) =$ valid pixel$)$
    end if
end for
return (r_f, c_f) ▷ Failure

Angular Sector Computation

Before both the LRN and ACD algorithms are applied, we perform a common pre-computation step to obtain angular sector-wise statistics (Algorithm 6). In this step, the field of view is partitioned into discrete angular sectors, and for each sector, we compute statistics such as average cost, minimum lethal depth, and a validity flag based on collision and depth thresholds. These statistics form the basis for the sector-level reasoning used by both algorithms to identify promising frontier directions.

More specifically, given the robot's image-plane origin (r_o, c_o) and allowable angular field of view $[\theta_{\min}, \theta_{\max}]$, the environment is partitioned into discrete angular sectors of stride $\Delta\theta$. Each pixel (r, c) is assigned to its corresponding angular bin based on its relative vector from the robot's origin:

$$\theta_{r,c} = \tan^{-1}\left(-\frac{r - r_o}{c - c_o}\right) \quad i = \left\lfloor \frac{\theta_{r,c} - \theta_{\min}}{\Delta\theta} \right\rfloor.$$

For each sector index i, we accumulate pixel statistics (ignoring sky):

$$S_i = \left\{(r, c) \mid i = \lfloor(\theta_{r,c} - \theta_{\min})/\Delta\theta\rfloor\right\},$$

and compute its average cost

$$\bar{C}_i = \frac{1}{|S_i|} \sum_{(r,c)\in S_i} I_{r,c},$$

along with its maximum cost, pixel count, and closest lethal depth, if any. Sectors containing lethal or near-lethal obstacles within a threshold distance d_{lethal} are marked as invalid and excluded from frontier consideration in ACD (Algorithm 8).

Goal-Aware Revalidation If a goal pixel (r_g, c_g) exists within the current field of view, its angular index is computed as

$$i_g = \left\lfloor \frac{\theta_g - \theta_{\min}}{\Delta\theta} \right\rfloor, \quad \theta_g = \tan^{-1}\left(-\frac{r_g - r_o}{c_g - c_o}\right)$$

Algorithm 6 `AngularSectorStat`: Computation of Angular Sector-wise Statistics

Require: Costmap I, depth map D, column threshold map C_{th}, angular range $[\theta_{\min}, \theta_{\max}]$, stride $\Delta\theta$, origin (r_o, c_o), minimum angular stride $\Delta\theta_{\min}$, angular step size $\Delta\theta_{\text{step}}$, max allowable cost $I_{\max}$

Ensure: Sector-wise statistics $\{S_i\}$ including average cost, min lethal depth, validity

Partition the field of view into angular sectors $\{S_i\}$ of stride $\Delta\theta$
for all pixels (r, c) above the origin **do**
    Compute $\theta_{r,c}$ and assign to sector index i
    if $I_{r,c} > I_{\max}$ **and** $D_{r,c} < d_{\text{lethal}}$ **then**
        Mark S_i as invalid and record/update minimal invalid depth for S_i
    end if
    Accumulate count and cost statistics for S_i
end for
Compute goal angle θ_g and index i_g
if S_{i_g} is invalid **and** $D(r_g, c_g) < d^{\min}_{\text{invalid}}(S_{i_g})$ **then**
    Lift invalid flag for goal sector S_{i_g} ▷ (Goal depth < closest lethal depth)
end if
Count total invalid sectors $N_{\text{inv}} \leftarrow |\{i : S_i \text{ invalid}\}|$
if $N_{\text{inv}} = N_\theta$ **then**
    while $\Delta\theta > \Delta\theta_{\min}$ **do**
        $\Delta\theta \leftarrow \Delta\theta - \Delta\theta_{\text{step}}$
        `success`, $\{S_i\} \leftarrow$ AngularSectorStat$(I, D, C_{\text{th}}, \theta_{\min}, \theta_{\max}, \Delta\theta)$
        if `success` **then**
            return `success`, $\{S_i\}$
        end if
    end while
    return `FALSE`, $\{S_i\}$
end if
Normalize accumulated costs $\bar{C}_i$ by the count for all sectors
return `TRUE`, $\{S_i\}$

If the goal sector S_{i_g} is marked invalid but the goal's depth $d_g = D(r_g, c_g)$ is lower than the closest lethal pixel in that sector (i.e., $d_g < d^{\min}_{\text{invalid}}$), the invalid flag is lifted to prevent false rejections when the goal lies in front of nearby but farther obstacles.

This step ensures that the planner does not ignore a reachable goal due to obstacle pixels at greater depths within the same angular sector.

Algorithm 7 LRN-Style Frontier Selection

Require: Costmap I, depth map D, column threshold map C_{th}, angular range $[\theta_{\min}, \theta_{\max}]$, stride $\Delta\theta$, origin (r_o, c_o), goal (r_g, c_g), minimum angular stride $\Delta\theta_{\min}$, angular step size $\Delta\theta_{\text{step}}$, max allowable cost $I_{\max}$, lethal depth threshold d_{lethal}

Ensure: Farthest valid frontier pixel (r_f, c_f)

Angular Sector-wise Computation: (Algorithm 6) Use min lethal depth

success, $\{S_i\} \leftarrow$ AngularSectorStat$(I, D, C_{\text{th}}, \theta_{\min}, \theta_{\max}, \Delta\theta)$

if not success **then**

  return $(-1, -1)$ ▷ Failure

end if

LRN Strategy: Prefers sectors with farthest lethal depth

Compute goal angle θ_g and index i_g

Retrieve goal depth $d_g \leftarrow D(r_g, c_g)$

if goal within the FPV field of view **then**

  Check goal sector S_{i_g} first; if $d^{\text{invalid}}_{\min}(S_{i_g}) > D(r_g, c_g)$, select it.

  Otherwise, expand search symmetrically about angular goal index i_g:

  Alternate left/right search w.r.t. i_g until i is found $d^{\text{invalid}}_{\min}(S_i) > D(r_g, c_g)$

  If both sides valid, choose the one with higher lethal depth (safer) $d^{\text{invalid}}_{\min}(S_i)$.

else

  $i^* \leftarrow \arg\max_i d^{\text{invalid}}_{\min}(S_i)$ ▷ Goal outside FoV — pick safest sector

end if

Frontier Extraction:

Within sector S_{i^*}, find farthest valid pixel:

$$(r_f, c_f) = \arg \max_{(r,c) \in S_{i^*},\, I_{r,c} < C_{\text{th}}(r,c)} \left[(r - r_o)^2 + (c - c_o)^2 \right]$$

return (r_f, c_f)

5.3.2 LRN

Algorithm 7 is inspired by the Long-Range Navigation (LRN) approach [20]. It uses the angular sector statistics to select the farthest valid frontier located within the sector that has the largest minimum lethal depth, thereby preferring directions that appear more open or safer for traversal. When the goal is visible within the field of view, the algorithm first evaluates the goal-aligned sector; if it is invalid, a symmetric search is performed left and right about the goal direction to find the nearest valid sector. If the goal lies outside the field of view, the algorithm directly selects the safest sector based on its lethal depth estimate. In our implementation, the original LRN formulation is adapted to our context, since the original method jointly learns affordance and frontier without using explicit depth information. Here, we proxy affordance using our cost model and incorporate a reliable depth estimate for more robust decision-making.

5.3.3 ACD

Our **Angular Cost & Depth-Aware Frontier (ACD)** algorithm (Algorithm 8) extends the same framework by incorporating both average cost and lethal depth information when selecting a frontier. Unlike LRN, which prioritizes openness (via depth in our implementation), ACD favors sectors with lower average cost while still satisfying a minimum lethal depth threshold. A symmetric search is performed around the goal direction to locate the best valid (i.e. lethal pixels are not too close) sector, and within that sector, the farthest valid pixel below the cost threshold is chosen as the frontier.

In cluttered or densely obstructed environments, openness alone can be misleading, as visually open regions may be difficult to identify due to occlusions and so on. Instead, ACD leverages average cost statistics and lethal depth cues to navigate through such challenging terrain. This formulation aims to generalize better across diverse off-road environments, reducing bias toward open spaces and improving robustness in difficult conditions.

Farthest Valid Frontier Extraction Within a Sector

After determining the optimal sector index i^* using either LRN (Algorithm 7) or ACD (Algorithm 8), the next step is to extract a valid frontier within that sector. The same procedure is applied for both approaches. The frontier pixel is defined as the farthest valid point (i.e., with the maximum radial distance) within the selected angular sector whose cost remains below the collision threshold map C_{th}:

$$(r_f, c_f) = \arg\max_{(r,c)\in S_{i^*}} \left\{ (r - r_o)^2 + (c - c_o)^2 \;\middle|\; I_{r,c} < C_{\text{th}}(r, c) \right\}.$$

This yields the final depth-aware angular frontier, which is passed to the local motion planner as a directional target.

Algorithm 8 ACD: Angular Cost&Depth-Aware Frontier Selection

Require: Costmap I, depth map D, column threshold map C_{th}, angular range $[\theta_{\min}, \theta_{\max}]$, stride $\Delta\theta$, origin (r_o, c_o), optional goal (r_g, c_g), stuck flag b_{stuck}, minimum angular stride $\Delta\theta_{\min}$, angular step size $\Delta\theta_{\text{step}}$, max allowable cost $I_{\max}$

Ensure: Farthest frontier pixel (r_f, c_f)

Angular Sector-wise Computation: (Algorithm 6) Use costs and validity

`success`, $\{S_i\} \leftarrow$ AngularSectorStat$(I, D, C_{\text{th}}, \theta_{\min}, \theta_{\max}, \Delta\theta)$

if `not success` **then**

  return $(-1, -1)$ ▷ Failure

end if

ACD: Prefers sectors with lower cost under a lethal depth threshold

Initialize optimal sector index: $i^* \leftarrow -1$

Check goal sector S_{i_g} first; if valid and $\bar{C}_{i_g}$ is below threshold, select it.

Otherwise, expand search symmetrically about angular goal index i_g:

Alternate left/right search relative to i_g until a valid sector is found

If both sides valid, choose the one with the lower average cost $\bar{C}_i$.

Frontier Extraction:

Within S_{i^*}, find farthest pixel (r, c) satisfying $I_{r,c} < C_{\text{th}}(r, c)$:

$$(r_f, c_f) = \arg\max_{(r,c)\in S_{i_{\text{best}}}} \left[(r - r_o)^2 + (c - c_o)^2 \right]$$

return (r_f, c_f)

Chapter 6

Planning

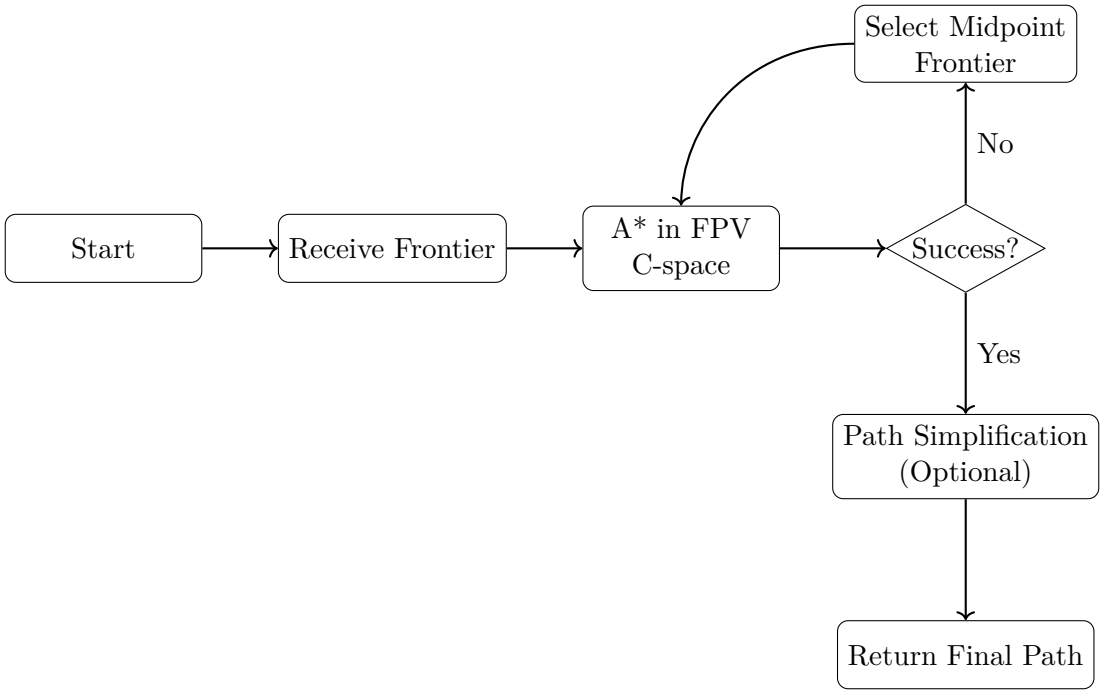

Figure 6.1: **Flowchart of the planning loop:** A* planning towards the selected frontier, optional path simplification upon success, and midpoint-frontier fallback with replanning upon failure.

6.1 A* Implementation

Figure 6.1 summarizes the planning loop used in our FPV C-space planner. Starting from the vehicle's current position (usually near the tip of the vehicle), the system first receives a frontier goal (Section 5.3) and runs A* in the C-space costmap. If the search succeeds, the resulting path may optionally undergo pixel-space simplification before being returned for execution. If the search fails to reach the frontier, the planner selects a midpoint frontier located approximately halfway between the start and the original (or previous) frontier recursively in an attempt to make forward progress.

Below, we highlight three aspects of our A* implementation that we believe are worth mentioning for reproducibility.

6.1.1 Cost Renormalization for Heuristic Validation

Although A* is a classical and widely established search algorithm, its correct application in unconventional domains such as FPV image-space planning is not entirely straightforward. Even small numerical inconsistencies between the heuristic and the cost formulation can silently break its optimality guarantees. In our case, the costmap values were normalized to the range $[0, 1]$, representing terrain traversability. While this scaling is convenient for visualization and neural network outputs, it introduces a subtle but important trap: in free-space regions, the local cost values become so small that the heuristic may overestimate the true path cost. The planner still produces visually plausible paths, but the search ceases to be theoretically optimal. To prevent this, we introduced a constant offset to the costmap before planning:

$$C'(p) = 1 + C(p),$$

ensuring that each step carries at least its geometric traversal cost. Below we provide the theoretical justification for this normalization and its relationship to the admissibility and consistency properties of A*.

Foundations of A*

At each iteration, A* expands the node n with the lowest total estimated cost

$$F(n) = G(n) + H(n),$$

where $G(n)$ is the cumulative cost from the start node to n, and $H(n)$ is the estimated remaining cost to the goal.

For the algorithm to be both optimal, the heuristic H must satisfy two classical properties [9]:

- **Admissibility.** The heuristic never overestimates the true remaining cost:

  $$H(n) \leq J^*(n), \qquad \forall n,$$

  where $J^*(n)$ is the minimal true cost from n to the goal. This ensures that $F(n)$ is a valid lower bound on the total path cost.
- **Consistency (Monotonicity).** For every edge (n, p), the heuristic must satisfy

  $$H(n) \leq c(n, p) + H(p),$$

  ensuring that F-values are non-decreasing along any valid path. Consistency implies admissibility and guarantees that A* never needs to re-expand closed nodes.

Cost model in the image-space planner

Each pixel in the FPV costmap represents a node p, with traversability $C(p) \in [0, 1]$. For our five-connected grid (up, up–{left/right}, left/right), the geometric step lengths are

$$d(n, p) \in \{1, \sqrt{2}\}.$$

The raw transition cost is defined as

$$c(n, p) = d(n, p)\, C(p),$$

and the heuristic is the Euclidean distance to the goal,

$$H(n) = \|p_n - p_g\|_2$$

Failure modes with an unshifted costmap

When $C(p)$ is small in free-space regions, the step costs $c(n, p)$ approach zero. Over long paths, the total accumulated cost $J^*(n)$ can then become far smaller than the Euclidean distance $H(n)$, which is measured in pixel units. This breaks the admissibility condition:

$$H(n) > J^*(n),$$

meaning that the heuristic overestimates the true cost-to-go. The node expansions are no longer guided by valid lower bounds, and A* may terminate with a suboptimal path.

The same issue undermines consistency. For neighboring nodes n and p, the heuristic difference satisfies $H(n) - H(p) \approx d(n, p)$, but the transition cost $c(n, p) = d(n, p)\, C(p)$ may be far smaller. Thus, the required inequality

$$H(n) \leq c(n, p) + H(p)$$

fails, since $c(n, p) \ll d(n, p)$.

Geometric interpretation via the triangle inequality

For Euclidean heuristics, consistency follows naturally from geometry. The triangle inequality states that for any three points (p_n, p_p, p_g),

$$\|p_n - p_g\|_2 \leq \|p_n - p_p\|_2 + \|p_p - p_g\|_2.$$

If the step cost equals or exceeds the geometric distance, i.e. $c(n, p) \geq \|p_n - p_p\|_2 = d(n, p)$, then

$$H(n) \leq d(n, p) + H(p) \leq c(n, p) + H(p),$$

and the heuristic is guaranteed to be consistent. However, when $c(n, p)$ is smaller than the geometric distance, which is unavoidable with $C(p) \in [0, 1]$.

Restoring admissibility and consistency with an additive offset

To maintain the proper geometric relationship between cost and heuristic, we redefine the costmap as

$$C'(p) = 1 + C(p), \quad c'(n,p) = d(n,p)\, C'(p).$$

This guarantees that every move costs at least its Euclidean length:

$$c'(n,p) \geq d(n,p).$$

With this scaling, the true path cost satisfies $J^*(n) \geq \|p_n - p_g\|_2$, preserving admissibility, and the triangle inequality ensures

$$H(n) \leq d(n,p) + H(p) \leq c'(n,p) + H(p),$$

which restores consistency. Under this formulation, F-values are strictly non-decreasing along any path, and A* expansion is optimal. Conceptually, the adjustment ensures that in free-space regions, where $C(p) \approx 0$, the algorithm still perceives a meaningful geometric cost per step.

6.1.2 Dynamic Local Rollouts and Path Partiality

In our C++ implementation, we consider two distinct motion groups to regulate local search behavior and curvature near the vehicle:

- **Dynamic rollouts:** Close to the vehicle, a restricted set of three pixel-level motion primitives {up, up-left, up-right} is used to prevent excessively sharp turns that would violate the vehicle's steering constraints or produce unrealistic lateral motions in image space. These rollouts enforce a smoother initial curvature by biasing the local expansion toward forward-facing directions and limiting the angular deviation between consecutive moves. Beyond a predefined proximal region (typically $20 - 30\%$ of the FPV height), the planner expands using the full set of five motion primitives {up, up-left, up-right, left, right }, allowing broader lateral exploration and global path coverage. This staged design preserves stability near the vehicle while maintaining navigational flexibility farther ahead.
- **Partial path extraction:** When the full A* search cannot reach the projected

goal satisfying a collision threshold, the planner seeks a partial path by selecting a new frontier node that lies approximately halfway between the current exploration frontier and the start position. This midpoint frontier serves as a stable intermediate goal that represents meaningful forward progress.

6.2 Path Simplification in Pixel $\mathcal{C}$-space

The discrete path generated by the A* planner typically contains redundant waypoints and high-frequency zig-zag patterns introduced by grid discretization. Although each waypoint is individually valid, these artifacts lead to unnecessary curvature and instability once the path is projected into 3D space for vehicle execution. To mitigate this, we apply greedy simplification of the A* path.
Algorithm 9 depicts the greedy path simplification procedure applied in the pixel $\mathcal{C}$-space costmap. Starting from the initial waypoint, the algorithm iteratively seeks the farthest subsequent point that can be connected without violating the traversability constraint. For each current point p_i, it tests candidate endpoints p_j in reverse order, from the end of the path toward p_{i+1}, and uses Bresenham's line algorithm to enumerate the discrete pixels along the segment $\text{line}(p_i, p_j)$. If the maximum cost along this line is below the collision threshold $C_{\max}$, i.e.

$$\max_{\mathbf{x}\in\text{line}(p_i,p_j)} \mathcal{C}(\mathbf{x}) < C_{\max},$$

all intermediate waypoints between p_i and p_j are removed and p_j is appended to the simplified path. If no longer valid connection is found, the immediate next point p_{i+1} is retained to ensure forward progress. This greedy backward search preserves only those points that are required for obstacle avoidance, resulting in a more compact path in pixel space.

Algorithm 9 Greedy Path Simplification in Pixel $\mathcal{C}$-space

Require: Raw path $\mathcal{P} = [p_1, p_2, \ldots, p_N]$, costmap $\mathcal{C}$, threshold $C_{\max}$
Ensure: Simplified path $\mathcal{P}'$

$\mathcal{P}' \leftarrow [p_1]$
$i \leftarrow 1$
while $i < N$ **do**
    $j \leftarrow N$
    while $j > i + 1$ **do**
        line $\leftarrow$ Bresenham(p_i, p_j)
        if $\max_{\mathbf{x} \in \text{line}} \mathcal{C}(\mathbf{x}) < C_{\max}$ **then**
            Append p_j to $\mathcal{P}'$
            $i \leftarrow j$
            break
        else
            $j \leftarrow j - 1$
        end if
    end while
    if $j = i + 1$ **then**
        Append p_j to $\mathcal{P}'$
        $i \leftarrow j$
    end if
end while
return $\mathcal{P}'$

Chapter 7

Experiments

In Section 7.1, we examine whether mono–LiDAR calibration reduces depth bias. Section 7.2, compares different frontiering strategies in terms of distance traveled across diverse Falcon simulator environments.

7.1 Test-Time Depth Calibration

In Chapter 4, we calibrated raw monocular predictions with sparse LiDAR using first-order least-squares fits (no network updates). Two variants were evaluated in practice: a direct *affine* fit in metric depth and a *log-affine* fit in the log-depth domain. Qualitative examples for our three Falcon simulation environments, Star Rock Farm (SR-Farm), Trabuco, and Desert, are shown in Figures 7.1, 7.2, and 7.3.

We evaluate mean absolute error (MAE) in meters over three disjoint ranges: $[0, 50]$, $[50, 100]$, and $[100, 200]$ m. Table 7.1 summarizes the numbers. Across all three environments and all range bins, both the affine and log-affine calibrations substantially reduce MAE relative to raw monocular predictions, particularly in the $[50, 200]$ m regime where scale drift is most harmful. The affine fit generally provides the strongest far-range correction, while the log-affine fit offers more uniform improvements across mid-range distances.

Note that the calibration is performed using only sparse simulated LiDAR measurements, not dense ground-truth supervision. The evaluation, however, is carried out

Table 7.1: Depth MAE (± std) in *meters* across ranges and environments

Environment	Method	Mean Absolute Error (MAE) (± std)		
		$[0-50]$ m	$[50-100]$ m	$[100-200]$ m
SR-Farm	Mono depth estimate (MDE)	1.49 (2.88)	20.19 (16.13)	55.16 (28.28)
	MDE + Affine calibration	1.82 (3.13)	10.15 (15.24)	22.61 (29.82)
	MDE + Log-affine calibration	0.73 (1.92)	10.01 (14.29)	30.05 (31.77)
Trabuco	Mono depth estimate (MDE)	2.09 (4.24)	39.15 (15.10)	97.56 (26.89)
	MDE + Affine calibration	2.35 (4.23)	9.19 (12.50)	18.07 (26.80)
	MDE + Log-affine calibration	1.07 (2.52)	16.58 (16.34)	35.66 (43.75)
Desert	Mono depth estimate (MDE)	2.32 (4.38)	46.67 (14.33)	115.49 (26.95)
	MDE + Affine calibration	2.10 (4.22)	11.26 (13.43)	39.12 (38.03)
	MDE + Log-affine calibration	0.97 (2.35)	19.17 (14.89)	55.22 (51.11)

using the dense reference depth maps from depth camera, which are never seen by the calibration procedure. Hence, the observed improvements validate the generalization capability of both calibration variants beyond the sparse calibration samples.

Qualitatively, calibration mitigates the multiplicative bias visible in far-field terrain while maintaining local smoothness useful for C-space inflation. Quantitatively, both affine and log-affine fits yield consistent MAE reductions, with the later offering more uniform improvements and the former providing stronger far-range correction.

In our experiments, we employ log-affine calibration. Although the affine fit achieves the lowest error in $[50, 200]$ m band, the log-affine variant provides lower error in $[0, 50]$ m, which is more critical for real-world execution of long-range image-space plans. This is because near-range pixels correspond to regions the robot will traverse within only a few planning cycles. Thus, errors in this regime directly affect footprint inflation, angular-sector scoring, and the C-space trajectory executed by the local planner. By contrast, far-range pixels primarily serve as coarse directional guidance. Any depth at these distances is repeatedly re-estimated and re-planned as the robot progresses. Hence, the operational importance of reduced near-range error outweighs the far-range MAE reduction offered by the affine model. Technically, the log-domain regression places approximately uniform weight across depth scales.

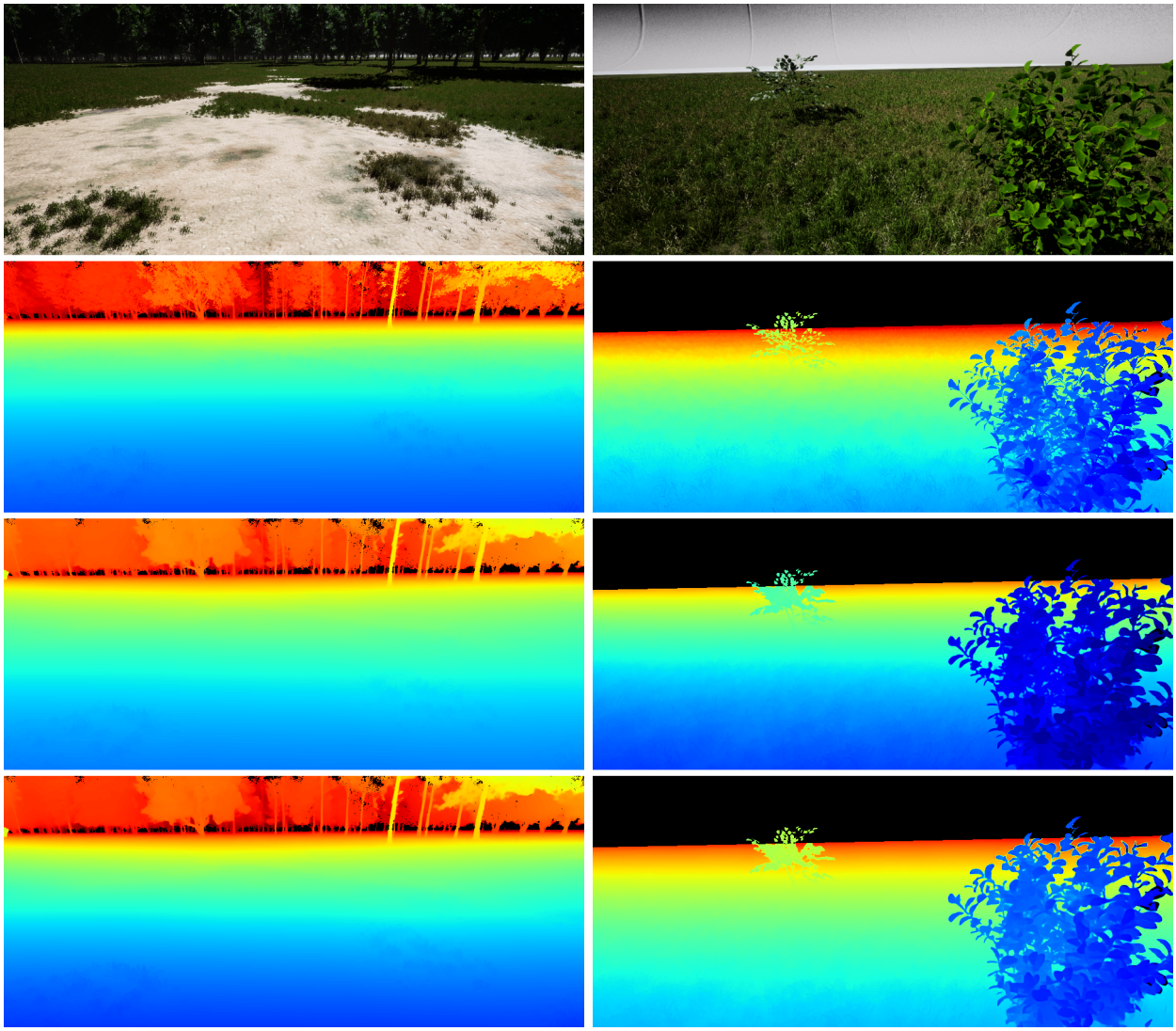

Figure 7.1: **(SR-Farm environment)** Qualitative comparison of depth modalities: (top) RGB image, (second) dense simulator depth, (third) monocular depth prediction, and (bottom) LiDAR-calibrated (log-affine) monocular depth. Calibration mitigates mid/far-range bias while preserving spatial smoothness. All depth values are normalized to the $[0, 200]m$ range and visualized using a `jet-log` colormap.

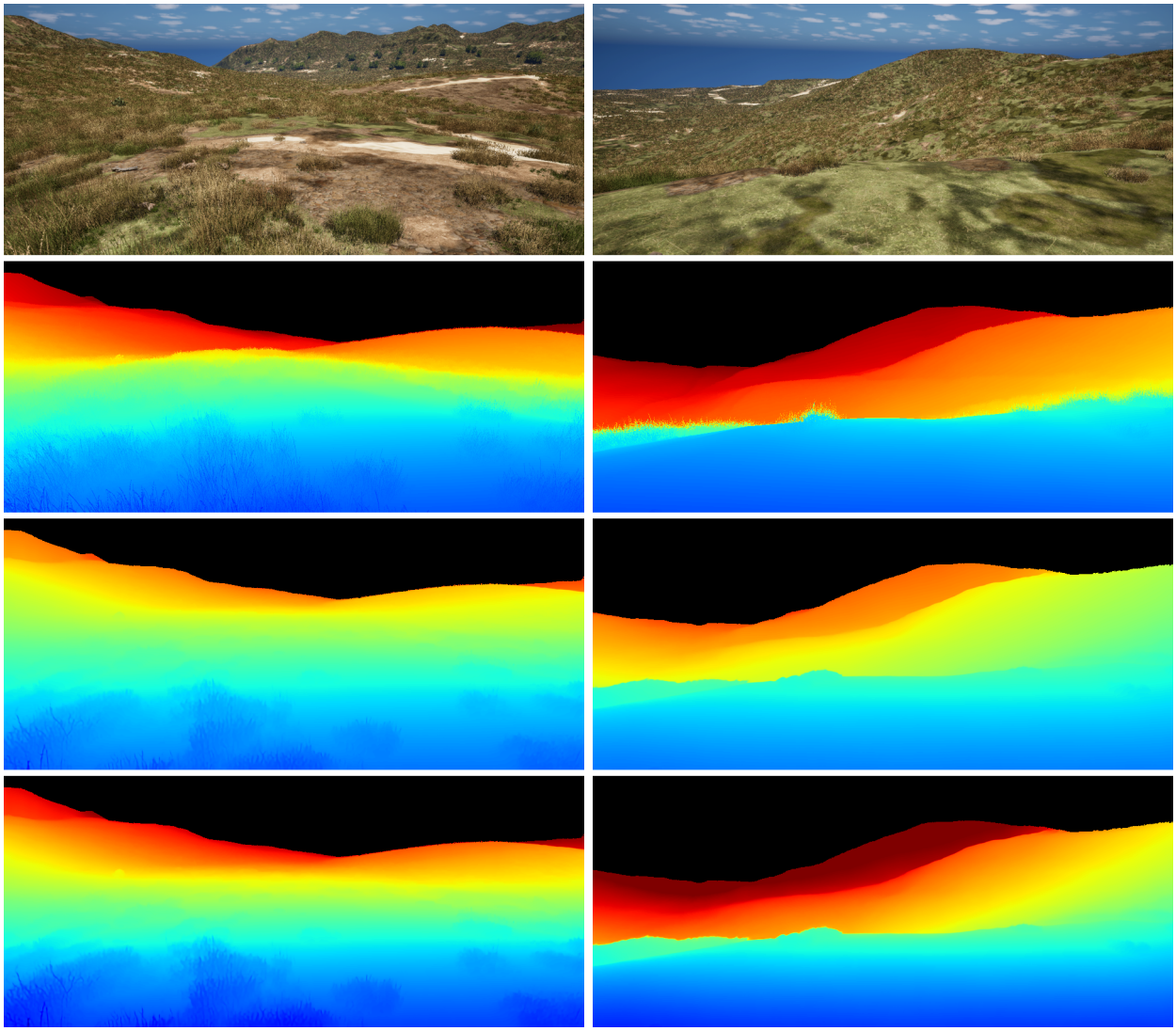

Figure 7.2: **(Trabuco environment)** Qualitative comparison of depth modalities: (top) RGB image, (second) dense simulator depth, (third) monocular depth prediction, and (bottom) LiDAR-calibrated (log-affine) monocular depth. Calibration improves scale and reduces slope-dependent error. All depth values are normalized to the $[0, 200]m$ range and visualized using a `jet-log` colormap.

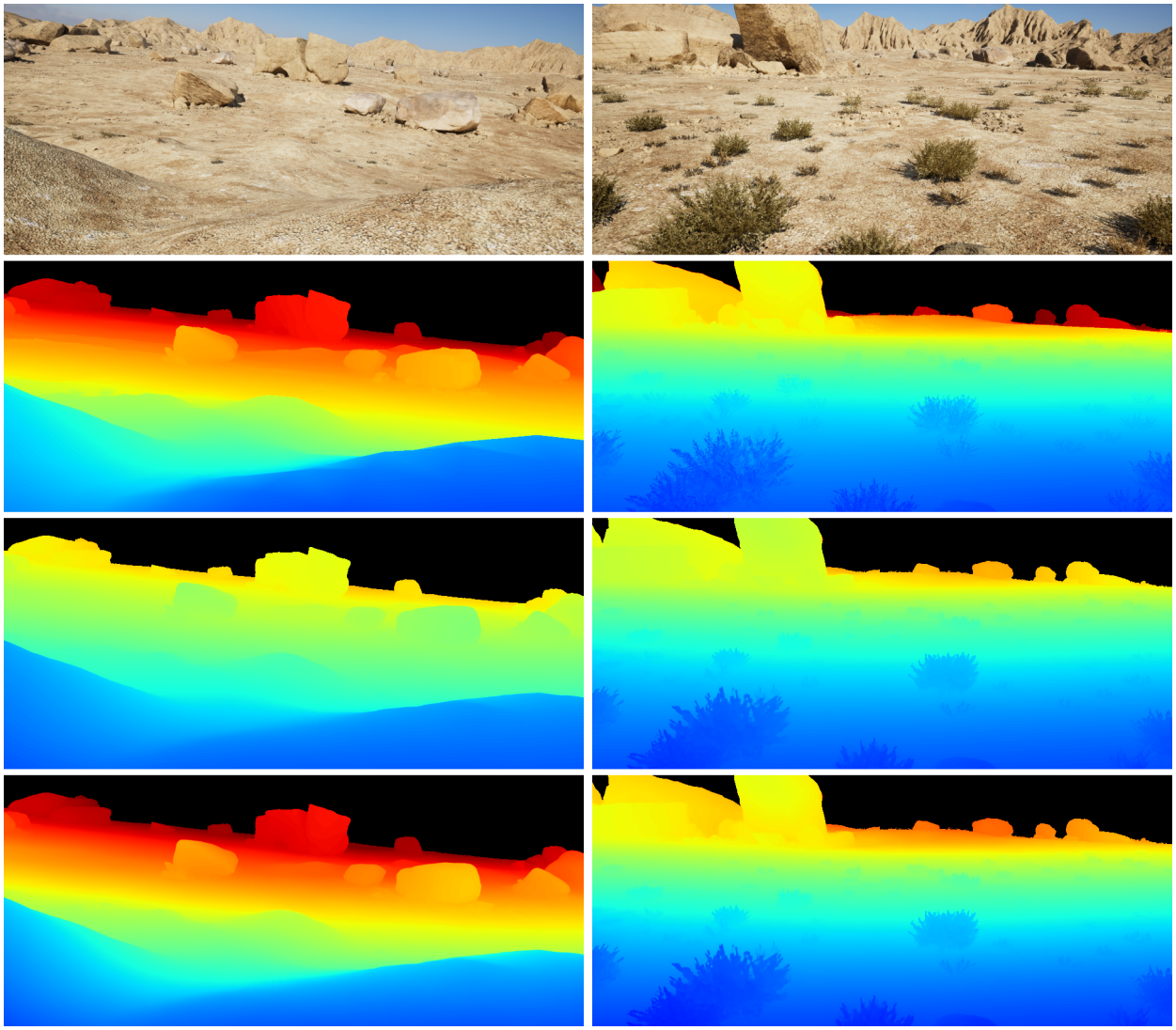

Figure 7.3: **(Desert environment)** Qualitative comparison of depth modalities: (top) RGB image, (second) dense simulator depth, (third) monocular depth prediction, and (bottom) LiDAR-calibrated (log-affine) monocular depth. Large far-field bias in raw monocular depth is significantly corrected. All depth values are normalized to the $[0, 200]m$ range and visualized using a `jet-log` colormap.

7.2 Navigation

7.2.1 Frontiering Strategies in a Simple Simulation

Before proceeding to full end-to-end autonomous simulation with other modules in the high-fidelity Falcon simulator, we first conduct a comparative evaluation of long-range frontiering strategies using a simplified 2D global costmap simulation (Figure 7.4). This setup enables direct visualization of frontier selection and exploration patterns, which is nearly impossible in Falcon due to the interaction of perception noise, local-planner dynamics, and vehicle motion. By isolating frontiering behavior, the effect of the selection strategy can be examined independently of interactions with cost prediction, depth estimation, and local planning, each of which can significantly influence overall performance.

While these results provide useful insights into the intrinsic tendencies of each strategy, they should be considered indicative rather than definitive. In integrated systems, sensor noise, planning dynamics, and module interactions can obscure the individual contribution of frontiering decisions. The simplified simulation thus serves as a preliminary, controlled baseline that highlights the strengths and limitations of each strategy before progressing to more realistic (simulation) environments and fully autonomous evaluations.

Simple Simulation Workflow

Oriented local crop extraction At each iteration, the robot extracts an oriented crop of the global costmap based on its current position and heading. Although the real system reasons in FPV, the BEV crop serves as a convenient surrogate. It preserves the finite forward visibility and directional bias of long-range planning while allowing precise visual inspection of frontier choices. From this cropped region we compute a distance transform, which operates as a geometric proxy for depth. Pixels with large distances indicate extended free-space corridors, whereas shallow regions emulate the effect of low predicted depth in FPV. Sample local oriented cropping results are shown in Figure 7.5. Full 2D simulation results are available. [1]

[1] See: 2D simulation results.

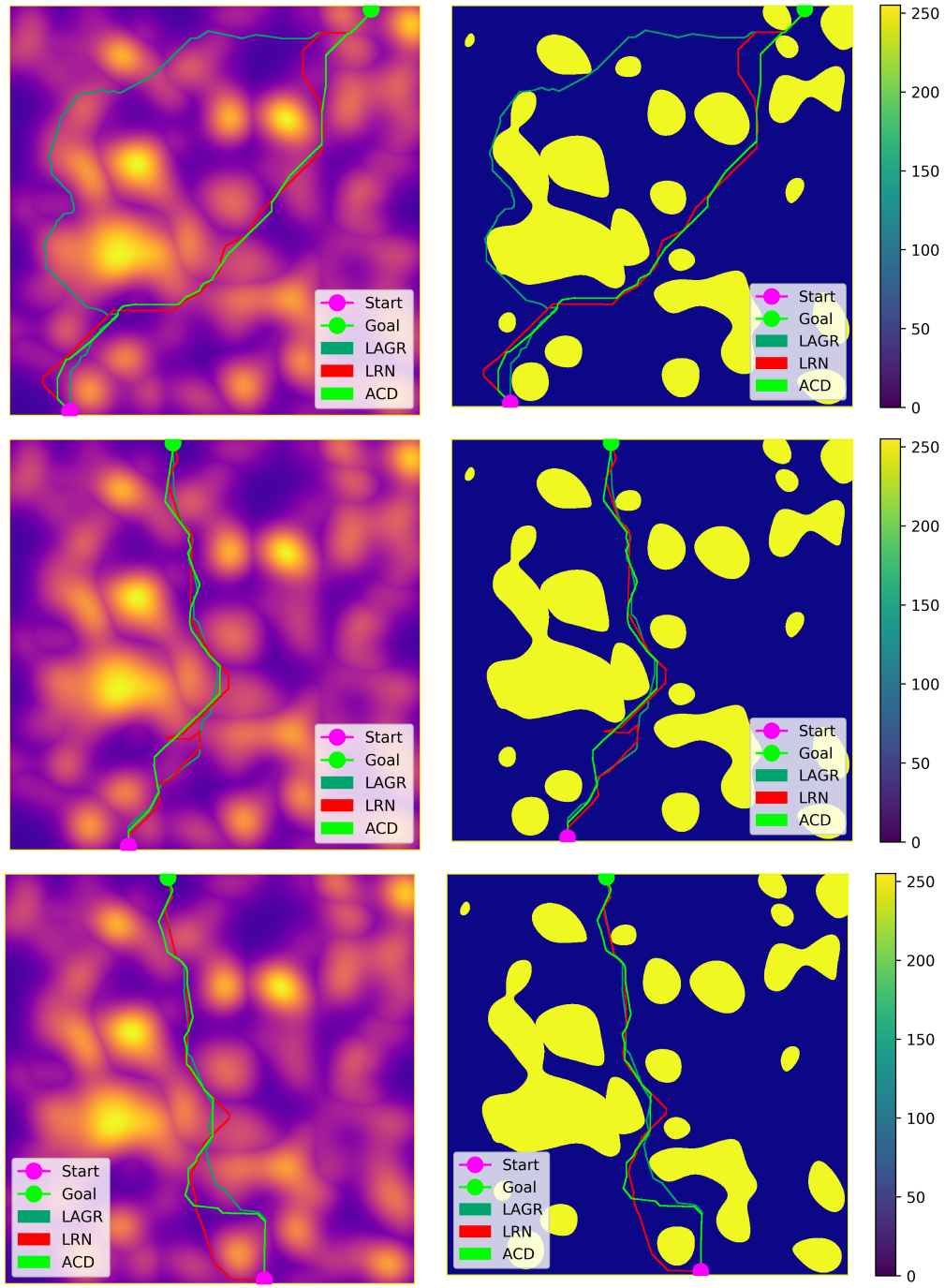

Figure 7.4: Frontiering strategies in the simplified 2D global-costmap simulation. Each row corresponds to a different (start, goal) pair (Row-1/Row-2/Row-3 in Table 7.2). For each setup, the left pane shows the costmap with planned trajectories and the right pane shows the corresponding obstacle map with the collision threshold of 0.5. Comparing three strategies illustrates how row-wise search (LAGR), openness-based angular selection (LRN), and cost-and-depth coupling (ACD) lead to distinct long-range exploration patterns.

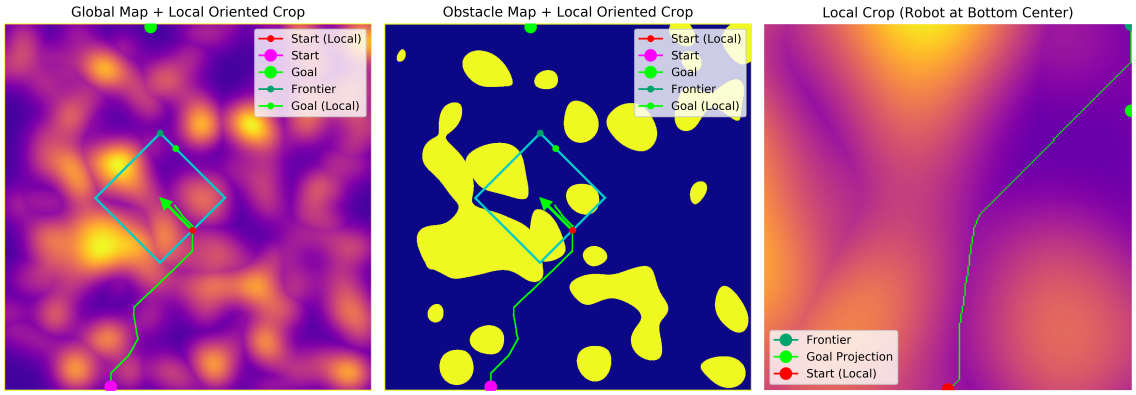

Figure 7.5: (Left: global costmap, middle: global obstacle map, right: local map). Example of local oriented cropping, goal projection, and planning in the simple 2D simulation. Here we show a single ACD rollout corresponding to the middle row of Figure 7.4. The oriented crop, projected goal, selected frontier, and resulting A* path together illustrate one iteration of the cropping → frontiering → planning loop used in the global simulation.

Global goal projection The global goal is then projected into this cropped frame using an SE(2) transformation, giving a goal pixel that anchors the frontier search.

Frontier selection Using the cropped costmap and its distance transform, we evaluate the same three frontiering strategies later used in Falcon:

- **LAGR:** It favors direct progression toward the goal row using a row-wise search starting from the goal row. As shown in the first row of Figure 7.4, for larger obstacles this row-wise search tends to kiss the obstacle walls until it finds an opening toward the goal direction.
- **LRN:** The key idea here is a lethal-free openness heuristic, which partitions the crop into angular sectors and selects the sector with the farthest depth-valid reach. In low-density scenes this tends to identify wide forward corridors, though it can overcommit in cluttered regions where openness changes sharply, as shown in the middle and bottom rows of Figure 7.4.
- **ACD:** It incorporates both sector cost smoothness and a minimal lethal-depth criterion. This coupling mitigates the frontiering fluctuations of LRN, particularly in cluttered scenarios (Figure 7.4).

In this simple setting, ACD generally achieves either the shortest or near-shortest

Table 7.2: Frontiering results for the 2D global-costmap simulation. For each (start, goal) pair (Row-1/Row-2/Row-3, matching Figure 7.4), we report the normalized path length and total accumulated cost achieved by each strategy.

Course	Metric	LAGR	LRN	ACD
Row-1 (Figure 7.4)	Path Length†	2.66	2.26	2.02
	Total Cost†	0.46	0.34	0.33
Row-2 (Figure 7.4)	Path Length	2.07	2.37	1.99
	Total Cost	0.28	0.36	0.32
Row-3 (Figure 7.4)	Path Length	1.91	2.10	2.23
	Total Cost	0.34	0.32	0.30

† Normalized by the straight-line distance from start to goal.

normalized path length while also maintaining the lowest or near-lowest accumulated cost, especially in the first two layouts. By contrast, LAGR often incurs longer, wall-hugging paths around large obstacles, and LRN can produce overextended detours in cluttered regions when the openness heuristic overestimates a particular sector.

Planning After selecting a frontier pixel, we run A* from the robot's position to that frontier within the cropped map. If a complete collision-free path exists, it is used; if not, the planner falls back to the partial-path procedure described in Chapter 6, selecting an intermediate frontier to ensure forward progress.

Execution A percentage (here one-third) of the resulting A* path is executed before replanning. This reflects the long-range behavior of the FPV pipeline, i.e., farther segments of the path are inherently less reliable due to limited visibility and resolution uncertainty. In BEV simulation, a similar phenomenon occurs because the crop has fixed extent. A promising frontier may lead into an unseen obstacle just beyond the crop boundary. Executing only a fraction of the planned path avoids unrealistic overcommitment while capturing the long-horizon nature of the process.

Repeat At the end of each executed segment, the robot is moved to the terminal point of the executed path, its heading is updated, and a new oriented crop is extracted. The sequence of cropping, goal projection, frontiering, A* planning, and execution then repeats until the robot either reaches the goal or gets trapped.

7.2.2 Falcon Simulation Results

Table 7.3: Course definitions in the Falcon simulator. For each course and waypoint, we provide the UTM coordinates along with the straight-line segment length from the previous waypoint (rounded). The exact waypoint values are included to ensure reproducibility in Falcon.

Course	Waypoints			
	0	1	2	3
SR-Farm-0	95.56,-122.46,0/155†	80.89,-274.89,0/153†	60.71,-380.61,0/108	-402.79,-38.02,0/576
Trabuco-0	58.08,71.27,27/92	2.09,70.02,8/56	-136.70,94.80,-1/141	-120.48,-41.0,-6/137
Desert-0	18.45,51.99,0/55	31.59,137.45,0/86	28.19,283.6,0/146	160.19,247.6,0/137

† Straight-line distance (rounded) from the previous waypoint or initial position (0, 0, 0) for waypoint 0.

We evaluate in three Falcon simulator environments with predefined waypoint courses defined in a UTM coordinate frame, centered relative to a known origin. Table 7.3 lists the course waypoints and straight-line separations from the previous waypoint or initial position for waypoint 0. Each waypoint is specified by an index and its (North, East, Altitude) position in meters, which fixes a common geometric reference for comparing frontiering strategies in each environment.

Evaluation protocol We compare three frontier strategies, all using the same perception and planner back-end:

1. **LAGR (row-wise):** Row scanning toward the goal row (Algorithm 5).
2. **LRN-style (depth-aware openness):** Angular sectoring and selection by the farthest lethal-free sector. We proxy affordance with the explicit costmap and inject calibrated depth (Algorithm 7).
3. **ACD:** Angular Cost & Depth-aware lowest average cost sector subject to a minimal lethal-depth constraint (Algorithm 8).

For all methods we (i) use the same FPV C-space (separable row- and column-wise inflation), and (ii) apply planner cost renormalization $C'(p) = 1 + C(p)$ to preserve A* heuristic consistency in pixel space (Chapter 6).

A run is considered successful at a waypoint if the vehicle reaches within the goal

radius of the 3D goal without collision. We report:

- **Distance traveled to goal (m)** per waypoint, or † when the goal is not reached; in that case we also report percentage of the course completed with respect to straight-line distance to the goal (rounded, ignoring collision), as shown in Table 7.4.
- **Qualitative trajectories** for each environment in Figures 7.6, 7.7, and 7.8, along with deviation and intervention views in Figures 7.9–7.16.

Table 7.4: Results for runs across environments. For each course and method, we report the distance traveled to each waypoint. Red entries with † indicate failures, along with the percentage of the course completed.

Course	Method	Distance Traveled (meters) to Reach the Waypoint			
		0	1	2	3
SR-Farm-0	LAGR	150.09	184.15	79.59 (31 %)†	-
	LRN	151.24	163.26	105.52	513.28 (83 %)†
	ACD	139.10	155.80	110.58	510.34 (83 %)†
Trabuco-0	LAGR	310.26	52.28	143.88	126.14 (49%)†
	LRN	83.64	106.27	204.48	323.06
	ACD	75.33	70.75	141.92	303.43
Desert-0	LAGR	76.62	68.72	69.96 (30%)†	-
	LRN	44.09	90.63	90.39 (39%)†	-
	ACD	43.49	91.96	157.89	115.97

† Distance traveled toward but failing to reach the goal.
† Percentage shows the course completed relative to the straight-line distance (ignoring collision) to the goal; successful runs (black entries) implicitly achieve 100%.

Table 7.4 summarizes the distance-based performance across environments. Together with the trajectory plots and deviation/intervention views, it reveals environment-specific failure modes for each strategy.

Following are our observations across the three Falcon environments, which generally echo the tendencies already seen in the simple 2D simulation.

- **SR-Farm:** This environment contains long, uneven stretches with intermittent vegetation and partially occluded obstacles. LAGR frequently follows near-row alignments and therefore tends to clip obstacle boundaries, leading to early terminations. LRN improves overall progress but occasionally overextends into openings that later converge into tight corridors, reflecting its sensitivity to local changes in apparent openness. ACD produces comparable or slightly more stable progress by downweighting narrow, high-cost sectors while still exploiting depth-valid forward corridors. These patterns are visible in the full-run trajectories in Figure 7.6 and in the SR-Farm deviation and intervention views (Figures 7.9–7.12).
- **Trabuco:** The Trabuco course contains rolling elevation changes and mixed open grassland, leading to fluctuating costmap structure. LAGR's row-wise progression again results in large deviations as it tends to push forward in directions where visibility and terrain constraints are poorly resolved. LRN can enter wide but ultimately inefficient corridors when local openness momentarily increases. ACD generally maintains more consistent headings and obtains the lowest or near-lowest travel distances across waypoints, suggesting that cost averaging and depth gating help avoid misleading shallow sectors. These trends are reflected in the trajectories and intervention points in Figures 7.7, 7.13, 7.14, and 7.15.
- **Desert:** This course consists of broad open areas punctuated by medium-sized rocks whose appearance is often visually similar to free space. The full-run and intervention views in Figures 7.8 and 7.16 illustrate how row-wise search (LAGR) and explicit openness (LRN) alone are more easily misled by costmap fluctuations, whereas cost averaging (ACD) has a little stabilizing effect.

In general, the trajectory plots (Figures 7.6, 7.7, and 7.8) qualitatively reflect these trends: LAGR (magenta) tends to sweep rows and zig-zag near lethal regions under limited visibility, LRN (blue) tracks open headings but may select low-cost / low-safety wedges in dense or ambiguous terrain, and ACD (orange) favors low-cost sectors that also pass a depth threshold, yielding a somewhat more stable long-range behavior.

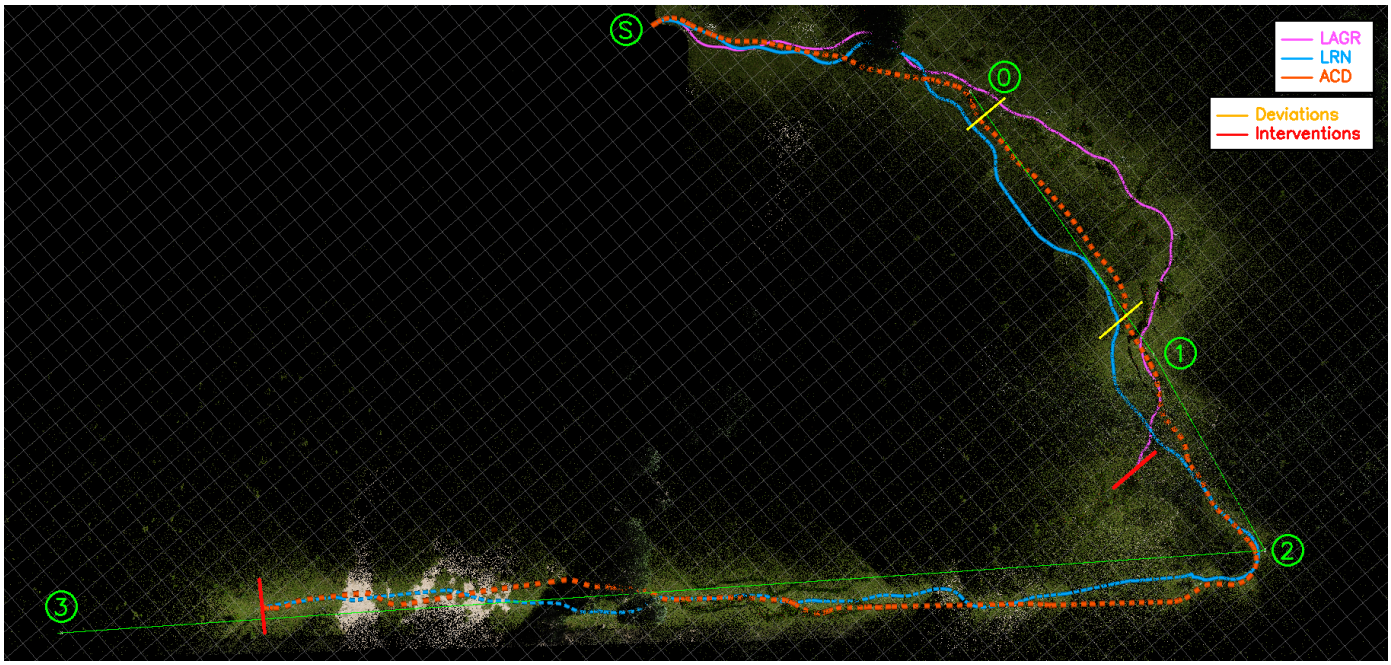

Figure 7.6: **SR-Farm environment.** Full-run trajectories for the three strategies: LAGR (magenta), LRN (blue), and ACD (orange). The green straight lines connect the course waypoints, and the green endpoints (start (s), waypoints 0–3) mark their positions in the global map. These waypoint connections represent the ideal straight-line paths between waypoints without considering collision or terrain obstacles. Deviations introduced by the frontiering strategies (not the local planner) are highlighted in yellow (see also Figures 7.9 and 7.10), while the final intervention locations for each strategy are shown in red (also Figures 7.11 and 7.12).

Failure Analysis

Across environments, we observe that nearly all failures (to reach the goal – which we call interventions) arise from perception-level limitations or local planner issues rather than from the global frontiering logic itself. In SR-Farm, the most common cause is the perception module failing to detect small trees, plants or thin vegetation, so that costmaps underestimate the true obstacle extent (Figure 7.11, 7.12). In the Desert course, medium-sized rocks often receive insufficiently high cost due to their visual similarity to traversable terrain, leading frontiering and planning to treat them as free space until a late collision (Figure 7.16). In Trabuco, open grasslands with uneven elevation can induce cost imbalance, making some slopes appear artificially

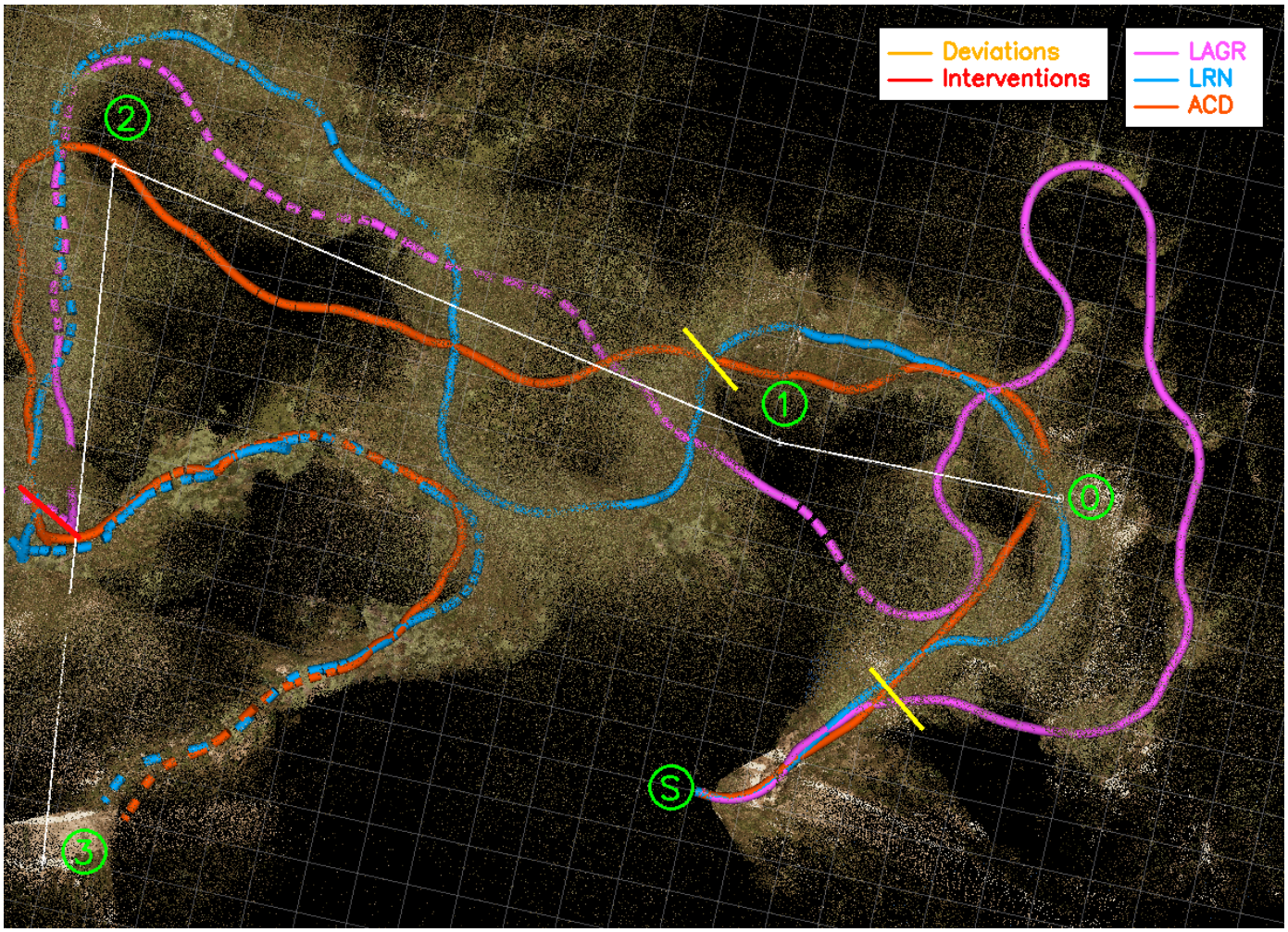

Figure 7.7: **Trabuco environment.** Full-run trajectories for three strategies: LAGR (magenta), LRN (blue), and ACD (orange). The white straight lines connect the designated waypoints, and the green endpoints (start (s), waypoints 0–3) mark their positions in the global map. These straight connections indicate the nominal geometric path between waypoints, ignoring collisions and terrain constraints. Deviations introduced by the frontiering strategies (not the local planner) are highlighted in yellow (see also Figures 7.13 and 7.14), while the final interventions for each strategy are shown in red (also Figure 7.15).

attractive or unattractive (Figure 7.15).

We hypothesize that these errors primarily stem from two factors: (1) the inherent sim-to-real-like gap between visual appearance and geometric traversability in the Falcon assets, and (2) the lack of representative samples for such terrain struc-

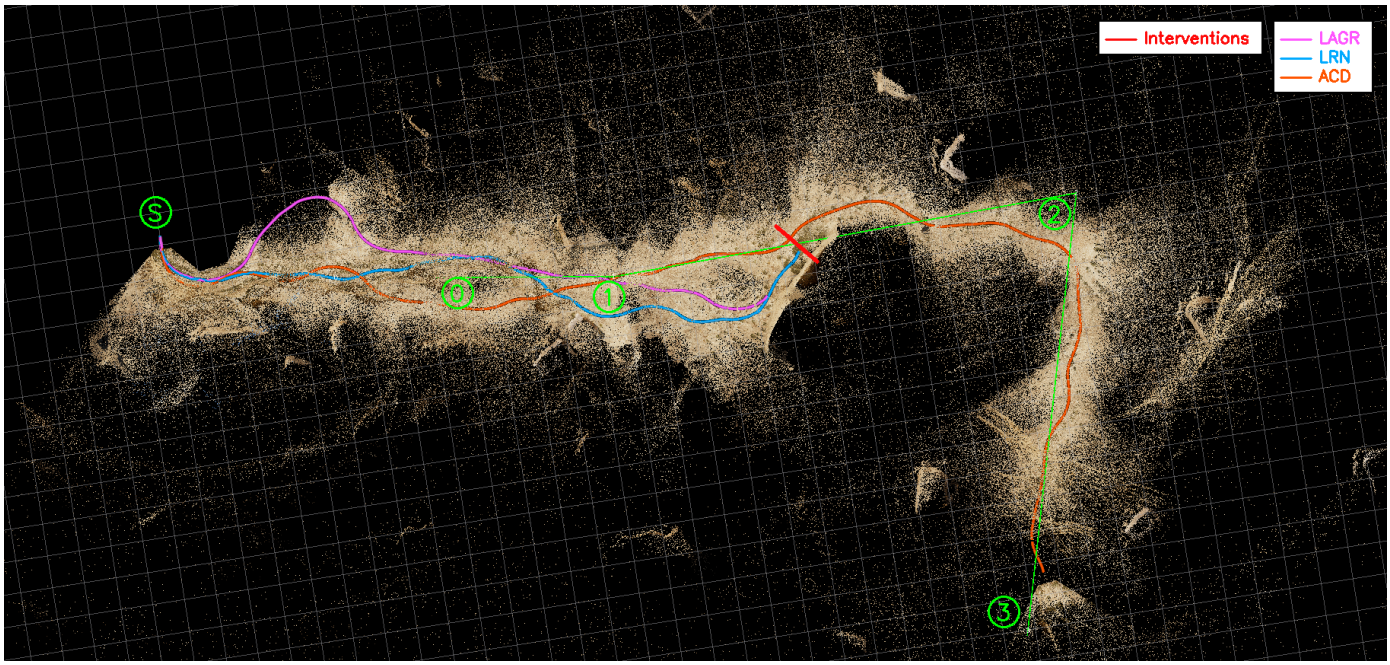

Figure 7.8: **Desert environment.** Full-run trajectories for three strategies: LAGR (magenta), LRN (blue), and ACD (orange). The green lines connect the designated waypoints, and the green endpoints (start (s), waypoints 0–3) mark the waypoint locations in the global map. These lines illustrate the nominal straight-line connections between waypoints, drawn without considering collision or terrain elevation. The final interventions for each strategy are shown in red (also Figure 7.16).

tures in the training distribution of current foundation models. In some cases, the naive primitive-based local planner from the FieldAI stack also contributes to detours (Figure 7.9) even though the global frontiering strategy successfully identifies collision-free long-range paths.

Overall, these findings suggest that improving the non-planning modules—particularly perception and local planner components (both plug-and-play in our pipeline, Figure 3.1) will directly enhance overall navigation robustness and success rate. The full-run videos in ROS and Falcon are available for further reference. [2]

Runtime and implementation notes All results are generated within the same ROS graph (Figure 3.1), with identical perception and planning nodes for all fron-

[2]See: `Videos`.

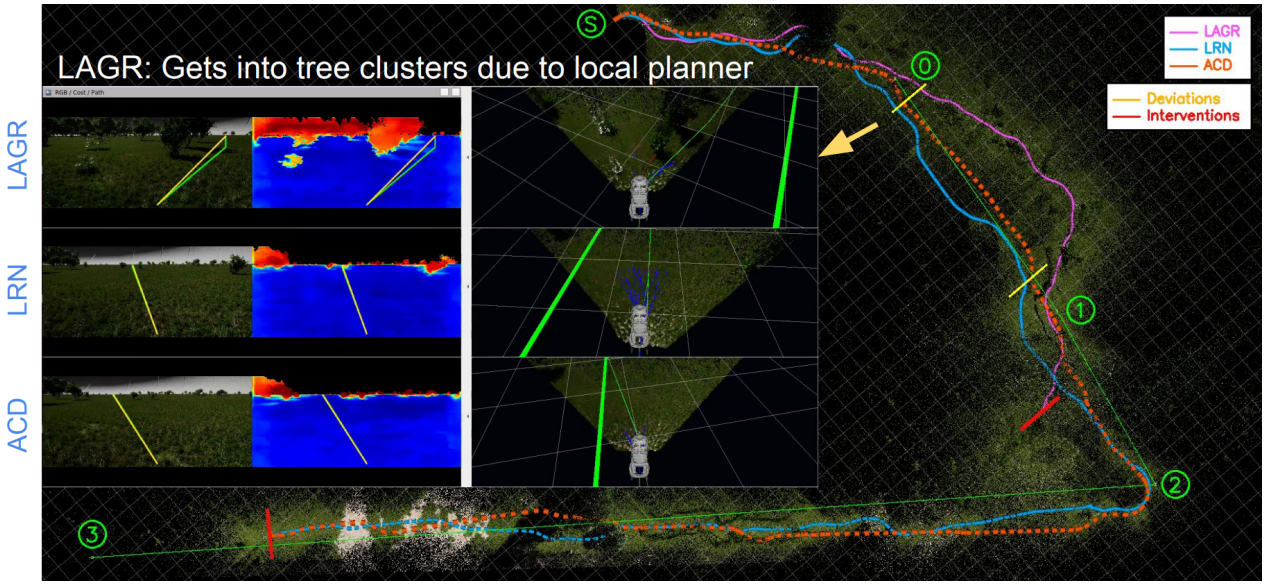

Figure 7.9: **SR-Farm – LAGR deviation.** Example of LAGR deviating into clusters of trees as a consequence of its row-wise forward-search behavior. The frontiering strategy steers the vehicle along obstacle boundaries, and in combination with the local planner this eventually drives the robot into dense vegetation.

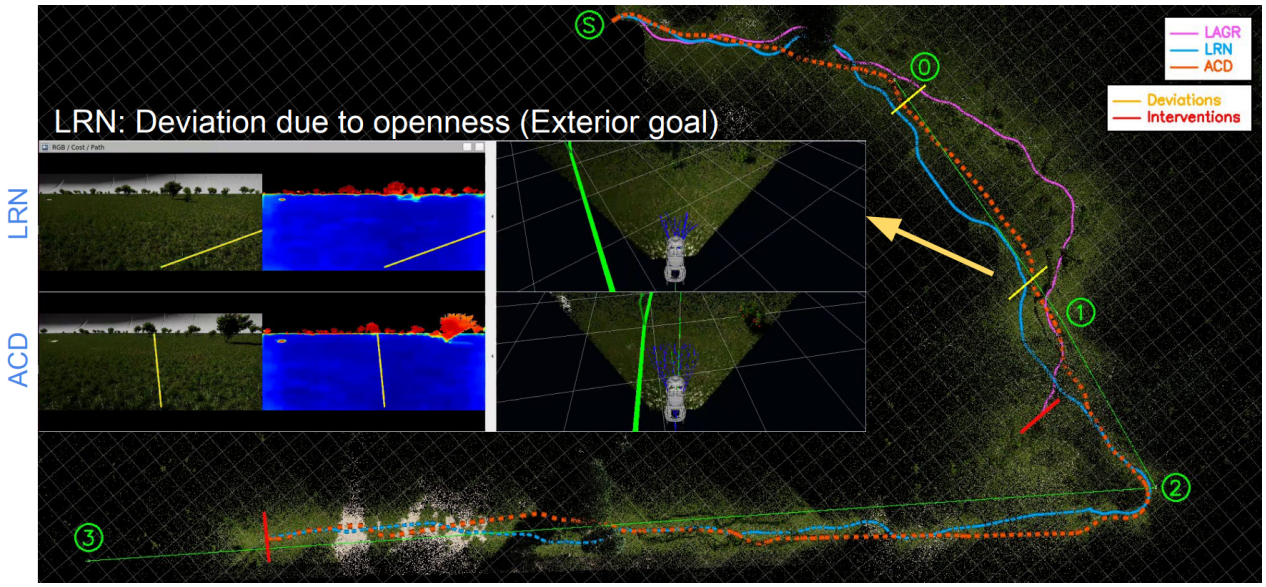

Figure 7.10: **SR-Farm – LRN deviation.** Example of LRN deviating due to the openness heuristic. The strategy selects an apparently open angular sector that later narrows, illustrating how lethal-free reach alone can be misleading in dense vegetation.

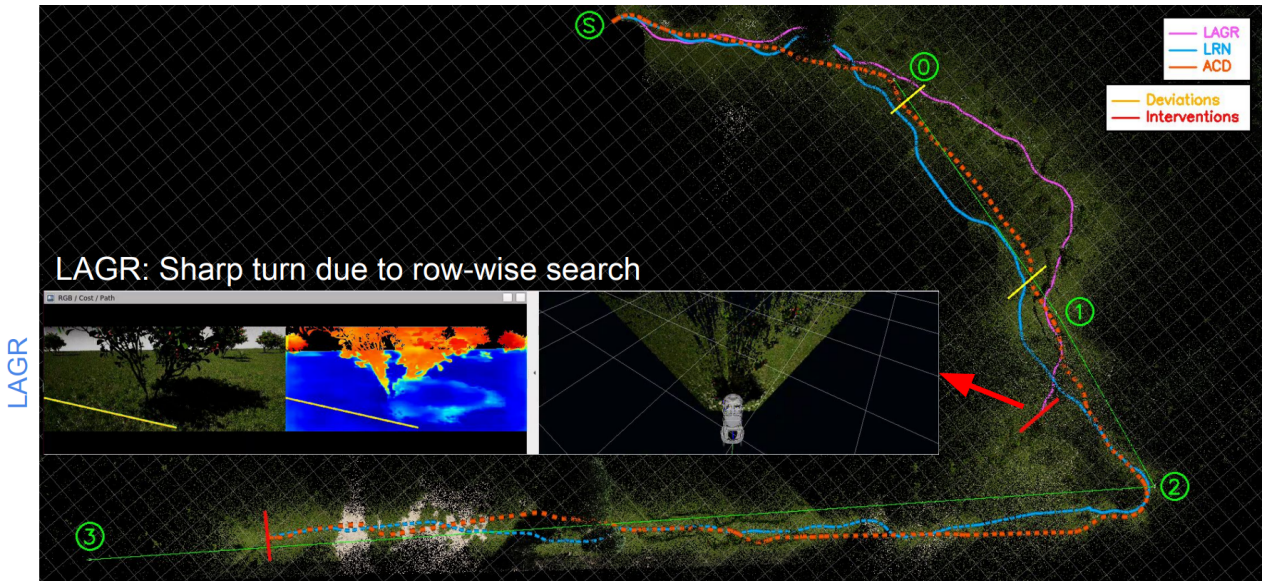

Figure 7.11: **SR-Farm – LAGR intervention.** Row-wise frontiering drives the vehicle into sharp turns around obstacles, causing the controller to saturate and ultimately requiring intervention.

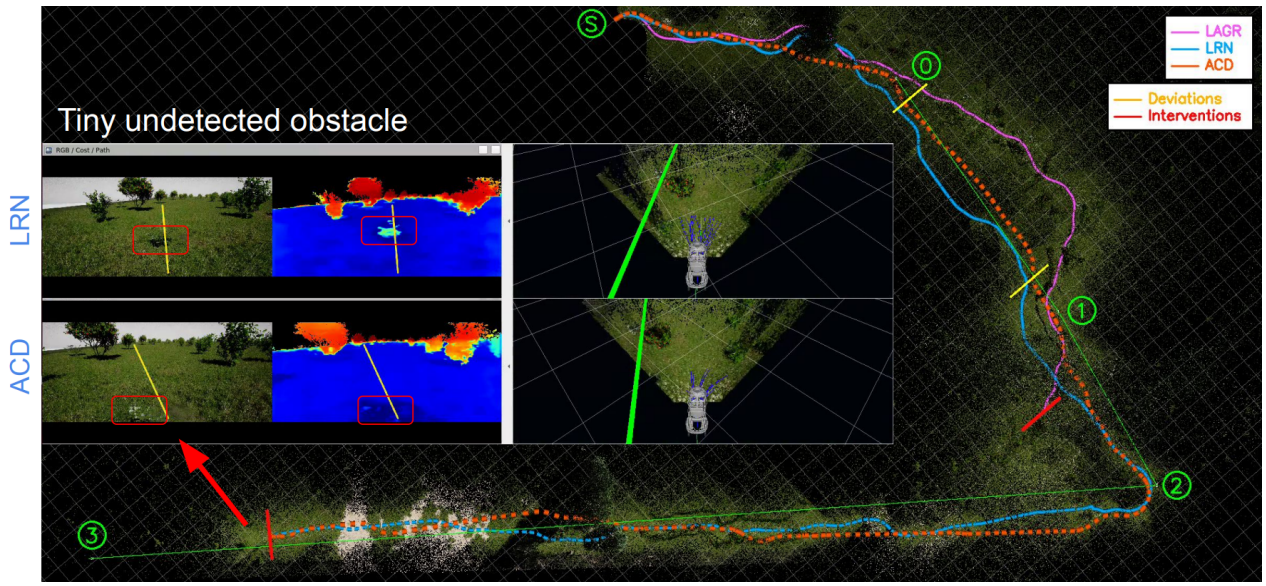

Figure 7.12: **SR-Farm – LRN and ACD interventions.** Both LRN and ACD coincidentally fail on tiny undetected obstacles that are either underweighted in the costmap by the perception modules, reflecting sim-to-real and distributional gaps.

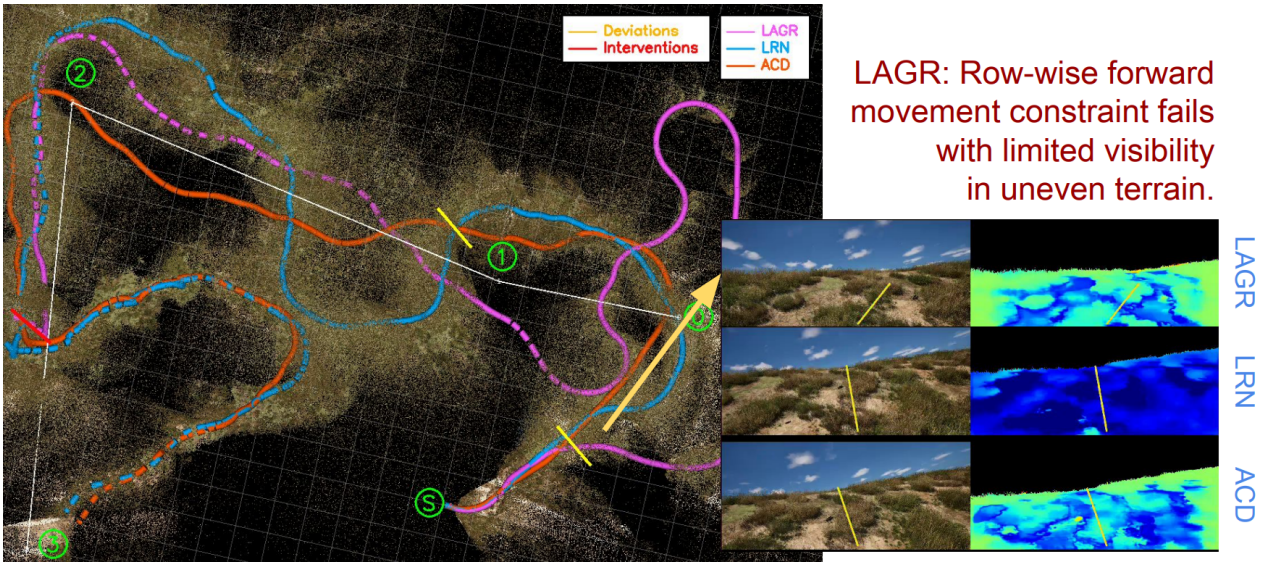

Figure 7.13: **Trabuco – LAGR deviation.** The row-wise forward movement constraint of LAGR is insufficient under limited visibility and slope variation, leading the robot toward regions that are geometrically or dynamically unfavorable.

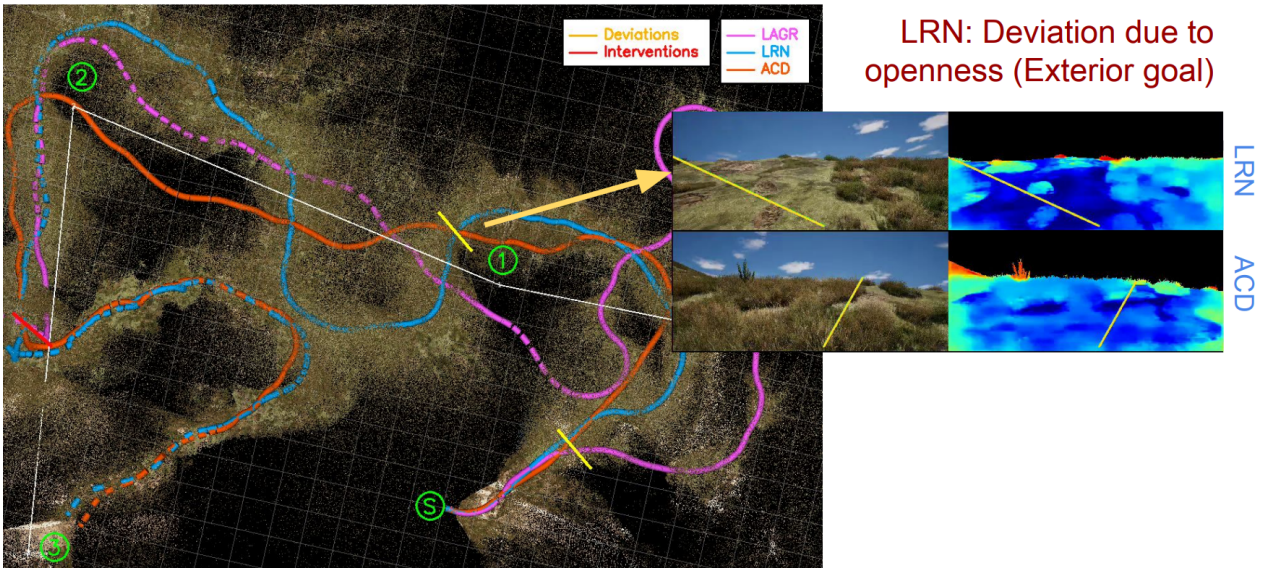

Figure 7.14: **Trabuco – LRN deviation.** The selected angular sector based on openness heuristic appears promising locally but induces a large detour once the terrain and visibility change.

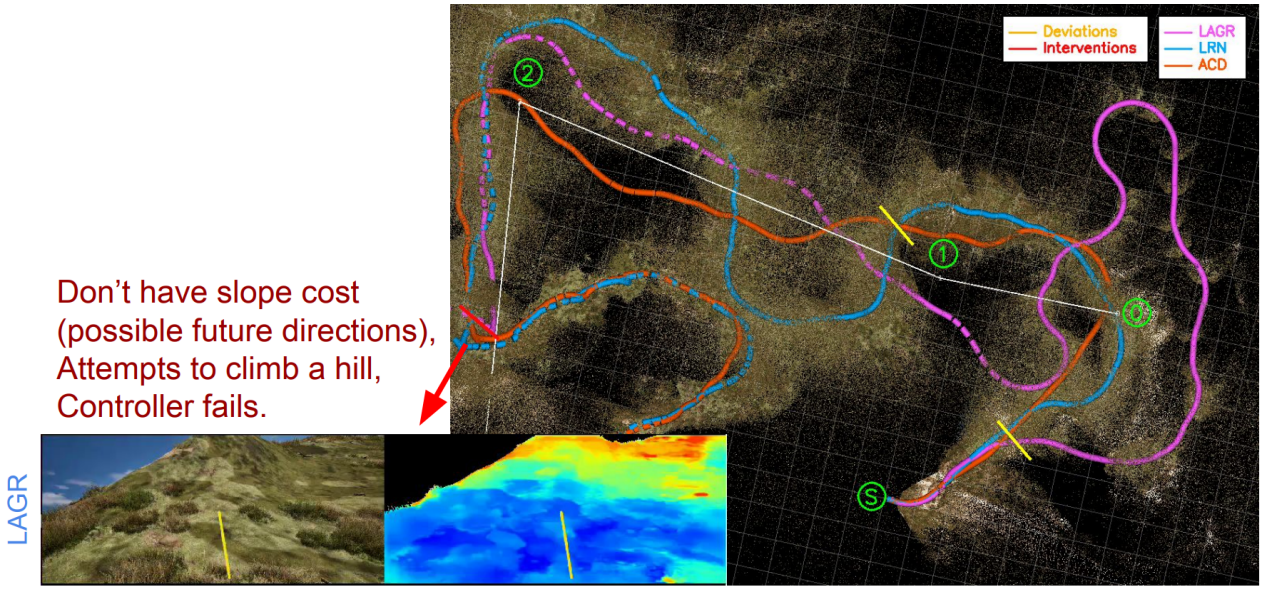

Figure 7.15: **Trabuco – LAGR intervention.** LAGR suggests an attempt to climb a steep hill due to the lack of consideration of the terrain slope in our costmap prediction module – a possible future work.

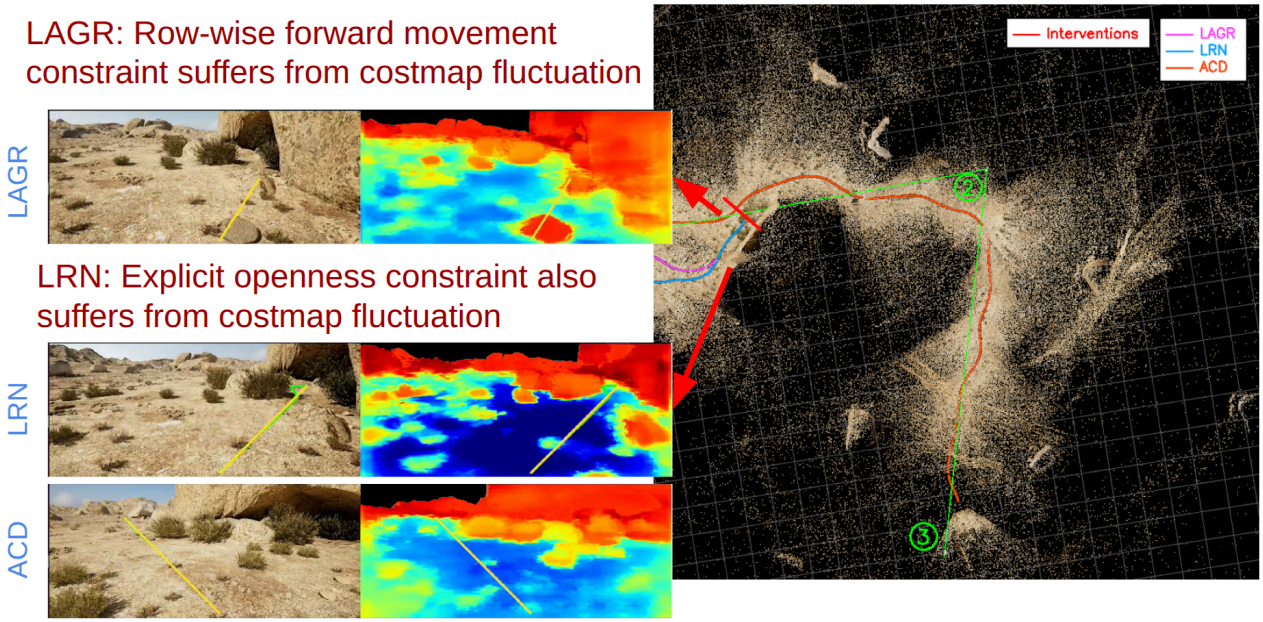

Figure 7.16: **Desert – LAGR and LRN interventions.** Both LAGR and LRN are affected by costmap fluctuations induced by medium-sized rocks whose visual appearance closely resembles traversable terrain. ACD's cost averaging (and depth gating – not shown) mitigate this issue to some extent.

tier strategies. The LiDAR mask synthesis used to emulate sparse beams follows Appendix B. End-to-end planning operates in real time on our workstation; deque-based separable C-space filtering (row/column) is $O(HWk)$ for $k \ll H$, and A* expands over the same grid for all methods.

Chapter 8

Conclusion and Future Work

Summary

We present a modular, image-space approach to long-range navigation in unstructured, off-road environments. We combined (i) a lightweight test-time calibration that makes monocular depth metrically usable with only sparse LiDAR, (ii) an analytic, depth-aware configuration-space (C-space) transform implemented as efficient separable row/column sliding-max filters with depth consistency checks, (iii) a set of interpretable frontier selection strategies in angular sectors that reasons jointly about cost and depth, and (iv) an A* planner in FPV pixel space with a simple cost renormalization to preserve heuristic consistency. Together, these components provide a practical alternative to fully depth-free affordance methods and to heavy-weight mapping, while retaining the constant-size state benefits of image-space planning.

Test-time depth calibration Across three Falcon simulator environments (SR-Farm, Trabuco, Desert), an affine log-domain calibration of monocular predictions with sparse LiDAR reduced depth MAE in all tested distance bins, with the largest gains in the 50–200 m range where raw monocular scale drift is most harmful (Table 7.1). This improvement is obtained without any network updates and is validated against dense reference depth that is never used during calibration.

Navigation with angular frontiers Using a shared FPV C-space and planner, we evaluated three frontier-selection strategies: (1) a LAGR-style row-wise search, (2) an LRN-inspired sector selection that favors openness (proxied by calibrated depth), and (3) an Angular Cost & Depth (ACD) strategy that prefers low average cost under a minimum-lethal-depth constraint. On the reported courses, ACD was competitive or better on earlier legs and remained comparable on longer segments, benefiting from depth-aware, cost-sensitive sectoring. Both LRN and ACD improve over LAGR on longer traverses.

Planning in pixel C-space We formulated A* directly on the FPV grid and applied the normalization $C'(p) = 1 + C(p)$ so that the Euclidean heuristic remains admissible and consistent. This renormalization preserves the optimality guarantees while keeping the implementation simple and fast in image space.

Key Takeaways

- **Reliable, light-weight depth at test time:** A log-linear, LiDAR-anchored correction suffices to make monocular depth usable for mid/far range FPV reasoning without retraining.
- **Depth-aware FPV C-space is practical:** Approximate, analytic footprint inflation using separable row/column sliding maxima with depth gating can be performed in real-time and is robust to mixed-depth scenes.
- **Interpretable frontiering helps:** Angular-sector statistics (average cost, minimal lethal depth) provide a transparent alternative to purely learned affordances, and combined depth & cost-aware criterion yields stable choices across courses.
- **Heuristic soundness can be subtle:** Simple cost renormalization avoided subtle suboptimalities while preserving A* efficiency.

Limitations

- **Simulator scope:** All quantitative results are in the Falcon simulator; although the stack uses realistic intrinsics and pitch, sim-to-real gaps (appearance, lidar sparsity pattern, vibration, latency) remain.

- **Separable footprint approximation:** The row/column max filters approximate a slanted quadrilateral footprint. This can be conservative in some geometries and insufficiently conservative during sharp turns near obstacles.
- **Frontier failure modes:** The ACD/LRN sectoring may overvalue thin apertures that look open in FPV but are inadmissible after reprojection, and can be sensitive to small errors in goal projection near the FPV boundary.
- **ROS1 dependency:** The current implementation is integrated within ROS1 Noetic due to compatibility with the FieldAI (our sponsor) stack, which is also ROS1 based. While Behavior Trees can, in principle, be implemented in any framework, their support is much more mature and natively integrated within ROS2 (through libraries such as `behavior_tree_cpp`). Therefore, migrating to ROS2 in the future would simplify the addition of structured decision-making and recovery behaviors, improve modularity, and enable better real-time communication between perception, planning, and control modules.

Future Work

- **Confidence-aware planning:** Propagate depth and costmap uncertainty through the C-space transform and A* (e.g., risk-sensitive costs) with sector validity decided on probabilistic lethal-depth estimates.
- **Learned sector priors with geometric checks:** Combine ACD's interpretable statistics with a lightweight policy that predicts sector preference priors from features, but require hard depth/cost pass-fail gates to keep behavior explainable.
- **Reverse motion and behavioral recovery:** If the vehicle becomes stuck or encounters an obstacle dead-end, the planner could initiate a controlled reverse motion and then use a high-level Behavior Tree (BT) to evaluate alternative strategies such as reorienting, exploring adjacent sectors, or replanning from a new vantage point. Although Behavior Trees are not exclusive to ROS2, their native toolchain and lifecycle integration in ROS2 make them especially suitable for implementing such structured, modular recovery logic.
- **Topological memory for revisitation:** To support these behaviors, a lightweight topological graph can be maintained to record visited regions or angular sectors.

This graph would require negligible memory (only a few kilobytes even for large maps) but would provide strong contextual awareness to prevent cyclic failures, aid backtracking, and guide exploration of previously unseen areas.

- **ROS2 transition:** Porting the current stack to ROS2 would naturally enable more robust asynchronous processing, improved message transport, and native use of Behavior Trees and hierarchical decision nodes.
- **Real-world validation:** Due to the limited project duration (May 2025 – present), and the effort required to establish the Falcon simulator in a stable and reproducible configuration for testing the long-range navigation pipeline, additional real-world field testing could not be conducted within this timeframe. However, the modular structure of the proposed system ensures that future team members should find it relatively straightforward to integrate this sub-stack into the real vehicle stack. The compartmentalized design, reliance on well-defined interfaces, and limited dependency on simulator-specific modules significantly reduce integration overhead. In future, I would love to see the stack being evaluated on a physical platform operating on graded slopes, vegetation, and mixed substrates. In particular, established off-road autonomy stacks, such as, Velociraptor [23], augmented with our navigation sub-stack might enable more reliable autonomy in terms of intervention rate, traversal pace, and energy efficiency in off-road environments.
- **Closed-loop metrics:** Complement waypoint-distance summaries with success rate, time-to-goal (currently infeasible in Falcon due to some instability), and uncertainty-aware safety margins.
- **Interface with a better local planner:** Tighter coupling to a more sophisticated local planner used in off-road autonomy stacks [23].

Concluding Remarks

The work demonstrates that structured geometric reasoning can substantially improve long-range FPV navigation without heavy learning or hallucination or dense mapping. The addition of simple test-time calibration and interpretable depth-aware frontiering already shows tangible improvements. Future extensions, such as integrating recovery behaviors, reverse motion, and lightweight topological reasoning

within a ROS2 Behavior Tree framework would make the system more autonomous, fault-tolerant, and deployable in real-world unstructured environments. Moreover, tighter coupling with more sophisticated local planners such as MPPI or other non-linear MPC variants could further enhance motion smoothness, stability, and obstacle clearance. Such coupling would bridge the gap between high-level FPV planning and low-level control, thus completing a unified long-horizon navigation pipeline suitable for robust and adaptive real-world deployment.

Appendix A

On Efficient C-Space Transform

Why True 2D Dilation is Problematic A theoretically accurate C-space transformation would apply a maximum filter over the projected quadrilateral at each pixel. However, the filter window $\mathcal{W}_{i,j}$ at pixel (i, j) is non-rectangular, not axis-aligned, and varies spatially. Therefore, the operation $I'_{i,j} = \max_{(u,v)\in\mathcal{W}_{i,j}} I_{u,v}$ cannot be decomposed into a separable row-column formulation such as $I' = \max_{\text{col}}(\max_{\text{row}}(I))$. This rules out efficient 1D sliding window algorithms and makes the operation computationally intractable for real-time systems.

GPU Inefficiency of 2D Quadrilateral Max Filtering Such quadrilateral filtering is also a poor fit for parallel architectures like CUDA. Since each pixel requires a differently shaped and oriented window, threads in a warp would exhibit divergent control flow and non-coalesced memory access with no shared memory tiling or reuse. Unlike uniform rectangular filters, where local data can be preloaded and reused efficiently, each thread here would access disjoint global memory addresses, severely degrading memory throughput and compute occupancy [17].

Naive 1D Implementation A straightforward approach to maximum filtering iterates over each window and computes its maximum using a linear scan, resulting in a time complexity of $O(nw)$, where n is the number of pixels and w is the window size. While conceptually simple, this method becomes computationally expensive for large large window sizes, which is the case for us. The robot width near the bottom of

the FPV image reaches roughly 1650 pixels, almost the full image width of 1920 while the projected width decreases (non-)linearly towards the top. Consequently, the per-row computation for each window nearly scales linearly with the image width, and summing across all rows produces an effective complexity approaching $O(n^2)$. This near-quadratic behavior renders the naive method unsuitable for per-frame maximum filtering of the FPV costmap in real-time.

Efficient 1D Implementation Using Deque Given an input array (i.e. each row) of real numbers and a fixed window size w, the objective is to determine the maximum value in each contiguous subarray (window) of size w as the window slides from left to right by one element at a time. More specifically, Let `nums` $= [a_1, a_2, \ldots, a_n]$ be a list of n reals and $w \in \mathbb{Z}^+$ such that $k \leq n$. Ignoring edge padding, for each index i in the range $0 \leq i \leq n - k$, define a window $W_i = [a_{i+1}, a_{i+2}, \ldots, a_{i+k}]$. The goal is to compute `result`$[i] = \max(W_i)$.

An efficient $O(n)$ solution is possible using a double-ended queue (deque) [5] to maintain candidate indices for the maximum in each window. The core idea is to preserve a decreasing sequence of values (by index) in the deque such that the front of the deque always corresponds to the index of the current maximum.

Algorithm

(1) Initialize an empty deque `dq` to store indices and a list `result` of size n.

(2) For each index i from 0 to $n - 1$:

- Define the window bounds as:

$$\texttt{left} = \max(0, i - \lfloor w/2 \rfloor), \quad \texttt{right} = \min(n - 1, i + \lfloor w/2 \rfloor)$$

- Remove indices from the front of the deque if they are outside the current window (i.e., less than `left`).
- For all new indices j in the range not yet covered in the deque and up to `right`:
  - Remove all indices from the back of the deque if `nums`$[j] \geq$ `nums[dq[-1]]`

    - Append j to the back of the deque.
- Set `result`$[i]$ = `nums[dq[0]]`, which is the maximum value in the current (possibly partial) centered window.

Analysis The deque maintains at most w indices at any time. All elements smaller than the current are pruned since they will never be the maximum as long as the current value remains in the window. The maximum for each window is always at the front, making retrieval efficient. Each element is inserted and removed from the deque at most once, leading to an overall time complexity of $O(n)$. The deque stores at most w elements, resulting in $O(w)$ auxiliary space.

Complexity Analysis of Complete C-Space Transform Let the grid have dimensions $H \times W$, and let the sliding windows along rows and columns have sizes w_i and l_i for row i and column i, respectively.

For each row i, the sliding window considers w_i elements per column. Using a deque-based implementation, each element is pushed and popped at most once, so the operations for row i are amortized linear $O(W)$ in the number of columns. Across all rows, the total row-wise computation is $O(HW)$.

After computing the row-wise maxima, we perform a column-wise sliding window with effective length l_i (fraction of projected length) per row. In general, $H < W$. Let $k \ll H$ denote the effective amortized number of operations per row in the column, accounting for deque push/pop behavior. Each column then costs $O(Hk)$, and across all W columns, the total column-wise step is $O(HWk)$, which is also the total amortized runtime for our C-space transformation.

Appendix B

LiDAR Mask Synthesis

The Falcon simulator's native LiDAR is computationally heavy for real-time experiments. To approximate a single LiDAR scan while preserving the scanner's projection characteristics, we aggregate historical per-frame binary projections of the simulated LiDAR into a mask prior, and then compress this dense mask into a sparse, scan-like set of sample points. This derived mask is used by the `lidar_sim` node to publish lightweight, LiDAR-like observations in the ROS pipeline (Sec. **??**). The high-level steps are as follows:

- **Aggregate dense envelope:** We process all the frames in the trajectories to generate a binary dense mask, indicating every pixel that has been hit by the Falcon LiDAR scanners. This results in a dense mask, as shown in Figure B.1, with approximately $130K$ hits for the FPV mask at a resolution of 1080×1920.
- **Component-wise singleton selection:** Next, we compute the connected components on the dense mask and choose one representative pixel per component. For simplicity, we select the midpoint of the pixel list within each component, although any consistent selection method would work. The resulting image, shown in Figure B.2, contains approximately $16.5K$ nonzero points.
- **Patch-level deduplication:** To avoid clusters of nearby points, we apply a sliding $k \times k$ patch (with $k = 25$ in our implementation for the FPV resolution of 1080×1920) and retain only one point per patch, specifically the pixel closest to the center. This step produces the final sparse mask (Figure B.3), which resembles

a sparse LiDAR scan. The number of nonzero points in this final mask is about 5K, which aligns with the projected $3.5 - 5K$ points typically found in each Falcon LiDAR scan. During runtime, the mask is applied to sample from the dense depth data provided by the depth camera, generating a simulated LiDAR output. Thus, it provides a lightweight alternative to real-time ray casting done by the Falcon LiDAR sensors, while still maintaining the LiDAR projection envelope in the simulator.

Algorithm 10 provides a concise high-level description of the LiDAR mask synthesis process.

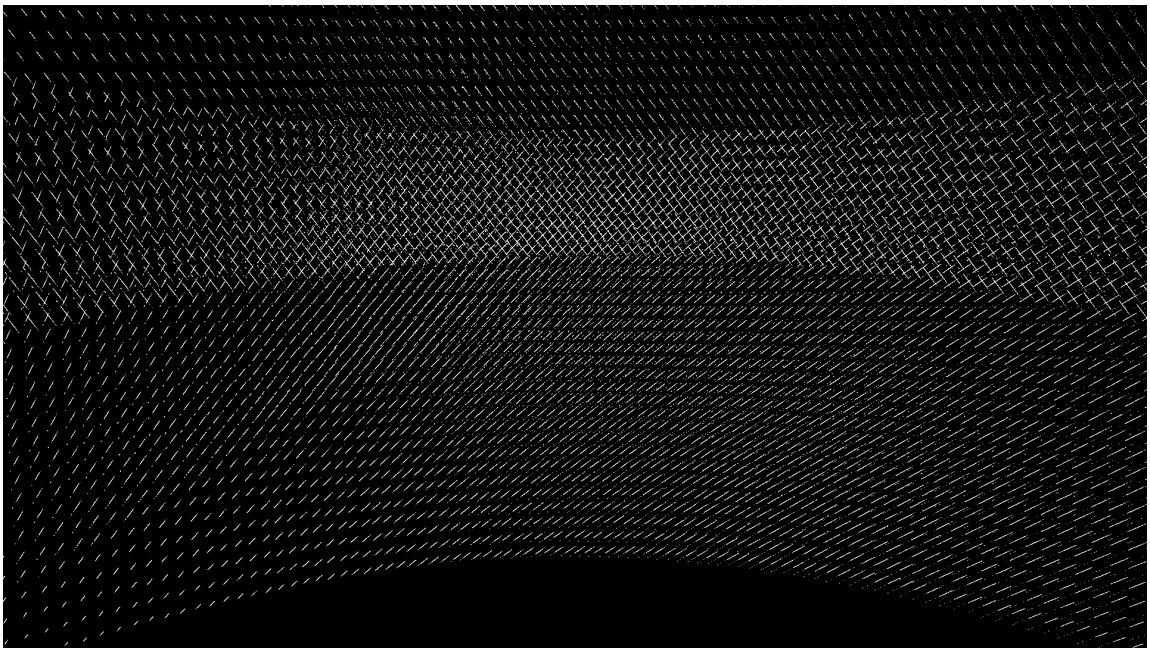

Figure B.1: Accumulated per-frame masks from the Falcon LiDAR sensors showing the history of the projection envelope.

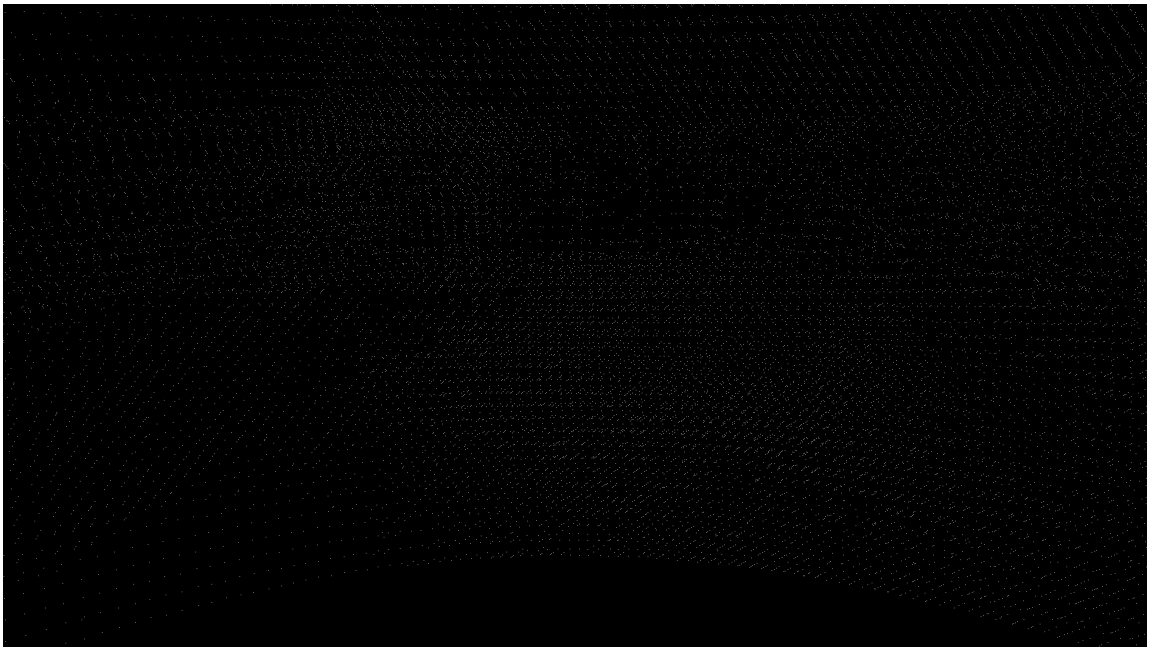

Figure B.2: Connected component singleton picking yielding a coarse sampling of the envelope in Figure B.1.

Figure B.3: Non-maximal suppression of Figure B.2 over $k \times k$ patches retains at most one point per patch, ensuring each nonzero point is visited only once. This produces a scan-like sparsity suitable for real-time use.

Algorithm 10 LiDAR Mask Synthesis

Require: Directory of binary masks $\mathcal{D}$; patch size k (odd, e.g., 25)

Ensure: Sparse mask M emulating a single LiDAR scan

$M_{\text{dense}} \leftarrow \mathbf{0}$
for each file $f \in \mathcal{D}$ **do**
    $I \leftarrow \textsc{ReadBinary}(f)$
    $M_{\text{dense}} \leftarrow M_{\text{dense}} \vee I$
end for
$\{C_i\} \leftarrow \textsc{ConnectedComponents}(M_{\text{dense}})$
$M_{\text{pt}} \leftarrow \mathbf{0}$
for each component C_i **do**
    $P \leftarrow \textsc{PixelList}(C_i)$
    idx $\leftarrow \lfloor |P|/2 \rfloor$ ▷ choose a representative (midpoint rule)
    $M_{\text{pt}}[P[\text{idx}]] \leftarrow 1$
end for
$M \leftarrow \mathbf{0}$; Processed $\leftarrow \mathbf{0}$
for each point (y, x) where $M_{\text{pt}}(y, x) = 1$ **do**
    if Processed$(y, x) = 0$ **then**
        $(y_1{:}y_2, x_1{:}x_2) \leftarrow k \times k$ patch around (y, x) within bounds
        $Q \leftarrow \textsc{PixelList}(M_{\text{pt}}[y_1{:}y_2, x_1{:}x_2])$
        choose $q^\star \in Q$ closest to patch center
        $M[q^\star] \leftarrow 1$; Processed$[y_1{:}y_2, x_1{:}x_2] \leftarrow 1$
    end if
end for
return M

Appendix C

Matrix Class for Interfacing

Many computational modules of the proposed planning framework are written in C++ for performance, while higher-level orchestration and visualization are performed in Python. To allow zero-copy data transfer between the two environments, a lightweight `Matrix4` abstraction inspired by the `Eigen` library [7] was implemented. This class enables direct access to NumPy's C-contiguous buffers through `pybind11`, without intermediate data conversion or redundant allocation.

`Matrix4` Implementation

Listing C.1 shows the full implementation of the `Matrix4` class. It is included in its entirety for completeness and reproducibility. Note that this is a minimal implementation to serve as a multidimensional array wrapper for external memory provided by NumPy through `pybind11`. The layout is fully contiguous in row-major order, which matches default NumPy default memory convention. Moreover, the destructor does not deallocate memory, ensuring that ownership remains with the Python runtime and avoiding double-free errors.

```
1 #ifndef MATRIX_4D_H
2 #define MATRIX_4D_H
3
4 #include <iostream>
5 #include <memory>
6 #include <cassert>
```

```
7  #include <limits>
8
9  #if defined(_MSC_VER)
10     #define FORCE_INLINE __forceinline
11 #elif defined(__GNUC__) || defined(__clang__)
12     #define FORCE_INLINE inline __attribute__((always_inline))
13 #else
14     #define FORCE_INLINE inline
15 #endif
16
17 template<typename T>
18 class Matrix4 {
19 private:
20     size_t depth, rows, cols, layers;
21     size_t dims;
22     T * data;
23
24 public:
25     Matrix4() : depth(0), rows(0), cols(0), layers(0), dims(0),
           data(nullptr) {}
26
27     // 3d-multi-layer (full)
28     void init(size_t d, size_t r, size_t c, size_t l, T* data) {
29         depth = d; rows = r; cols = c; layers = l; dims = 4;
30         this->data = data;
31     }
32
33     // 2d-multi-layer
34     void init(size_t r, size_t c, size_t l, T* data) {
35         depth = 1; rows = r; cols = c; layers = l; dims = 3;
36         this->data = data;
37     }
38
39     // 2d
40     void init(size_t r, size_t c, T * data) {
41         depth = 1; rows = r; cols = c; layers = 1; dims = 2;
42         this->data = data;
43     }
44
```

```
45      FORCE_INLINE size_t size() const {
46          return depth * rows * cols * layers;
47      }
48
49      FORCE_INLINE T min() const {
50          T min_ = std::numeric_limits<T>::max();
51          for (size_t i=0; i<this->size(); ++i) {
52              min_ = (data[i] < min_) ? data[i] : min_;
53          }
54
55          return min_;
56      }
57
58      FORCE_INLINE T max() const {
59          T max_ = std::numeric_limits<T>::min();
60          for (size_t i=0; i<this->size(); ++i) {
61              max_ = (data[i] > max_) ? data[i] : max_;
62          }
63
64          return max_;
65      }
66
67      FORCE_INLINE T& operator()(size_t d, size_t i, size_t j, size_t
            l) {
68          return data[d * rows * cols * layers + i * cols * layers +
                j * layers + l];
69      }
70
71      FORCE_INLINE T& operator()(size_t i, size_t j, size_t l) {
72          assert (dims <= 3);
73          return data[i * cols * layers + j * layers + l];
74      }
75
76      FORCE_INLINE T& operator()(size_t i, size_t j) {
77          assert (dims == 2);
78          return data[i * cols + j];
79      }
80
81      FORCE_INLINE T operator()(size_t d, size_t i, size_t j, size_t
```

```
        l) const {
82          return data[d * rows * cols * layers + i * cols * layers +
                j * layers + l];
83      }
84
85      FORCE_INLINE T operator()(size_t i, size_t j, size_t l) const {
86          return data[i * cols * layers + j * layers + l];
87      }
88
89      FORCE_INLINE T operator()(size_t i, size_t j) const {
90          return data[i * cols + j];
91      }
92
93      // 3d, should not index over the layers (same point)
94      FORCE_INLINE size_t indexSpatial(size_t d, size_t i, size_t j)
        const {
95          return d * rows * cols * layers + i * cols * layers + j *
                layers;
96      }
97
98      // 2d, should not index over the layers (same point)
99      FORCE_INLINE size_t indexSpatial(size_t i, size_t j) const {
100         assert((depth == 1) && (dims <= 3));
101         return i * cols * layers + j * layers;
102     }
103
104     FORCE_INLINE size_t numDepth() const { return depth; }
105     FORCE_INLINE size_t numRows() const { return rows; }
106     FORCE_INLINE size_t numCols() const { return cols; }
107     FORCE_INLINE size_t numLayers() const { return layers; }
108     FORCE_INLINE size_t numDimensions() const { return dims; }
109
110     void printShape() const {
111         std::cout << "(" << depth << " x ";
112         std::cout << rows << " x ";
113         std::cout << cols << " x ";
114         std::cout << layers << ")" << std::endl;
115     }
116
```

```
};

#endif
```

Listing C.1: Complete implementation of the Eigen-inspired Matrix4 class used as the foundation for all C++ modules in this thesis.

Minimal Example of Usage

Below we show how we perform C-space transformation, for example, using the `Matrix4` class shown above.

```
#include <pybind11/pybind11.h>
#include <pybind11/numpy.h>
#include <pybind11/stl.h>

#include "cspace.hpp"

template <typename T>
void cspace_separable_depth_aware(
    pybind11::array_t<T> py_grid_costmap,
    pybind11::array_t<T> py_grid_costmap_cspace,
    pybind11::array_t<T> py_grid_depthmap,
    T depth_tol,
    const std::vector<int> & robot_width_l,
    const std::vector<int> & robot_length_l
) {

    pybind11::buffer_info grid_info_costmap =
        py_grid_costmap.request();
    pybind11::buffer_info grid_info_costmap_cspace =
        py_grid_costmap_cspace.request();
    pybind11::buffer_info grid_info_depthmap =
        py_grid_depthmap.request();

    // currently only works for 2d
    assert(grid_info_costmap.shape.size() == 2);
    assert(grid_info_costmap_cspace.shape.size() == 2);
    assert(grid_info_depthmap.shape.size() == 2);
```

```
25
26      auto grid_ptr_costmap = static_cast <T *>
            (grid_info_costmap.ptr);
27      auto grid_ptr_costmap_cspace = static_cast <T *>
            (grid_info_costmap_cspace.ptr);
28      auto grid_ptr_depthmap = static_cast <T *>
            (grid_info_depthmap.ptr);
29
30      // configuration space transform
31      Matrix4 <T> grid_costmap, grid_costmap_cspace, grid_depthmap;
32      grid_costmap.init(
33          grid_info_costmap.shape[0],
34          grid_info_costmap.shape[1],
35          grid_ptr_costmap
36      );
37      grid_costmap_cspace.init(
38          grid_info_costmap_cspace.shape[0],
39          grid_info_costmap_cspace.shape[1],
40          grid_ptr_costmap_cspace
41      );
42      grid_depthmap.init(
43          grid_info_depthmap.shape[0],
44          grid_info_depthmap.shape[1],
45          grid_ptr_depthmap
46      );
47
48      sliding_window_max_separable_depth_aware<T>(
49          grid_costmap, grid_costmap_cspace,
50          grid_depthmap, depth_tol,
51          robot_width_l, robot_length_l);
52  }
53
54  void bind_cspace(pybind11::module &m) {
55      m.def("cspace_separable_depth_aware_f32",
56          &cspace_separable_depth_aware<float>,
57          pybind11::arg("py_grid_costmap"),
58          pybind11::arg("py_grid_costmap_cspace"),
59          pybind11::arg("py_grid_depthmap"),
60          pybind11::arg("depth_tol"),
```

```
61          pybind11::arg("robot_width_l"),
62          pybind11::arg("robot_length_l"),
63          "Cspace (separable row/col, depth-aware) <float> transform."
64      );
65
66      m.def("cspace_separable_depth_aware_f64",
67          &cspace_separable_depth_aware<double>,
68          pybind11::arg("py_grid_costmap"),
69          pybind11::arg("py_grid_costmap_cspace"),
70          pybind11::arg("py_grid_depthmap"),
71          pybind11::arg("depth_tol"),
72          pybind11::arg("robot_width_l"),
73          pybind11::arg("robot_length_l"),
74          "Cspace (separable row/col, depth-aware) <double>
                transform."
75      );
76  }
77
78  PYBIND11_MODULE(planning, m) {
79      m.doc() = "pybind11 planning plugin";
80      bind_cspace(m);
81      // other bindings ...
82  }
```

Listing C.2: Minimal `pybind11` bridge using `Matrix4` class as a C++ view over NumPy arrays.

On the Python side, arrays are explicitly converted to contiguous memory before being passed to C++. This ensures that all arrays passed to C++ use a flat, row-major (C-style) memory layout compatible with the internal pointer arithmetic of the `Matrix4` class. Listing C.3 shows how to call the `pybind11` bridge from Python for the C-space transformation.

```
1  # ...
2
3  def make_contiguous(x: np.ndarray) -> np.ndarray:
4      x = np.ascontiguousarray(x, dtype=x.dtype)
5      assert x.flags['C_CONTIGUOUS'], "Failed to make numpy array C contiguous"
6      return x
```

```
7
8  # ...
9
10 planning.cspace_separable_depth_aware_f32(
11     make_contiguous(costmap),
12     make_contiguous(costmap_csp),
13     make_contiguous(depth),
14     depth_tol=depth_tol,
15     robot_width_l=robot_width_l,
16     robot_length_l=robot_length_l,
17 )
```

Listing C.3: Python bridge for contiguous NumPy array preparation.

Bibliography

[1] Falcon – Digital Twin Simulation Platform. https://www.duality.ai. (document), 1, 1, 3.1, 3

[2] Field AI. https://www.fieldai.com. (document), 3

[3] Max Bajracharya, Benyang Tang, Andrew Howard, Michael Turmon, and Larry Matthies. Learning long-range terrain classification for autonomous navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 4018–4024, 2008. doi: 10.1109/ROBOT.2008.4543828. 2

[4] Mateo Guaman Castro, Samuel Triest, Wenshan Wang, Jason M. Gregory, Felix Sanchez, John G. Rogers, and Sebastian Scherer. How does it feel? self-supervised costmap learning for off-road vehicle traversability. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 931–938, 2023. doi: 10.1109/ICRA48891.2023.10160856. 2

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844. A

[6] Ethan Fahnestock, Erick Fuentes, Samuel Prentice, Vasileios Vasilopoulos, Philip R Osteen, Thomas Howard, and Nicholas Roy. Far-field image-based traversability mapping for a priori unknown natural environments. *IEEE Robotics and Automation Letters*, 10(6):6039–6046, 2025. doi: 10.1109/LRA.2025.3563808. 1

[7] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. C

[8] M. Happold, M. Ollis, and N. Johnson. Enhancing supervised terrain classification with predictive unsupervised learning. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006. doi: 10.15607/RSS.2006.II.006. 2

[9] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

6.1.1

[10] Haiwen Huang, Anpei Chen, Volodymyr Havrylov, Andreas Geiger, and Dan Zhang. Loftup: Learning a coordinate-based feature upsampler for vision foundation models, 2025. URL https://arxiv.org/abs/2504.14032. 3

[11] Wesley H. Huang, Mark Ollis, Michael Happold, and Brian A. Stancil. Image-based path planning for outdoor mobile robots. *J. Field Robot.*, 26(2):196–211, February 2009. ISSN 1556-4959. 1, 1, 2, 5.2.2, 1, 5.3

[12] Larry D. Jackel, Eric Krotkov, Michael Perschbacher, James Pippine, and Chad Sullivan. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *Journal of Field Robotics*, 23, 2006. 1

[13] Dongshin Kim, Jie Sun, Sang Min Oh, J.M. Rehg, and A.F. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 518–525, 2006. doi: 10.1109/ROBOT.2006.1641763. 2

[14] Roberto Manduchi, Andres Castano, Ashit Talukder, and Larry H. Matthies. Obstacle detection and terrain classification for autonomous off-road navigation. *Auton. Robots*, 18(1):81–102, 2005. doi: 10.1023/B:AURO.0000047286.62481.1D. URL https://doi.org/10.1023/B:AURO.0000047286.62481.1d. 2

[15] Matias Mattamala, Jonas Frey, Piotr Libera, Nived Chebrolu, Georg Martius, Cesar Cadena, Marco Hutter, and Maurice Fallon. Wild visual navigation: fast traversability learning via pre-trained models and online self-supervision. *Auton. Robots*, 49(3), July 2025. ISSN 0929-5593. doi: 10.1007/s10514-025-10202-x. URL https://doi.org/10.1007/s10514-025-10202-x. 2

[16] Xiangyun Meng, Nathan Hatch, Alexander Lambert, Anqi Li, Nolan Wagener, Matthew Schmittle, Joonho Lee, Wentao Yuan, Zoey Qiuyu Chen, Samuel Deng, Greg Okopal, Dieter Fox, Byron Boots, and Amirreza Shaban. Terrainnet: Visual modeling of complex terrain for high-speed, off-road navigation. In *Robotics: Science and Systems*, 2023. 2

[17] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, March 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL https://doi.org/10.1145/1365490.1365500. A

[18] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma,

Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL https://arxiv.org/abs/2304.07193. 3

[19] Manthan Patel, Jonas Frey, Deegan Atha, Patrick Spieler, Marco Hutter, and Shehryar Khattak. RoadRunner M&M - Learning Multi-Range Multi-Resolution Traversability Maps for Autonomous Off-Road Navigation. *IEEE Robotics and Automation Letters*, 9(12):11425–11432, 2024. doi: 10.1109/LRA.2024.3490404. 2

[20] Matt Schmittle, Rohan Baijal, Nathan Hatch, Rosario Scalise, Mateo Guaman Castro, Sidharth Talia, Khimya Khetarpal, Byron Boots, and Siddhartha Srinivasa. Long range navigator (LRN): Extending robot planning horizons beyond metric maps. In *9th Annual Conference on Robot Learning*, 2025. URL https://openreview.net/forum?id=QtVZUPCKrY. 1, 1, 2, 2, 5.3, 5.3.2

[21] Matthew Sivaprakasam, Samuel Triest, Cherie Ho, Shubhra Aich, Jeric Lew, Isaiah Adu, Wenshan Wang, and Sebastian Scherer. Salon: Self-supervised adaptive learning for off-road navigation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16999–17006, 2025. doi: 10.1109/ICRA55743.2025.11128268. 2, 3

[22] Boris Sofman, Ellie Lin, J. Andrew Bagnell, John Cole, Nicolas Vandapel, and Anthony Stentz. Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, 23(11-12):1059–1075, 2006. doi: https://doi.org/10.1002/rob.20169. 2

[23] Samuel Triest, Matthew Sivaprakasam, Shubhra Aich, David Fan, Wenshan Wang, and Sebastian Scherer. Velociraptor: Leveraging visual foundation models for label-free, risk-aware off-road navigation. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=AhEE5wrcLU. 1, 2, 8

[24] Ruicheng Wang, Sicheng Xu, Yue Dong, Yu Deng, Jianfeng Xiang, Zelong Lv, Guangzhong Sun, Xin Tong, and Jiaolong Yang. Moge-2: Accurate monocular geometry with metric scale and sharp details. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=16mDq7m2OK. 3

[25] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997. doi: 10.1109/CIRA.1997.613851. 2