

Terrain Aware Tactical Motion Planning and Control Algorithms for Off-Road UGVs in GNSS-Denied Hostile Environments

Jyotirmoy Mukherjee

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Andrea L'Afflitto, Ph.D., Chair

Corina Sandu, Ph.D., Co-chair

Steve Southward, Ph.D.

Kaveh Akbari Hamed, Ph.D.

David Gorsich, Ph.D.

April 30, 2025

Blacksburg, Virginia

Keywords: Motion Planning, Unmanned Ground Vehicles, GNSS Denied, Off-Road
Navigation, Tactical Contro

Copyright 2025, Jyotirmoy Mukherjee

Terrain Aware Tactical Motion Planning and Control Algorithms for Off-Road UGVs in GNSS-Denied Hostile Environments

Jyotirmoy Mukherjee

(ABSTRACT)

This dissertation introduces an advanced framework for terrain-aware tactical motion planning and control of off-road unmanned ground vehicles operating in environments where global navigation satellite systems are unavailable due to adversarial interference or structural constraints. The research focuses on enabling autonomous navigation in uncharted, hostile terrains by developing a hierarchical autonomy stack that seamlessly integrates navigation, path planning, trajectory planning, and control functionalities. The navigation system employs onboard vision-based and inertial sensors to construct real-time environmental representations, utilizing geometric segmentation techniques such as random sample consensus and inverse ray tracing to differentiate traversable surfaces from obstacles. These representations, encompassing occupancy grids and topographic profiles, account for terrain geometry and surface characteristics, providing a foundation for subsequent planning stages.

The path planning module leverages heuristic-driven graph search strategies to compute waypoint sequences that optimize for tactical concealment while ensuring efficient progression toward a goal. A novel stealth-inducing mechanism biases paths to exploit obstacle proximity for cover, and a dual-mode interaction paradigm distinguishes between protective shelters and hazardous entities, enhancing strategic navigation in contested settings. The trajectory planning module transforms these waypoints into smooth, time-parameterized trajectories through cubic polynomial spline interpolation, incorporating terrain-adaptive orientation via

rotation-minimizing frames to maintain kinematic consistency over uneven landscapes. A tactical velocity modulation scheme adjusts motion dynamics based on terrain elevation and obstacle proximity, while collision avoidance is achieved through the generation of safe navigation corridors using geometric constructs. The control system ensures robust trajectory execution, compensating for environmental uncertainties.

Field deployment on a robotic platform in a controlled indoor environment validates the framework's capability to map unknown terrains, generate stealth-aware paths, and produce feasible trajectories under GNSS-denied conditions. The research advances autonomous off-road navigation by offering scalable algorithms that enhance tactical decision-making, with potential applications in military reconnaissance, disaster response, and extraterrestrial exploration.

Terrain Aware Tactical Motion Planning and Control Algorithms for Off-Road UGVs in GNSS-Denied Hostile Environments

Jyotirmoy Mukherjee

(GENERAL AUDIENCE ABSTRACT)

This dissertation explores new ways to help self-driving off-road vehicles travel through dangerous areas where GPS signals don't work, such as in war zones or disaster-stricken regions. The goal is to make these vehicles, called unmanned ground vehicles, smart enough to move on their own while staying hidden and safe in tough environments with uneven ground and obstacles. The research creates a system that acts like a brain for the vehicle, breaking down the job into steps: understanding the surroundings, planning a safe route, figuring out how to move smoothly, and making sure the vehicle follows that plan.

The system starts by using cameras and sensors on the vehicle to create a map of the area as it moves, spotting things like rocks or walls and figuring out what parts of the ground are safe to drive on. Then, it plans a path that keeps the vehicle close to objects that can hide it, like trees or buildings, while still heading toward its destination. This path avoids dangerous spots and chooses routes that make the vehicle harder to spot. Next, the system makes a detailed plan for how the vehicle should move over the bumpy ground, adjusting its speed to be careful on steep slopes or near hiding spots, and ensuring it doesn't bump into anything by creating safe zones to travel through. Finally, the vehicle follows this plan, making small changes as needed to handle unexpected challenges.

The ideas were tested on a real robotic vehicle in an indoor space, showing that the vehicle could map its surroundings, pick a hidden route, and move smoothly while avoiding obstacles.

This work helps make self-driving vehicles more useful for tasks like delivering supplies in war zones, helping after natural disasters, or even exploring other planets.

Dedication

To Mom and Dad

Acknowledgments

Pursuing research is both an exciting and challenging journey — one that demands constant navigation through the unknown, a space where curiosity flourishes but comes *vis-à-vis* with uncertainty. In such uncharted terrain, companionship becomes not just comforting but essential. It is the steadfast support, encouragement, and wisdom of others that serve as guiding stars — helping one stay the course when the path ahead is unclear. This journey would not have been possible without the presence of many remarkable individuals who walked beside me, lit the way, or simply stood by through it all.

I would like to express my heartfelt gratitude to my advisors, Dr. Andrea L’Afflitto and Dr. Corina Sandu, for their unwavering support and guidance throughout this journey. Their technical expertise undoubtedly shaped the trajectory of this research, but more than that, it was our conversations — the questions they asked, the challenges they posed, the brainstorming sessions, and the trust they placed in my growth — that truly defined my experience. I began this work with curiosity and a handful of questions. Through our regular discussions, that curiosity evolved into a structured research program — from unboxing the Husky to developing the autonomy stack, one layer at a time. Their mentorship extended beyond research; they taught me what no textbook could - how to think with clarity, persist with confidence, and build with purpose. In many ways, they were my “navigation and guidance system,” and I am deeply grateful for the lasting impact they have had on my work and mindset.

I would also like to thank my committee members — Dr. Steve Southward, Dr. Kaveh Akbari Hamed, and Dr. David Gorsich — for their guidance and support throughout this work. I thank Dr. Southward for sharing valuable insights on real-time operations in au-

onomous systems. Serving as his graduate teaching assistant offered a memorable perspective on bridging simulation with hardware, reinforcing the connection between theory and practice. I am grateful to Dr. Hamed for his course on robotics, which introduced me to a comprehensive view of modeling and control. His structured and insightful teaching greatly influenced the direction of this dissertation. I also thank Dr. Gorsich for his continued involvement and thoughtful feedback throughout this project. His insights helped frame the broader relevance of this research and kept its real-world impact in focus.

I would like to thank my colleagues from the Advanced Control Systems Laboratory (ACSL) — Julius, Mattia, Giri, and Paul — and from the Terramechanics, Multibody, and Vehicle Systems (TMVS) Lab — Ekansh, Alex, Varsha, Chaitanya, Hannah, Sadegh, Mohit, and others — for their collaboration and camaraderie throughout my time at Virginia Tech. I am grateful for the technical discussions, shared projects, and moments of teamwork — whether directly related to my research or beyond it. These interactions enriched my understanding, broadened my perspective, and made the research environment dynamic and engaging.

I am deeply thankful to my friends — Almas ji, Diksha, Kushal, Disha, Anish, Karthik, Anamika, Kewal, Aish, Balu, Harsh, Samruddha, Abi, Santhiya, Harish, Namrata, Debanjali, and many others — for being a constant source of support and joy. Thank you for the thoughtful conversations, shared travels, and for celebrating the small wins along the way. Your presence made this journey far more meaningful.

Above all, I owe everything to my parents. My mother, who first taught me to read, and my father, who introduced me to mathematics, planted the earliest seeds of this work. Their encouragement to pursue higher education, even from afar, gave me the strength and confidence to move forward. Despite the distance, they were always within reach — their unwavering support, steady words, and belief in me made all the difference. This work stands

on the foundation they built, and I am forever grateful.

Lastly, I offer my deepest gratitude to God. The timeless wisdom of the Gita, the Vedas, and the Upanishads has been a constant source of clarity, perspective, and inner strength throughout this journey. Words written centuries ago often illuminated the path ahead more than anything else. This work would not have been possible without His grace.

*I set out not with answers, but with questions in my hand,
A seeker on uneven ground, no map, no promised land.
No plotted line to guide my steps, no signal from the skies,
Yet every slip and course correction bid the system rise.*

*With trial and trust, the frame was tuned, through feedback sharp and slow,
And limits once called boundaries became the place to grow.
In loss appeared a bearing point; in stillness, quiet gain.
In every loop that broke apart, emerged a truth from strain.*

*No lines of code stood clean and bare — within them ran a thread,
Of questions left unspoken and of paths not always read.
Some voices gave the heading true; some wings, the lift to soar;
And some remained the ground below — steady, evermore.*

*So here I stand, the model trained, with sensors pointing true,
Still charting paths through spaces where the questions rise anew.
This work is not just autonomy, nor charts of cost and span —
It's but the quiet art of seeking all I can.*

Contents

List of Figures	xvi
1 Introduction	1
1.1 Motivation and Background	1
1.2 Mission Requirement and Challenges	3
1.3 Autonomy Stack	3
1.4 Research Questions	6
1.5 Contributions	7
2 Global Path Planning	9
2.1 Introduction	9
2.2 Literature review on global path planning	11
2.2.1 Algorithms in path planning	12
2.2.2 Tactical path planning	14
2.2.3 Terrain-aware path planning	15
2.3 Proposed global tactical A^* path planner	17
2.3.1 Weighing Function for Concealment-Seeking Behavior in Global Paths	18
2.3.2 Proofs of Optimality and Admissibility	20

2.3.3	Execution Framework of the Tactical A^* Algorithm	23
2.3.4	Numerical Results for Tactical A^* Planner	25
2.4	Challenges of A^* in Dynamic Environments with Incremental Map-Building	30
2.5	Proposed Global Tactical LPA^* Path Planner	32
2.5.1	Execution Framework of the Tactical LPA^* Algorithm	33
2.5.2	Data Structure for the Tactical LPA^* Priority Queue	36
2.5.3	Numerical Results for Tactical LPA^* Planner	37
2.6	Dual mode obstacle interaction in tactical path planning	43
2.6.1	Heuristic Redesign in Dual-Mode Interaction	45
2.6.2	Case Study Setup: Mine Evasion in Military Applications	46
2.7	Conclusions	48
3	Trajectory planning	50
3.1	Introduction	50
3.1.1	Problem Formulation of Trajectory Planning	52
3.1.2	Design philosophy of trajectory planning	53
3.2	Literature review	55
3.2.1	Review on collision avoidance strategies in local planners:	57
3.2.2	Review on terrain aware trajectory planning strategies:	59
3.3	Proposed cubic polynomial trajectory planner	64

3.4	Attitude Planner	65
3.4.1	The Double Reflection Method	66
3.4.2	Proposed Double Projection Method	68
3.5	Spline Parameterization and Time Scaling	70
3.5.1	<i>Op-Tempo</i> Tactical Velocity Planning	71
3.6	Angular Velocity Planner	73
3.6.1	Forward Euler Angular Interpolation	75
3.6.2	Rodrigues axis-angle method	76
3.6.3	SLERP Interpolation	77
3.7	Proposed collision avoidance algorithms	78
3.7.1	Safety corridor generation via point cloud partitioning	79
3.7.2	Safe Convex Volume Maximization for Collision Avoidance	81
3.8	Numerical case studies	84
3.8.1	Results on Attitude Planning	84
3.8.2	Results on Spline Parameterization	90
3.8.3	Results on Reference Angular Velocity Planning	94
3.8.4	Results on Collision Avoidance	96
3.9	Conclusions and Future Work	99
4	Navigation	103

4.1	Introduction	103
4.2	Review on Sensing for Tactical Navigation in GNSS-Denied Environments	105
4.3	Sensors for tactical navigation	109
4.4	Vehicle pose estimation	112
4.4.1	Sensor Fusion for Pose Estimation	112
4.5	Point cloud segmentation	115
4.5.1	Problem Formulation for Point Cloud Segmentation	116
4.5.2	Segmentation using the random sample consensus (RANSAC) algorithm	118
4.5.3	Segmentation Using Inverse Ray Tracing (IRT)	120
4.6	Voxel-Based 3D Mapping	123
4.6.1	Bayesian Inverse Sensor Model for 3D Mapping	126
4.6.2	Experimental Tests for 3D Voxel Mapping	129
4.7	Occupancy grid mapping - experimental results and validation	132
4.7.1	Experimental Validation of 2D Occupancy Grid Mapping	132
4.8	Terrain Topography Mapping	133
4.8.1	Iterative convex hull-based terrain topography mapping	136
4.8.2	Experimental validation of terrain topography mapping	137
4.9	Conclusions and Future Work	144
5	Real-time experiments	147

5.1	Introduction	147
5.2	Clearpath Husky Robotic platform	149
5.2.1	Vehicle description	150
5.2.2	Distributed ROS Communication Architecture	152
5.3	Autonomy stack architecture	153
5.4	Experimental Test Scenario	155
5.4.1	Test Scenario Description	155
5.4.2	Module Evaluation	157
5.5	Conclusions	160
6	Conclusions and Future Work	162
6.1	Problems Addressed	162
6.2	Contributions and Developments	163
6.3	Key Learnings	163
6.4	Future Recommendations	164
	Bibliography	165

List of Figures

1.1	ROS-based autonomy stack for tactical navigation on the Clearpath Husky A200. The diagram illustrates the data flow from sensors (Intel RealSense T265 and D435i) through EKF sensor fusion, point cloud segmentation, mapping, path planning, and trajectory planning modules, culminating in control commands	4
1.2	An integrated autonomy stack for UGVs: The figure illustrates the inter-dependent modules of Navigation, Path Planning, Trajectory Planning, and Control.	5
2.1	Plot of the stealth inducing $\kappa(\alpha)$ function with different tunable parameter triplets.	19
2.2	Simulated environment for path planning algorithm testing: The Blender-created world features uneven terrain and deliberately placed obstacles, with the green flag indicating the start point and the red circle marking the goal. This environment is used to validate the UGV's path planning and obstacle avoidance algorithms.	26
2.3	Results of tactical path planning simulation: The visual shows the computed path in five scenarios ranging from extreme recklessness shown by the red markers to extreme levels of stealth as shown by the white markers.	28

2.4	Top view of tactical path planning simulation results: The overhead perspective shows the UGV's computed paths through five scenarios, with red markers indicating extreme recklessness and white markers representing extreme stealth by seeking shelter around the obstacles.	29
2.5	Distance between UGV path and obstacles for different tactical parameters: The graph shows the trade-off between distance traveled versus stealth. The more stealthy the path, the more distance the UGV travels.	30
2.6	Virtual test environment developed in Blender for evaluating tactical and terrain-aware navigation strategies based on Lifelong Planning A* (LPA*). The simulation incorporates realistic terrain undulations and heterogeneous obstacles representing shelters	38
2.7	Simulated paths using <i>LPA*</i> . The black path represents the most tactical trajectory, prioritizing concealment and obstacle cover. The blue path reflects a reckless strategy, favoring direct goal pursuit with minimal regard for tactical considerations. The magenta path demonstrates a semi-tactical approach, balancing stealth and goal-directedness	39
2.8	Path length comparison between <i>LPA*</i> and <i>A*</i> across stealth levels: The bar chart shows increasing path lengths as stealth level rises, reflecting detours to coast shelters.	40
2.9	Distance to nearest obstacle across stealth levels for <i>LPA*</i> : The box plot shows decreasing median distances as stealth level increases, indicating closer shelter-coasting behavior.	41

2.10	Shannon entropy of path directions across stealth levels for LPA^* : The bar chart shows increasing entropy as stealth level rises, indicating greater directional unpredictability.	43
2.11	Blender generated virtual environment with safe shelters and dangerous mines. This simulated world is used to evaluate the efficiency of the proposed dual obstacle interactive path planner exhibiting both concealment seeking and mine-evasive characteristics	44
2.12	Top view of the Blender generated world with safe shelters and mines used to test the dual obstacle interactive path planner	45
2.13	Simulated result of the dual obstacle aware LPA^* path planner. The green path represents a reckless goal pursuing path whilst being mine evasive whereas the red path exhibits higher stealth as well as mine evasiveness	47
3.1	Frame of reference: Gravity-aligned fixed inertial reference frame and body fixed reference frame	53
3.2	Interpolated spline points with the $SO(3)$ matrices representing the UGV orientation at every time instant in the future horizon.	74
3.3	Reference waypoints: Sequence of waypoints $\mathcal{W}(r)$ produced by the path planner with different levels of tactical parameters as in Chapter 2. The red markers represent the most reckless path whereas the white markers represent the most tactical path	85
3.4	Path smoothening and time scaling: the figure shows the C^2 continuous and smoothened waypoints received from the path planner.	85

3.5	Frénet-Serret frames determined along the spline trajectory: The plot illustrates how the Frénet-Serret frames undergo twisting at the inflection points of the trajectory, causing reorientation of the normal and binormal axes. . . .	86
3.6	Rotation-minimizing frames to determine the UGV orientation along the spline trajectory: The plot shows the application of rotation-minimizing frames (RMFs) to maintain smooth UGV orientation along the trajectory, minimizing unnecessary twisting and ensuring stable orientation.	87
3.7	Torsion and curvature comparison for the spline trajectory: The top plot shows the torsion values, while the bottom plot depicts the curvature along the spline trajectory. The near-zero torsion for RMFs demonstrates their effectiveness in minimizing twisting, making them suitable for UGV orientation.	87
3.8	The torsion (top) and curvature (bottom) of the splines using Wang's double reflection technique compared to those obtained using the proposed double projection technique indicate reduced torsion values almost equal to zero but preservation of the splines curvature	89
3.9	Comparison of simulation time for attitude planners: The plot compares the computational time of Wang's double reflection method (literature) and the proposed double projection method across varying time vector resolutions. The double projection method consistently shows lower computational time, suggesting better suitability for real-time applications.	89
3.10	Components of the velocity vector for constant velocity: The top row shows the components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$ as a function of time, while the bottom row shows them as a function of the spline parameter ξ . . .	91

3.11	Velocity profile along the spline for constant velocity: The graph shows a constant velocity of 5 ms^{-1} over time, representing uniform motion along the cubic spline.	91
3.12	Velocity scaling with elevation along the spline: The plot shows the tactical velocity $V(\xi)$ modulated by the op-tempo function $\chi(\xi)$, decreasing during elevation climbs and increasing on flat terrain.	92
3.13	Velocity scaling with shelter proximity along the spline: The plot shows the tactical velocity $V(\xi)$ decreasing near shelters (low $d_{\mathcal{O}}(\xi)$) and increasing when farther away.	92
3.14	Velocity components in inertial and body frames on uneven terrain: The top row shows components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$, while the bottom row shows $\hat{v}_x(t)$, $\hat{v}_y(t)$, $\hat{v}_z(t)$ in the body frame $\mathbb{J}$, with near-zero $\hat{v}_y(t)$ and $\hat{v}_z(t)$ confirming non-holonomic constraints.	93
3.15	Velocity components in inertial and body frames on flat terrain: The top row shows components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$, while the bottom row shows $\hat{v}_x(t)$, $\hat{v}_y(t)$, $\hat{v}_z(t)$ in the body frame $\mathbb{J}$, with minimal $v_z(t)$ and $\hat{v}_z(t)$ due to flat terrain.	93
3.16	Angular velocities at high resolution: The plots show $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ computed by SLERP, SQUAD, Rodrigues, and Forward Euler at 640 time points, compared to the ground truth. All methods converge closely at high resolution, but struggle at lower resolutions.	95

3.17	Angular velocities at low resolution: The plots show $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ computed by SLERP, SQUAD, Rodrigues, and Forward Euler at 40 time points, compared to the ground truth. Rodrigues, SLERP and SQUAD track the ground truth more effectively than Forward Euler.	96
3.18	Obstacle map with various geometric shapes and a noise-inclusive model to simulate real-world conditions: The map displays different geometric obstacles (circles, squares, and ellipses) with noise added to simulate environmental uncertainty, providing a complex terrain for UGV path planning and obstacle avoidance.	97
3.19	Safety corridor generated by the algorithm, highlighting the collision avoidance strategy: The figure shows the safety corridor (in blue) around the trajectory, with red indicating areas of potential collision. The algorithm adjusts the path within the corridor to ensure safe navigation and obstacle avoidance.	98
3.20	Ellipsoid-based collision avoidance: The figure shows two ellipsoids (blue wireframes) centered along the trajectory, connected by a red line from $(x, y, z) \approx (5, 5, -2)$ to $(x, y, z) \approx (15, 15, 2)$, marked by green and red stars. Obstacle points (black dots) lie outside the ellipsoids, ensuring a collision-free path.	100
3.21	Ellipsoid-based collision avoidance (close-up): The figure shows two ellipsoids along the UGV trajectory.	100
4.1	The Intel RealSense D435i camera produces color and depth information of surfaces in its field of view in the form of point clouds	110

4.2	Intel RealSense T265 tracking camera provides 6DoF pose and inertial information through visual-inertial odometry (VIO), enabling robust localization in GNSS-denied environments.	111
4.3	Comparison of UGV path using sensor fusion versus dead reckoning: The red line represents the path estimated by the T265 camera alone, while the blue line corresponds to the path obtained through Extended Kalman Filter (EKF) fusion of T265 pose, MicroStrain 3DM-GX3-25 IMU, and wheel encoder data.	114
4.4	RGB view of a test terrain with artificial obstacles: The image shows a controlled experimental setup featuring a terrain with visible tracks, simulating real-world conditions for testing terrain segmentation and obstacle detection systems	115
4.5	Point cloud representation of terrain: A 3D visualization of the terrain captured by the UGV's sensors, showing the distribution of surface points used for ground segmentation and topography mapping	116
4.6	Segmented point clouds in the camera frame. The yellow dots correspond to the positive obstacle set whereas the green dots capture the terrain topography	119
4.7	A dome with discernible hexagonal features on its surface, illustrating the line-of-sight principle used in inverse ray tracing for terrain segmentation. . .	121
4.8	Illustration of the inverse ray tracing technique for terrain topography extraction: The diagram shows how virtual rays are cast from a plane $\mathcal{V}_p$ below the terrain to detect terrain points $\mathcal{T}$, while ignoring obstacles $\mathcal{O}$, enabling accurate terrain modeling.	122

4.9	3D representation of terrain topography: The mesh grid illustrates the terrain's surface, highlighting key features such as elevation changes and curvatures. The point labeled with the yellow circle corresponds to the terrain set $\mathcal{T}$, which is registered in the global frame to form the terrain topography map $\mathfrak{M}^{\mathcal{T}}$.	125
4.10	Point cloud registration: A 3D visualization demonstrating the process of aligning point cloud data in virtual discrete bins called voxels, with point clouds (blue) representing obstacle locations $\mathcal{O}$ within the mapped environment. The registration process is crucial for accurately modeling the terrain and obstacles in UGV navigation.	126
4.11	Voxel map representation of obstacle data $\mathfrak{M}^{\mathcal{O}}$: The 3D grid illustrates the discretization of space into voxels, with black cubes representing grid elements as occupied after registration of the obstacle set $\mathcal{O}$.	127
4.12	Experimental setup and results for 3D voxel mapping in an indoor lab environment. Top: Photographs of the lab (20 m $\times$ 20 m) showing cluttered areas with desks and boxes (left) and open spaces (right). Bottom: Corresponding voxel maps visualized in RViz, with colors representing height (blue for higher elevations, green to yellow for lower elevations).	131
4.13	Validation of 2D occupancy grid mapping in an indoor hallway environment. Top: Photograph of the hallway with tiled floors (each tile 60 cm $\times$ 60 cm), showing a distance of 5 tiles (3 m) between walls	134
4.14	Indoor test environment with tiled floor and static obstacles: The image shows a hallway setup used for validating obstacle detection and occupancy grid mapping, with tile dimensions marked to evaluate occupancy map accuracy	135

4.15	Terramechanics rig: Controlled experimental facility with visible terrain features in the order of centimeters. The experimental rig consists of a carriage that allows for longitudinal motion along the rut, which was used for testing the accuracy of the on-board navigation system	139
4.16	Terramechanics rig: Controlled experimental facility with visible terrain features in the order of centimeters. The experimental rig consists of a carriage that allows for longitudinal motion along the rut, which was used for testing the accuracy of the on-board navigation system	139
4.17	3D model of rut captured using a laser scanner: The <i>.stl</i> file representation of the tire track rut provides high-resolution terrain data, used for validating UGV navigation algorithms against precise terrain features	140
4.18	Experimental validation of terrain topography modeling: The graph compares terrain topography captured by a laser scanner (black), iterative convex hull topography (red), and inverse sensor model (green). The results demonstrate the accuracy of the proposed algorithms against the ground truth obtained from the laser scanner	142
4.19	Experimental validation of terrain topography modeling: The graph compares terrain topography captured by a laser scanner (black), iterative convex hull topography (red), and inverse sensor model (green). The results demonstrate the accuracy of the proposed algorithms against the ground truth obtained from the laser scanner	142
4.20	Field test of terrain topography modeling: The image shows a bumpy terrain whose topography is modeled using the developed navigation system	143

4.21	Experimental validation of terrain topography modeling: The scatter plot represents the terrain points in the inertial frame, capturing the terrain topography. Green represents higher elevation, whereas blue represents lower elevation	144
5.1	Clearpath Husky UGV with key dimensions: 990 mm (length), 670 mm (width), 390 mm (height), 555 mm (track), and 512 mm (wheelbase). The platform is shown equipped with sensors used in this study, including Intel RealSense cameras and an IMU, mounted on the payload bay.	151
5.2	ROS-based autonomy stack for tactical navigation on the Clearpath Husky A200. The diagram illustrates the data flow from sensors (Intel RealSense T265 and D435i) through EKF sensor fusion, point cloud segmentation, mapping, path planning, and trajectory planning modules, culminating in control commands	155
5.3	Indoor test scenario in a lab environment. The top images show the maze constructed with cardboard boxes. The bottom left image displays the RViz visualization of the voxel map.	156
5.4	Indoor field test results for path planning. Navigation system capturing the lab maze, with boxes in the occupancy grid $\mathfrak{M}^{\mathcal{O}}$ (white) and planned path (green dashed line) from start navigating the maze while avoiding obstacles.	158
5.5	Trajectory planning results in the indoor lab maze. The trajectory (blue curve) is overlaid on the point cloud map, with $SO(3)$ frames (red, green, blue axes) visualizing the RMFs along the path, ensuring smooth, terrain-aware navigation.	160

List of Symbols

$\chi(\xi)$	Op-tempo function for velocity scaling
κ	Stealth-inducing function
$\mathbb{I}$	Gravity-aligned inertial reference frame
$\mathbb{J}$	Body-fixed reference frame
$\mathcal{G}$	Goal set
$\mathcal{O}$	Obstacle set
$\mathcal{T}$	Terrain set
$\mathcal{V}_p$	Virtual plane below the terrain (used in inverse ray tracing)
$\mathfrak{M}^{\mathcal{O}}$	Occupancy grid map (voxel map)
$\mathfrak{M}^T$	Terrain topography map
$\mathcal{R}(t)$	Reference trajectory (time-parameterized)
$\mathcal{W}$	Reference path (sequence of waypoints)
μ_1, μ_2, μ_3	Tunable parameters for the κ function
ω	Angular velocity
ξ	Spline parameter
d_2	Euclidean distance metric

g	Path cost function
h	Heuristic map
R	Rotation matrix
r	Position of the UGV's center of gravity
r_0	Initial position of the UGV
rhs	Right-hand side cost function (used in LPA*)
$SE(3)$	Special Euclidean group (pose transformations)
$SO(3)$	Special orthogonal group (rotation matrices)
T	Transformation matrix (pose of the UGV)
$V(\xi)$	Tactical velocity along the spline

Chapter 1

Introduction

The present research addresses the tactical necessities of Unmanned Ground Vehicles (UGVs) operating in hostile environments where global navigation satellite systems (GNSS) are denied and the terrain is highly unstructured. Drawing upon insights from animal behavior in [1, 2], the research aims to design algorithms for an autonomy stack, based on a hierarchical and modular approach, to imbue the UGV with stealth capabilities. The major contributions include the development of a fast geometry-based terrain segmentation algorithm, a terrain topography mapping algorithm, a tactical path planning algorithm for navigation in three dimensions, a polynomial-based trajectory planner with real-time update capabilities, and the real-world deployment and validation of the integrated autonomy stack on the Clearpath Husky platform.

1.1 Motivation and Background

Human-operated off-road vehicles have been pivotal in applications such as disaster response [3], military operations [4], and search and rescue [5]. These vehicles, while effective, often incur significant costs related to human operator training and vehicle crashes [6]. In scenarios like chemical, biological, radiological, and nuclear (CBRN) events [7], manned vehicles could expose operators to health hazards. UGVs, however, can minimize these risks. In urgent rescue operations, UGVs can operate when human responders are unavailable,

potentially saving lives during the critical "golden hour" [8]. The development of off-road UGVs thus offers benefits across multiple domains: economically, by reducing training and crash costs; in terms of human safety, by mitigating risks in hazardous scenarios; and scientifically, by creating scalable algorithms that advance robotic capabilities, as evidenced by growing research in this field [9, 10].

The remarkable advancements in autonomous vehicle technology trace back to the visionary work of the Defense Advanced Research Projects Agency (DARPA) [11]. A central challenge identified was creating robotic platforms capable of traversing complex, unpredictable terrains. Wheeled robots, in particular, show immense potential due to their ability to transport heavy payloads, supported by extensive literature on vehicle dynamics. However, while current technology suffices for navigation across flat, structured terrains, the unpredictability of unstructured environments poses significant mobility challenges. Larger vehicular systems often lack the tactical agility required for comprehensive combat operations, leading to DARPA's Tactical Mobile Robots (TMR) program, established by their Tactical Technology Office. This program drew inspiration from biomimetics, resulting in compact robotic platforms that emulate animal locomotion.

This research stems from the challenges posed by the TMR program. While the TMR program defined *tactical* as the ability to maneuver unstructured terrain, this dissertation redefines *tactical* as the UGV's capacity to strategize paths for concealment against opponents. The results extend beyond the TMR program, addressing needs at DoD research centers like ARL (TBAM, SARA programs), GVSC, and ERDC. The framework's applications also include civilian scenarios, such as supporting law enforcement and first responders in post-disaster environments (*e.g.*, earthquakes, flash floods), stealth monitoring in national parks, and exploration of celestial bodies like the moon and Mars through autonomous ground vehicles.

1.2 Mission Requirement and Challenges

The tactical algorithms investigated in this research target UGVs providing logistic support and exploring unknown, potentially hostile environments, indoors or outdoors. Whether the mission involves reaching a goal or exploring a region, a primary concern in hostile settings is balancing task completion with stealth and security. UGVs must operate covertly to reduce the risk of detection, targeting, and disablement by opponents. Additionally, these environments often lack prior models, and relying on external sources like GNSS is unreliable due to poor signal reception or intentional disruptions. Thus, the first overarching goal of this research is to develop algorithms that enable a UGV to navigate from a start point to a goal set without prior environmental knowledge, relying solely on onboard sensors while prioritizing covertness.

UGVs traversing uneven terrains are locally constrained to the surface but operate in 3D relative to a gravity-aligned inertial frame, necessitating path planning that accounts for terrain elevation. Terrain attributes—categorized as topography (geometric features) and surface properties (*e.g.*, cohesiveness, moisture, friction)—directly impact mobility. This leads to the second overarching goal: incorporating terrain properties into motion planning to ensure feasible, covert trajectories that adapt to varying surface conditions while maintaining stealth.

1.3 Autonomy Stack

This research adopts a hierarchical architecture for the autonomy stack, enabling a modular framework that integrates diverse functionalities. The stack comprises four main modules—navigation, path planning, trajectory planning, and trajectory tracking—each processing in-

puts progressively to refine raw data into actionable motor commands, as shown in Figures 1.2 and 1.1

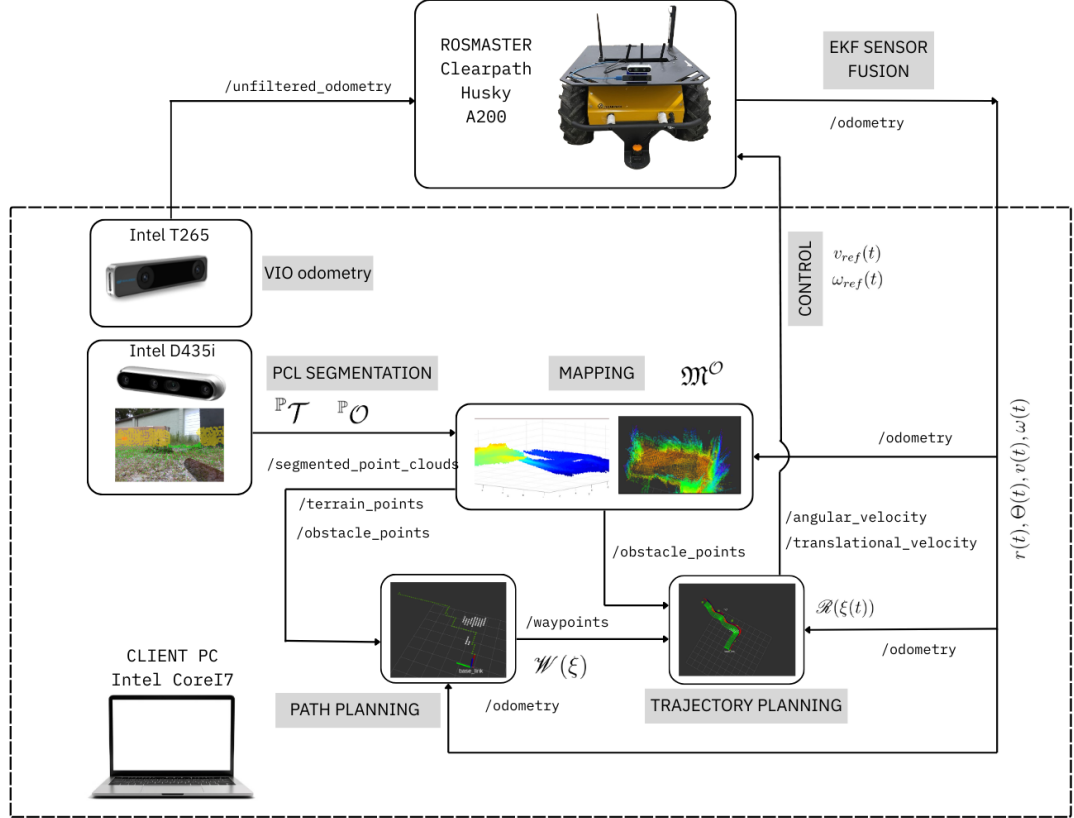


Figure 1.1: ROS-based autonomy stack for tactical navigation on the Clearpath Husky A200. The diagram illustrates the data flow from sensors (Intel RealSense T265 and D435i) through EKF sensor fusion, point cloud segmentation, mapping, path planning, and trajectory planning modules, culminating in control commands

Navigation: The navigation module uses onboard vision-based sensors for exteroceptive data and proprioceptive sensors to build a computational representation of the environment, including obstacles and terrain topography. This map, along with the UGV's updated position, is fed to other modules. In this research, the navigation system segments point clouds from depth cameras, registers them into a map, and performs sensor fusion for localization in GNSS-denied environments.

Path Planning: The path planning module uses the real-time map from the navigation module to generate a collision-free sequence of waypoints. These waypoints are spatial markers that prioritize the tactical aspect of the path, rewarding concealment. This research investigates a graph-based path planner with heuristic functions, minimizing a cost function that balances urgent goal pursuit with stealthy motion.

Trajectory Planning: The trajectory planning module generates a time-parameterized, high-resolution vector of vehicle configurations (position and orientation), ensuring kinematic and dynamic feasibility and short-range collision safety. This research employs polynomial curves with an analytical attitude planning framework to produce high-resolution trajectories.

Trajectory Tracking: The trajectory tracking module, or control module, generates commands to track the reference trajectory, ensuring precise execution despite unmodeled uncertainties.

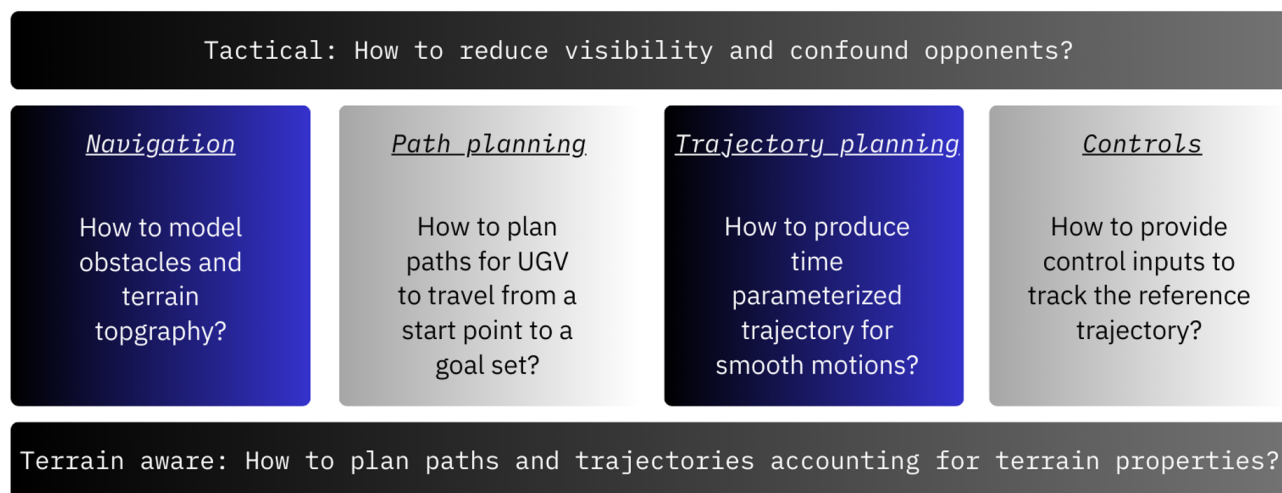


Figure 1.2: An integrated autonomy stack for UGVs: The figure illustrates the interdependent modules of Navigation, Path Planning, Trajectory Planning, and Control.

1.4 Research Questions

Each module in the autonomy stack serves a specific purpose. To develop terrain-aware, tactical motion trajectories, the following research questions are investigated:

1. How can the concept of “coverttness” be quantitatively modeled within motion planning algorithms to enhance the tactical capabilities of UGVs? Coverttness is crucial for strategic deployment in hostile environments. This research focuses on a heuristic-based algorithm that rewards paths closer to obstacles, reducing detectability while demonstrating scalability across robotic platforms.
2. In what ways can terrain topography be effectively incorporated into motion planning to ensure adaptability to uneven terrain profiles? Challenges like kinematic feasibility and load transfer require accounting for terrain elevation, improving path feasibility over planar assumptions.
3. What strategies can be employed to plan and adjust trajectories online, utilizing limited sensor data, without prior environmental knowledge? In hostile settings, relying on onboard sensors shields the UGV from external disruptions, enabling real-time trajectory adjustments.
4. How can the underlying algorithms be formulated to ensure reliable computational efficiency for real-time deployment? Deploying the motion planning algorithms interact with large amounts of data, hence optimizing the solution process and data handling is key to achieve desirable real-time performance.

1.5 Contributions

This research developed a comprehensive tactical navigation framework, addressing the challenges of GNSS-denied environments through novel algorithms and experimental validation. A key contribution is the navigation module, which includes two geometry-based ground segmentation algorithms to process point clouds from depth cameras: the Random Sample Consensus (RANSAC) algorithm and a novel inverse ray tracing (IRT) method (Chapter 4). IRT overcomes RANSAC’s limitation of false positives by leveraging geometric perspectives, ensuring accurate distinction between traversable surfaces and obstacles without the computational overhead of learning-based models [12].

The segmented point clouds are registered into maps using two approaches: a voxel-based mapping technique with a Bayesian inverse sensor model for the occupancy map, and a novel iterative convex hull method for the terrain topography map (Chapter 4). The voxel approach accurately captures obstacles, while the convex hull method preserves terrain curvature without artificial discretization, enhancing computational efficiency. These algorithms were experimentally validated in controlled indoor and outdoor settings, confirming their suitability for real-time mapping.

The path planning module introduces a tactical planner based on A* and LPA*, incorporating stealth-enhancing modifications through a heuristic that rewards paths closer to obstacles (Chapter 2). The planner uses three tunable parameters to modulate stealth intensity, enabling a spectrum from urgent goal pursuit to extreme caution with concealment. A dual-mode obstacle interaction strategy distinguishes shelters from hazards, ensuring safe navigation. While LPA* did not outperform A*, it provided complementary benefits in dynamic environments by reusing search results for faster updates.

The trajectory planner generates a time-parameterized trajectory using cubic polyno-

mial splines, interpolating waypoints while ensuring kinematic feasibility and collision safety (Chapter 3). A novel op-tempo velocity regulation framework adjusts speed based on terrain and obstacle proximity, and an attitude planner computes 3D orientations using rotation-minimizing frames (RMFs) via a double projection method, ensuring smooth, terrain-aware motion. The trajectory tracking module generates control commands to execute the trajectory on the Clearpath Husky, adapting to unmodeled uncertainties.

The integrated autonomy stack was deployed and validated on the Clearpath Husky in an indoor lab maze, demonstrating accurate mapping, stealth-aware path generation, and feasible trajectory planning (Chapter 5). The real-world tests highlight the framework’s potential for military applications, with LPA* enhancing path unpredictability, though full-stack integration revealed areas for improvement in reference tracking.

Chapter 2

Global Path Planning

Contributions

This chapter presents the algorithmic foundations of two tactical global path planners that generates a path as a series of waypoints in $\mathbb{R}^3$ connecting a start location to a goal set. The key contributions are:

- A novel stealth inducing function for concealment-seeking behavior
- Tactical extensions to the A^* and LPA^* algorithms using cost regulation
- A dual-mode obstacle interaction strategy for minefield scenarios
- Numerical case studies in realistic maps

2.1 Introduction

Autonomous navigation in uncharted territories necessitate the planning of a spatial route as well as a time parameterized trajectory from a start location in the fixed inertial frame to a goal point or a goal set in the fixed inertial frame. The term *motion planning* is used interchangeably in literature with contextual definitions in distinct applications [13]. In this dissertation, *motion planning* refers to the process of generating both, the reference spatial waypoints as well as the temporal reference state vector that connects a start point and a goal set within a map of the environment.

Let $\mathbb{I} \triangleq \{O; X, Y, Z\}$ denote a gravity-aligned orthonormal inertial reference frame centered at O . Let $r \in \mathbb{R}^3$ denote the position of the UGV's center of gravity (C.G.) and $\mathbb{J} \triangleq \{r; X', Y', Z'\}$ denote a body fixed orthonormal reference frame centered at the UGV's C.G., r . Let $r_0 \in \mathbb{R}^3$ denote the initial position of the UGV, $\mathcal{G} \triangleq \{r_i \mid \|r_i - r_g\|_2 < \sigma, i \in \mathbb{N}, r_i \in \mathbb{R}^3\}$ denote the goal set, of a radius $\sigma \in \mathbb{R}$ around an expected goal point $r_g \in \mathbb{R}^3$, in the reference frame $\mathbb{I}$ and $\mathfrak{M}^O \triangleq \mathbb{Z}_n \times \mathbb{Z}_m \times \mathbb{Z}_k \rightarrow \{0, 1\}$ denote the voxel map or occupancy map which represents the computational abstraction of the world. The transformation matrix

$$T \triangleq \begin{bmatrix} R & r \\ 0_{1 \times 3} & 1 \end{bmatrix}, \quad (2.1)$$

where $T \in SE(3)$ represents the pose of the UGV in $\mathbb{I}$, $R \in \mathbb{R}^{3 \times 3} \in SO(3)$ and $r \in \mathbb{R}^3$. The motion planning problem in this dissertation is thus defined more formally as determining both the spatial waypoints and subsequently a time parameterized reference vector of the UGV's states that connects the start point r_0 and the goal set $\mathcal{G}$.

Two pertinent definitions entail. If this reference vector is parameterized purely in space, without temporal dependence, then it is defined as a **path** and expressed as, $\mathcal{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$. Alternatively, if this reference vector is parameterized in time, it is defined as a **trajectory** and expressed as $\mathcal{R} \triangleq T(t), t \in [0, \infty)$. Furthermore, a planner can also be classified based on boundary conditions: a **global planner** connects a start point r_0 to the mission-wide goal $\mathcal{G}$, whereas, a **local planner** generates local or intermediate segments of the reference vector.

In pursuit of accomplishing the mission requirement in this research, that is, reaching the goal set whilst executing a stealthy maneuver, it is crucial to delineate the architecture of the motion planning system. A *global reference trajectory* characterized by the time attribute

could indeed ensure kinematic and dynamic feasibility, but could risk being a computational liability, for the emergence of obstacles in the UGV's field of view necessitates immediate adjustments and replanning, at least in the vicinity of the UGV. Conversely, a *global reference path* may not guarantee dynamic feasibility, but proves to be particularly suited in dynamic environments which demands fast replanning. Hence, the motion planner is divided into two levels namely, a global *path planner* and a local *trajectory planner*, thereby allowing the system for fast planning at the global level and maximizing the system's response to dynamic changes in the local environment [14]. Furthermore, since the UGV's controller ultimately tracks a finer resolution local trajectory, the global path planner's resolution can be modulated to generate dense or coarse spatial waypoints contingent on the available computational resources. The subsequent sections of this chapter elaborates on the global path planner.

2.2 Literature review on global path planning

Path planning algorithms are pervasive across autonomous robotic platforms such as UGVs, unmanned aerial vehicles (UAVs), autonomous underwater vehicles (AUVs), *etc.* Whereas the domain of applications are distinct, the mathematical basis of these algorithms remains consistent. This universality can be encapsulated via a generic function signature

$$\mathcal{W} \triangleq f(r_0, \mathcal{G}, \mathfrak{M}), \quad (2.2)$$

where $\mathfrak{M}$ denotes the map, r_0 denotes the start point, $\mathcal{G}$ denotes the goal set, and $\mathcal{W}$ denotes the sequence of waypoints. Multiple approaches to solve this problem leads to the categorization of planners such as graph-based planners, probabilistic planners and potential field based planners [14].

Although a comprehensive understanding of the advantages and disadvantages of each planning algorithm is crucial - considering path optimality, computational efficiency, collision avoidance capabilities and dimensionality- the ultimate choice of the algorithm is contingent on its capacity to fulfill the specific requirements of achieving terrain-aware covert maneuvers. To this end, the selected path planning algorithm must allow adaptive modification of a path in response to obstacles as they are detected by the on-board navigation system in an environment unknown *a priori*. An understanding of the theoretical background of the existing path planning algorithms to ascertain its applicability in this research is thus paramount.

2.2.1 Algorithms in path planning

The classification of path planning algorithms is predicated on various facets such as problem formulation, map data structure, requirement of optimality and feasibility, deterministic or stochastic, *etc.* Some of the notable path planning algorithms include Dijkstra’s search algorithm, A^* , D^* , lifelong planning A^* (LPA^*), Rapidly exploring random tree (RRT), RRT*, potential field method, wavefront planners [15], Bellman Ford algorithm [16], dynamic window approach [17], genetic algorithms [18], road map methods like visibility graph method [19], power diagrams [20], *etc.* For an in-depth exploration of path planning algorithms, readers are directed to [13, 15, 21]. Given the multitude of path planning algorithms, a pertinent question concerns the choice of the most suitable algorithm. The representation of the configuration space through a *map*, significantly influences the suitability of an algorithm; for instance, the authors in [15] presents a case where Wavefront planner could resolve the “local minima” issue that is inherent in potential field methods, however, it strictly requires grid space representations. This research however adopts the voxel map representation of the environment due to several advantages elaborated in Chapter 4. Furthermore, the global

level planner is designed for fast replanning with coarser resolution. Having thus established these requirements, we focus exclusively on those path planning algorithms that are suitable for a grid based *cell decomposed map*.

The earliest grid-based or graph-based planners can be traced back to Dijkstra’s algorithm, which was formulated to find the shortest path between two points [15]. Since then, several algorithms have emerged providing additional functionalities to address application specific needs. For example, the A^* search algorithm embedded a heuristic function over the classical Dijkstra’s algorithm to accelerate the goal finding process by directing the search direction biased towards the goal [13]. Although the A^* algorithm boasts computational efficiency, planning for finer resolution waypoints may not result in the true shortest path. The discretized representation of the world produces zig-zag like paths. This issue was mitigated by the ThetA^* and Phi^* algorithms that allow for any angle movements thereby producing more “natural” paths [22]. Further advancements in these algorithms addressed the needs of the dynamic nature of the world which necessitates real time capable navigation in voxel maps. This led to the development of algorithms like the D^* algorithm [15], which was designed to “adjust” the resultant path in a local neighborhood of the updated obstacle in the voxel map. The need for such algorithms becomes particularly relevant when the speed of the obstacle is neither too slow wherein it could be considered quasi static, nor too fast, wherein it has minimal influence on the previously planned path. This adaptability is underpinned by the algorithms’ ability to optimize a user-defined cost function that includes heuristic evaluations, balancing the need for rapid computation with the precision required for navigating complex environments.

The variants of the graph search methods discussed thus far indeed solve an optimization problem, but remain gradient free. Potential field methods [15] on the other hand solve a gradient-based optimization problem, but suffers from the problem of getting “stuck”

in a local minima [14]. It is noteworthy to mention that, whereas solving for a path by minimizing a cost function guarantees path optimality, planning on configuration spaces over high dimensions could pose substantial computational challenges. Addressing such issues paved way for the sampling based methods like RRT and its variants RRT*, RRT-connect [23]. The sampling methods are adept at exploring vast spaces of high dimensions by random sampling the environment and constructing a pathfinding tree incrementally.

Given that the present research seeks an algorithm that provides the capability of modifying the generated path based on the distance to some safe set, such as a neighborhood of obstacles in static environments, a heuristic based planner appears to be the most reasonable choice. Furthermore, since the dimensions of the configuration space is relatively low, heuristic planners will not only guarantee resolute completeness [15], but also ensure path optimality. Two challenges, however, are yet to be addressed. First, how to modify the algorithm to exhibit covertness and second, how to make the algorithm terrain topography aware. The next two sections provide a survey of the state-of-the-art approaches, limitations and mathematical constructs that will be used to address the challenge of formulating and solving the terrain topography aware, tactical path planning.

2.2.2 Tactical path planning

Although the tactical guidance problem for mobile ground robots in contested environments has already been explored, this field is at its earliest stages. Worthy of mention is CoverNav [24], where a deep reinforcement learning method is being employed to plan covert trajectories for off-road UGVs. In particular, CoverNav selects obstacles as shelters to hide behind thus overriding the segmentation problem employed in this paper and enabling the use of some user-defined criteria for the choice of the sheltering obstacle. The authors in

[25] present a solution to the path planning problem in contested environments, wherein the opponents' locations are unknown, by exploiting the terrain elevation. The authors in [2, 26] employed a potential field method to design tactical paths assuming that the traversable spaces are at the same elevation and obstacles are the only means of maintaining covertness. None of these works, however, seem to emphasize the relevance of planning a path or a trajectory that accounts for the terrain features and, hence, the cost of its traversability. Furthermore, these algorithms do not leverage the inherent advantages of heuristic based planners, which is the intention of the proposed research. Algorithms for tactical planning of UAVs, which leverage heuristic-based planners have been proposed in [1]. However, also this field is relatively under-explored.

2.2.3 Terrain-aware path planning

Successful planning off-road is contingent on three factors. First, Vehicle related information such as mass, inertia, tractive power, slip ratio, energy consumption, vehicle size, *etc* which essentially help to quantify the mobility aspect of a vehicle. Second, the geometric attributes of the environment which encompasses both positive as well as negative obstacles along with terrain topography. An obstacle is considered positive if it blocks a vehicle. Conversely, an obstacle is said to be negative, if the vehicle is rendered immobile due to terrain characteristics such as swamps or pits. The third factor is the physical attribute of an environment. The physical attributes of positive obstacles and terrain are both substantial. A positive obstacle can be considered as "traversable obstacle" if it does not hinder vehicle motion such as tall grass, however, a positive obstacle such as a rock or tree can indeed render the mission unsuccessful. Furthermore, the material properties of the terrain is substantial, as it directly affects vehicle mobility. For example, high speed causes less compaction compared to slow speeds, [27], thereby resulting in lesser terrain induced

resistance under high speeds compared to low speeds.

Heuristic-based planners have been widely studied in off-road path planning. The underlying philosophy of heuristic-based graph search algorithms remains consistent. They find an optimal path by minimizing a predefined cost function. The crucial aspect here is the design of the cost function as it encapsulates the factors affecting the path profile and off-road mobility. For example, the authors in [27] modified the A^* cost function for enhanced mobility, by incorporating parameters like vehicle footprint, traction type and a weighing factor to determine the severity of the cost. The authors in [28] on the other hand, modified the A^* algorithm for both positive and negative obstacle avoidance by penalizing paths in certain areas quantified by a simulated “uncertainty value” and closeness to an obstacle.

Some studies have leveraged on a hybrid architecture of path planning algorithms. For example, the authors in [29] addressed the issue of collision safe and mobility aware planning in 3 stages. First, a D^* lite algorithm was used to generate a coarse sequence of waypoints which essentially guide the node sampling of the second RRT* layer. A third, local optimization algorithm then penalized both collision prone paths and a parameter called, “connection judgment” which depends on the radius of curvature between two nodes. The authors in [30] addressed the mobility aware path planning using an ant colony optimization algorithm. The configuration space was represented as a 3D meshed grid and modeled with terrain height and soil remodeling index (RCI), a measure of soil strength. Recently, learning-based methods coupled with heuristic search algorithms have been investigated. For example, the authors in [31] formulated a hybrid A^* algorithm combined with a neural network, namely VGG to guide the results of the A^* algorithm. It is evident from these algorithms that modifications to the cost function by accounting for terrain height, terrain type, vehicle mobility parameters indeed, influence the quality of the produced path in terms of path shape with respect to positive and negative obstacles as well as vehicle mobility.

From this discussion and the requirement of heuristic-based planners on a voxel map framework, the present research adapts the strategy of tactical planning from [1, 2] to a two-dimensional voxel map context. Furthermore, acknowledging the challenge of deploying the path planner in a map that is incrementally built, this research extends the A^* path planner to the LPA^* methodology which is proven to be more computationally efficient in the described scenarios [32]. In order to imbue awareness of terrain topography, a voxel map based projection algorithm is developed. The subsequent sections delineate the theoretical framework and solution methodology to solve the path planning problem.

2.3 Proposed global tactical A^* path planner

The underlying philosophy of the A^* path planning problem addressed in this research is twofold. First, the path planner generates an optimal path on a 2D occupancy map $\mathfrak{M}^{\mathcal{O}} \triangleq \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$. This sequence of optimal waypoints in $\mathbb{R}^2$ is then transformed to $\mathbb{R}^3$ using a separate map, called the terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{r_i | r_i \in \mathbb{R}^3\}$ which essentially encapsulates the terrain's surface profile as a set of points. In practice, once the occupancy map $\mathfrak{M}^{\mathcal{O}}$ and the terrain topography map $\mathfrak{M}^{\mathcal{T}}$ are updated by the navigation system, they are synchronized for real-time applications, and the two-step process is employed. As delineated in Equation 2.2, the problem involves solving a sequence of waypoints that connect the starting position r_0 to a final goal set $\mathcal{G}$ but with an added requirement of minimizing a user-defined cost function. The canonical cost function for A^* path planning is defined as:

$$f_k \triangleq g_k + h_k, \quad k \in \{0, \dots, n_p\}, \quad (2.3)$$

where g_k is known as the **cost-to-come** function, h_k as the **heuristic cost**, the integer $k \in \{0, \dots, n_p\}$ serves as a waypoint index and $n_p \in \mathbb{N}$ is the size of the set of waypoints. The

function g_k typically represents the actual cost incurred to move from one node to another. Conversely, h_k is a user-defined heuristic that ideally underestimates the cost to reach the goal, infusing the optimization process with a strategic bias tailored to specific application needs. Formally, the cost-to-come function is expressed as,

$$g_k \triangleq \sum_{q=0}^k [\kappa(d_2(\hat{r}_q, \mathcal{O}))d_2(\hat{r}_q, \hat{r}_{q+1})], \quad (2.4)$$

where, $\hat{r}_q \in \mathfrak{M}^{\mathcal{O}}$ denotes the UGV's current location, $\{r\}_{q+1} \subset \mathfrak{M}^{\mathcal{O}}$ denotes the UGV's subsequent planned waypoints, and $d_2(\cdot, \cdot)$ denotes the *Euclidean distance*. The heuristic function is formally expressed as,

$$h_k \triangleq (1 - \mu_2)d_2(\hat{r}_k, \mathcal{G}) \quad (2.5)$$

and

$$\kappa(\alpha) \triangleq 1 - \mu_2 e^{4\mu_1\mu_3 - (\mu_3\alpha + \mu_1\alpha^{-1})^2}, \quad \alpha > 0. \quad (2.6)$$

The scalar variable $k \in \mathbb{N}$ represents the index of the query grid cell in the map $\mathfrak{M}^{\mathcal{O}}$.

2.3.1 Weighing Function for Concealment-Seeking Behavior in Global Paths

In the context of tactical path planning investigated in this research, a *tactical path shape* refers to a path that strategically navigates close to the obstacle set, $\mathcal{O} \triangleq \{x_i, y_i \mid \mathfrak{M}^{\mathcal{O}}(x_i, y_i) = 1\}$, to minimize the vehicle's exposure to external threats, such as detection by adversarial sensors, while still reaching the goal set $\mathcal{G}$, as it reduces the UGV's exposure to detection as

well as external threats. Otherwise, the path is said to be reckless. This behavior is critical in stealth-oriented missions, where the UGV must balance the trade-off between path length and detectability. A purely distance-optimal path, as produced by a classical A^* algorithm, may traverse open spaces, increasing the risk of detection. In contrast, a tactical path leverages environmental features, such as walls or barriers, to obscure the UGV's trajectory, even if it results in a slightly longer path.

The tactical path shape is induced by introducing the weighing function, $\kappa : (0, \infty) \rightarrow (0, 1]$ as defined in equation 2.6. Here, $\alpha \in \mathbb{R}$ represents the distance to the nearest obstacle and is computed for each node (x, y) by determining the minimum distance to an obstacle in any direction. The parameters $\mu_1 \in \mathbb{R}$, $\mu_2 \in \mathbb{R}$, and $\mu_3 \in \mathbb{R}$ can be adjusted by the user to modulate the algorithm's propensity towards reckless or tactical behavior. The function $\kappa(\alpha)$ scales the cost of traversing a node, rewarding paths closer to obstacles (small α) by reducing the effective cost, while penalizing paths in open spaces (large α).

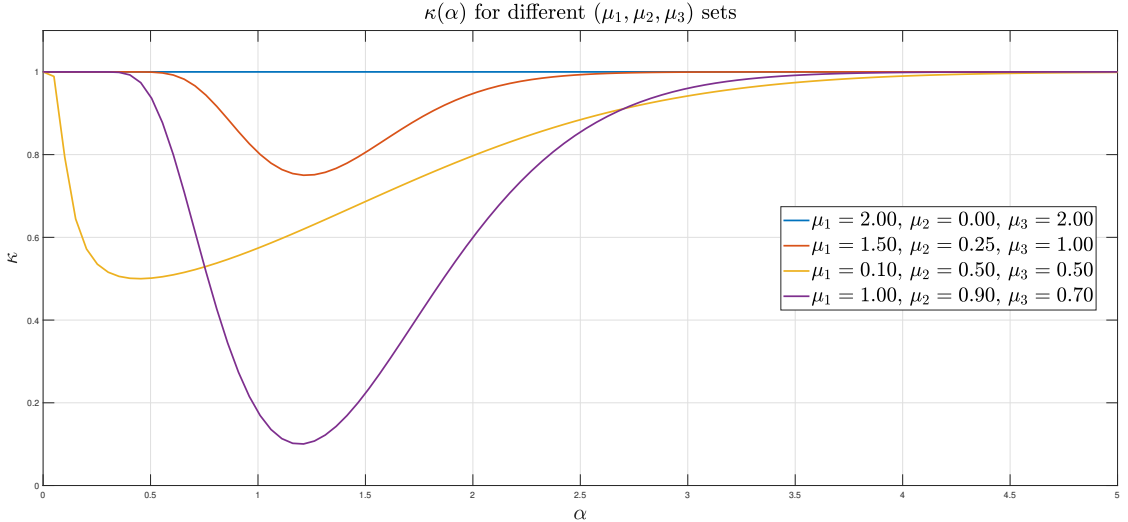


Figure 2.1: Plot of the stealth inducing $\kappa(\alpha)$ function with different tunable parameter triplets.

The underlying effects of the modulating parameters can be observed as follows:

- The parameters μ_1 and μ_3 dictate the UGV's propensity toward tactical behavior
- μ_1 controls the sensitivity to obstacle proximity. Larger values of μ_1 amplify the effect of α on κ , encouraging the UGV to stay closer to obstacles.
- μ_3 adjusts the decay rate of the exponential term, fine-tuning the transition between tactical and distance-optimal behavior.
- The parameter $\mu_2 \in [0, 0.5]$, which affects both the *cost-to-come* and *heuristic cost*, regulates the focus on goal pursuit. For example, setting $\mu_2 = 0$ encourages reckless goal pursuit as opposed to concealment seeking affinity otherwise.
- The minimizer of $\kappa(\cdot)$ is $\sqrt{\mu_1 \mu_3^{-1}}$ [1, 2]. Consequently, the choice of μ_1 and μ_3 affects the path behavior. The prescribed ranges for the parameters are established to be $\mu_1 \in (1, \infty)$, $\mu_2 \in [0, 0.5]$, and $\mu_3 \in (0.7, \infty)$ for reckless behaviors and $\mu_1 \in (0, 1)$, $\mu_2 \in (0.5, 1]$, $\mu_3 \in (0, 0.7]$ for tactical behavior.

This flexibility allows the path planner to adapt to mission-specific requirements, ensuring that the UGV's trajectory aligns with the desired tactical profile.

2.3.2 Proofs of Optimality and Admissibility

To ensure that the modified cost functions yield optimal yet tactical paths, we must verify that they satisfy the *principle of triangular inequality* (for the path optimality) and the *principle of admissibility* (for the heuristic). These properties guarantee that the A^* algorithm remains optimal while inducing the desired tactical behavior.

Principle of Triangular Inequality

The triangular inequality ensures that the cost of a path between two nodes via an intermediate node is at least as large as the direct cost between the two nodes. For any nodes u , v , and w , the modified cost must satisfy:

$$c(u, w) + c(w, v) \geq c(u, v), \quad (2.7)$$

where $c(u, v)$ is the cost of traversing from node u to node v , defined as:

$$c(u, v) = \kappa(d_2(u, \mathcal{O}))d_2(u, v). \quad (2.8)$$

Consider three nodes u , w , and v , with distances $d_2(u, w)$, $d_2(w, v)$, and $d_2(u, v)$, and obstacle distances $\alpha_u = d_2(u, \mathcal{O})$, $\alpha_w = d_2(w, \mathcal{O})$, and $\alpha_v = d_2(v, \mathcal{O})$. The left-hand side of Equation (2.7) is:

$$c(u, w) + c(w, v) = \kappa(\alpha_u)d_2(u, w) + \kappa(\alpha_w)d_2(w, v). \quad (2.9)$$

The right-hand side is:

$$c(u, v) = \kappa(\alpha_u)d_2(u, v). \quad (2.10)$$

Since $\kappa(\alpha) \in (0, 1]$, we can bound $\kappa(\alpha_u)$ and $\kappa(\alpha_w)$ by 1. The Euclidean distance satisfies the triangular inequality:

$$d_2(u, w) + d_2(w, v) \geq d_2(u, v). \quad (2.11)$$

Thus:

$$\kappa(\alpha_u)d_2(u, w) + \kappa(\alpha_w)d_2(w, v) \leq 1 \cdot d_2(u, w) + 1 \cdot d_2(w, v) = d_2(u, w) + d_2(w, v) \geq d_2(u, v). \quad (2.12)$$

To account for varying κ values, consider the worst-case scenario where $\kappa(\alpha_u) = \kappa(\alpha_w) = \kappa_{\min} > 0$ (*e.g.*, when α is small, minimizing κ). Then:

$$\kappa_{\min}d_2(u, w) + \kappa_{\min}d_2(w, v) = \kappa_{\min}(d_2(u, w) + d_2(w, v)) \geq \kappa_{\min}d_2(u, v). \quad (2.13)$$

However, the right-hand side uses $\kappa(\alpha_u)$, which may differ from $\kappa(\alpha_w)$. Since $\kappa(\alpha_u) \leq 1$, we have $\kappa(\alpha_u)d_2(u, v) \leq d_2(u, v)$. The inequality holds in the worst case (*e.g.*, $\kappa = 1$), and for smaller κ , the left-hand side decreases, but the right-hand side adjusts proportionally. Thus, the modified cost function satisfies the triangular inequality, ensuring geometric consistency of the path costs.

Principle of Admissibility

The principle of admissibility requires that the heuristic h_k never overestimates the true cost to the goal. Let h_k^* denote the true minimum cost from node k to the goal $\mathcal{G}$. The heuristic is given by:

$$h_k = (1 - \mu_2)d_2(\hat{r}_k, \mathcal{G}). \quad (2.14)$$

The true cost h_k^* is the minimum cost of a path from any node k to $\mathcal{G}$,

$$h_k^* = d_2(k, \mathcal{G}). \quad (2.15)$$

The true cost is therefore bounded below by the scaled Euclidean distance:

$$h_k^* \geq (1 - \mu_2)d_2(k, \mathcal{G}), \quad (2.16)$$

To ensure admissibility, we need:

$$h_k \leq h_k^*. \quad (2.17)$$

Since $1 - \mu_2 \leq 1$, we have:

$$h_k = (1 - \mu_2)d_2(k, \mathcal{G}) \leq d_2(k, \mathcal{G}). \quad (2.18)$$

or,

$$(1 - \mu_2)d_2(k, \mathcal{G}) \leq h_k^*. \quad (2.19)$$

proving that the heuristic is admissible. This ensures that A^* with the modified heuristic remains optimal, as it never overestimates the cost to the goal.

2.3.3 Execution Framework of the Tactical A^* Algorithm

Algorithm 1 Pre A^* Search: Heuristic Precomputation

```

1: Input: 2D occupancy grid  $\mathfrak{M}^\mathcal{O} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ , goal set  $\mathcal{G}$ , parameters  $\mu_1, \mu_2, \mu_3$ , scan limit  $\epsilon$ 
2: Output: Heuristic map  $h : \mathfrak{M}^\mathcal{O} \rightarrow \mathbb{R}_{\geq 0}$ ,  $\kappa$  map  $\kappa : \mathfrak{M}^\mathcal{O} \rightarrow (0, 1]$ 
3: function PREASTAR( $\mathfrak{M}^\mathcal{O}, \mathcal{G}, \mu_1, \mu_2, \mu_3, \epsilon$ )
4:   for each node  $k \in \mathfrak{M}^\mathcal{O}$  do
5:     if  $k$  is not an obstacle then
6:        $\alpha_k \leftarrow \min_{o \in \mathcal{O}} d_2(k, o)$  within distance  $\epsilon$  ▷ Scan for nearest obstacle
7:        $\kappa(k) \leftarrow 1 - \mu_1 e^{4\mu_1\mu_3 - (\mu_3\alpha_k + \mu_1/\alpha_k)^2}$  ▷ Compute  $\kappa$  for node  $k$ 
8:        $h(k) \leftarrow (1 - \mu_2)d_2(k, \mathcal{G})$  ▷ Compute heuristic scaled by  $\mu_2$ 
9:     else
10:       $h(k) \leftarrow \infty$  ▷ Obstacles have infinite heuristic cost
11:       $\kappa(k) \leftarrow 1$  ▷ Default  $\kappa$  for obstacles
12:    end if
13:  end for
14:  return  $h, \kappa$ 
15: end function

```

The tactical A^* algorithm computes an optimal path from a start node r_0 to a goal set $\mathcal{G}$ on a 2D occupancy grid $\mathfrak{M}^\mathcal{O} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$, integrating stealth-inducing behavior via the $\kappa(\alpha)$ function and μ_2 parameter. The algorithm initializes a priority queue with the

start node, setting $g(r_0) = 0$ and computing its heuristic $h(r_0)$ as per Equation (2.5). The start node is inserted into the queue along with its f -cost given by, $f = g + h$, where g is scaled by $\kappa(\alpha)$ to favor obstacle proximity, and h is modulated by μ_2 to balance goal pursuit. Other nodes are initialized with infinite g -costs and h -costs. A closed set, initially empty, tracks processed nodes.

The algorithm iterates until a node in $\mathcal{G}$ is found, removing the node with the lowest f -cost, marking it closed, and recording its parent for backtracking. It expands this node's eight neighbors in $\mathfrak{M}^{\mathcal{O}}$: obstacle or closed nodes are skipped; for others, the tentative g -cost, scaled by $\kappa(\alpha)$, is computed. Existing queue nodes are updated if a lower g -cost is found, adjusting parents; new nodes are inserted into the queue. Upon reaching $\mathcal{G}$, the path $\mathscr{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$ is reconstructed by backtracking using parent pointers, ensuring a tactical, optimal sequence of waypoints. This 2D path is then projected onto the terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, b] \times [0, d], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$, where b, d represent the breadth and depth of the occupancy grid, by performing a nearest neighbor search for each waypoint $r_i = (x_i, y_i)$, mapping it to the closest 3D point $(x_j, y_j, z_j) \in \mathfrak{M}^{\mathcal{T}}$, yielding a 3D path that accounts for terrain elevation.

Algorithm 2 Tactical A^* Search

```

1: Input: 2D occupancy grid  $\mathfrak{M}^\mathcal{O} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ , start node  $r_0$ , goal set  $\mathcal{G}$ , heuristic
   map  $h$ ,  $\kappa$  map  $\kappa$ 
2: Output: Reference path  $\mathscr{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$ 
3: function TACTICALASTARSEARCH( $\mathfrak{M}^\mathcal{O}, r_0, \mathcal{G}, h, \kappa$ )
4:   Initialize priority queue  $Q \leftarrow \{r_0\}$ , closed set  $C \leftarrow \emptyset$ ,  $g(r_0) \leftarrow 0$ ,  $f(r_0) \leftarrow h(r_0)$ 
5:   for each node  $k \neq r_0$  in  $\mathfrak{M}^\mathcal{O}$  do
6:      $g(k) \leftarrow \infty$ 
7:   end for
8:   while  $Q \neq \emptyset$  do
9:      $u \leftarrow$  Pop node with lowest  $f$ -cost from  $Q$ 
10:    if  $u \in \mathcal{G}$  then
11:      break
12:    end if
13:    Add  $u$  to  $C$ 
14:    for each neighbor  $v$  in eight directions from  $u$  do
15:      if  $v$  is not obstacle and  $v \notin C$  then
16:         $g_{\text{tentative}} \leftarrow g(u) + \kappa(v)d_2(u, v)$ 
17:        if  $v \notin Q$  or  $g_{\text{tentative}} < g(v)$  then
18:           $g(v) \leftarrow g_{\text{tentative}}$ ,  $f(v) \leftarrow g(v) + h(v)$ , parent of  $v \leftarrow u$ 
19:          Update or push  $v$  into  $Q$  with priority  $f(v)$ 
20:        end if
21:      end if
22:    end for
23:  end while
24:  if  $u \notin \mathcal{G}$  then
25:    return  $\emptyset$ 
26:  end if
27:   $\mathscr{W} \leftarrow$  Backtrack from  $u$  to  $r_0$  using parents
28:  return  $\mathscr{W}$ 
29: end function

```

2.3.4 Numerical Results for Tactical A^* Planner

The tactical A^* path planning algorithm, as delineated in the preceding sections, has been rigorously evaluated through simulations in a virtual environment, depicted in Figure 2.2, generated using Blender [33]. The environment, spanning a 500 m $\times$ 500 m area, emulates

the irregular terrain characteristics of an off-road region, featuring uneven topography with hills and strategically distributed obstacles that the UGV can coast for stealth. The start point, marked by a green flag, and the goal set $\mathcal{G}$, indicated by a red circle, are positioned to require navigation through the complex terrain, testing the algorithm’s ability to balance stealth and efficiency.

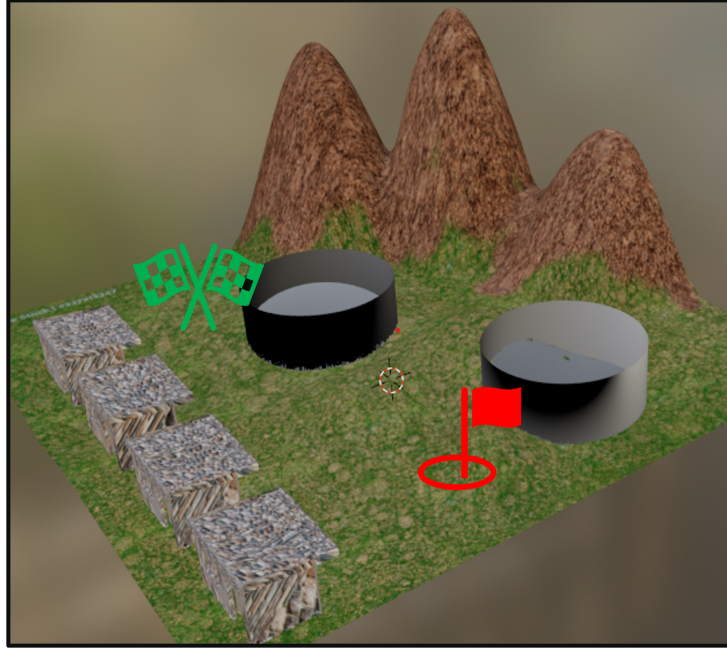


Figure 2.2: Simulated environment for path planning algorithm testing: The Blender-created world features uneven terrain and deliberately placed obstacles, with the green flag indicating the start point and the red circle marking the goal. This environment is used to validate the UGV’s path planning and obstacle avoidance algorithms.

To facilitate realistic testing, the 3D environment was converted from an *.stl* object into a 2D occupancy grid, $\mathfrak{M}^{\mathcal{O}} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0,1\}$, and a terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, 500] \times [0, 500], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$, mimicking the data processing pipeline of an onboard navigation system. The algorithm was implemented in C++ and executed on a system running Ubuntu 20.04, powered by an Intel Core i7-9750H CPU.

Table 2.1: Comparison of the length of traversal for varying levels of recklessness and covert-ness.

Mission	μ_1	μ_2	μ_3	Path length
1	2.00	0.00	2.00	456 m
2	1.50	0.25	1.00	507 m
3	0.10	0.50	0.50	492 m
4	0.50	0.60	0.10	509 m
5	1.00	0.90	0.70	516 m

Five mission scenarios were analyzed, each defined by a parameter triplet (μ_1, μ_2, μ_3) , as shown in Table 2.1, ranging from reckless (Mission 1: $\mu_1 = 2.00, \mu_2 = 0.00, \mu_3 = 2.00$) to highly tactical (Mission 5: $\mu_1 = 1.00, \mu_2 = 0.90, \mu_3 = 0.70$).

The computed paths, visualized in Figure 2.3 (3D view) and Figure 2.4 (top view), illustrate the UGV's trajectory from the start to the goal, with waypoints color-coded by mission: red for Mission 1 (reckless) to white for Mission 5 (stealthy). The visual perspective reveals a clear trend: as μ_2 increases, the UGV's path increasingly coasts the cylindrical shelters, reducing exposure to detection. In Mission 1, the red waypoints take a direct route, cutting through open spaces and avoiding detours. In contrast, Mission 5's white waypoints weave closely around the shelters, exploiting their cover to minimize visibility, as seen in the top view where the path hugs the cylindrical structures. The shelter-seeking behavior, driven by the $\kappa(\alpha)$ function, is quantified in Figure 2.5, which plots the distance between the path and the nearest obstacle $\mathcal{O}$ across waypoints. For Mission 1, the distance peaks at over 90 m in open areas, indicating high exposure risk, while Mission 5 maintains a distance below 20 m for most waypoints, often dipping to 5 m near shelters, confirming its stealth-oriented approach. This behavior reduces the UGV's detectability, a critical advantage in hostile environments, but introduces variability in the path's proximity to obstacles. The graph shows that tactical missions (*e.g.*, Mission 5) exhibit more frequent oscillations in distance, reflecting detours around shelters, which may challenge the UGV's control system

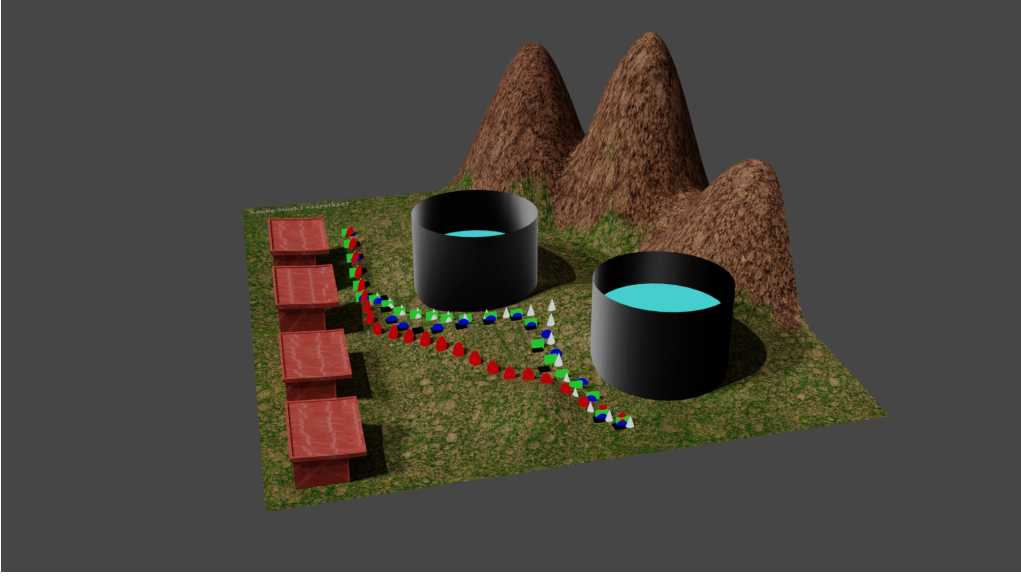


Figure 2.3: Results of tactical path planning simulation: The visual shows the computed path in five scenarios ranging from extreme recklessness shown by the red markers to extreme levels of stealth as shown by the white markers.

by requiring tighter navigation maneuvers. Conversely, reckless missions (*e.g.*, Mission 1) yield smoother paths with fewer directional changes, potentially simplifying control but increasing vulnerability.

Path length, reported in Table 2.1, varies significantly across missions, from 456 m in Mission 1 to 516 m in Mission 5, a 13% increase. This trade-off reflects the cost of stealth: tactical missions elongate the path to coast shelters, increasing the Euclidean distance traveled. Assuming a constant velocity $v = 5$ m/s, the travel time for Mission 1 is $t_1 = 456/5 = 91.2$ s, while for Mission 5, it rises to $t_5 = 516/5 = 103.2$ s, an additional 12 seconds. Similarly, fuel consumption, proportional to distance, increases by 13% (*e.g.*, $F \propto L$), impacting mission logistics. However, this cost is justified by the reduced exposure risk, as tactical paths minimize the time spent in open areas, a key consideration for military operations where detection could be catastrophic.

Computational efficiency further supports the algorithm’s practical applicability. Across

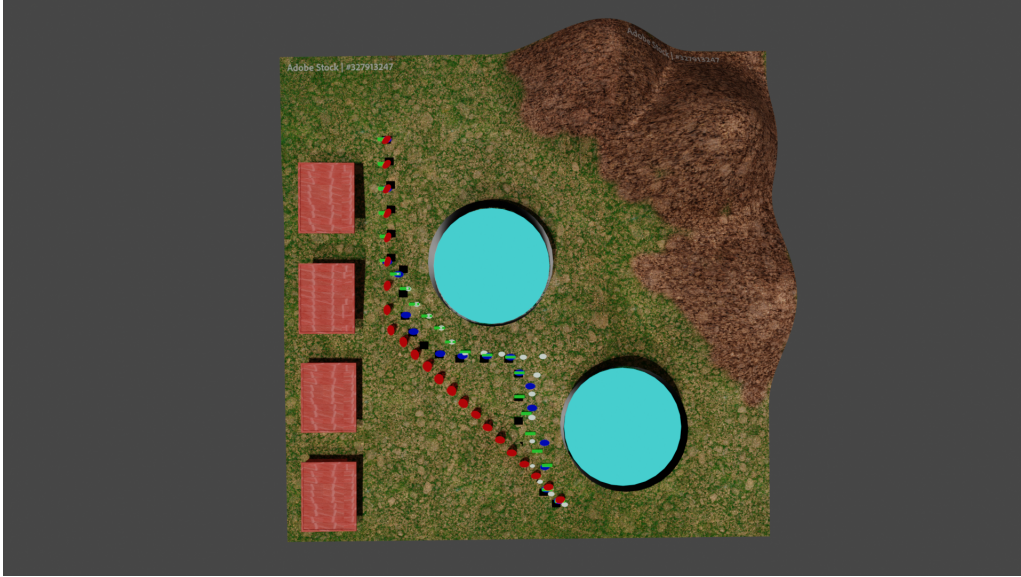


Figure 2.4: Top view of tactical path planning simulation results: The overhead perspective shows the UGV’s computed paths through five scenarios, with red markers indicating extreme recklessness and white markers representing extreme stealth by seeking shelter around the obstacles.

50 runs (10 per mission), the average runtime was 9436 microseconds (9.4 ms) on the Intel Core i7-9750H CPU, well within real-time constraints for UGV navigation. This efficiency, combined with the algorithm’s ability to navigate complex terrain while maintaining stealth, underscores its potential for real-world deployment. The results demonstrate a clear balance: reckless missions optimize for distance and simplicity at the expense of safety, while tactical missions prioritize stealth, accepting longer paths and increased navigational complexity. These insights inform mission planning, allowing operators to tailor parameters based on priorities—efficiency versus safety—while ensuring robust performance in dynamic, threat-prone environments.

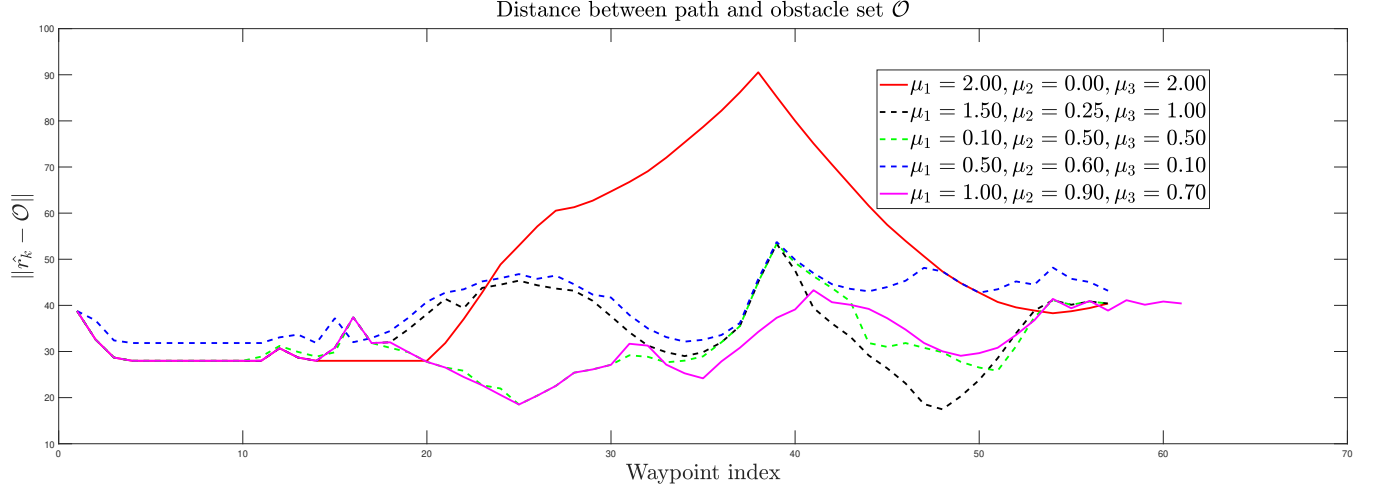


Figure 2.5: Distance between UGV path and obstacles for different tactical parameters: The graph shows the trade-off between distance traveled versus stealth. The more stealthy the path, the more distance the UGV travels.

2.4 Challenges of A^* in Dynamic Environments with Incremental Map-Building

The classical A^* algorithm, as detailed in Section 2.3.3, computes an optimal path from a start node r_0 to a goal set $\mathcal{G}$ on a 2D occupancy grid $\mathfrak{M}^\mathcal{O} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$, leveraging heuristic knowledge to focus the search. However, in the context of UGV navigation without an a priori map, where the map is built incrementally as new obstacles are discovered, A^* exhibits significant limitations. As the UGV explores the environment, the occupancy grid $\mathfrak{M}^\mathcal{O}$ evolves with newly detected obstacles (*e.g.*, shelters or barriers), necessitating replanning to adapt the path $\mathscr{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$. Whereas A^* can replan by restarting the search from scratch each time the map changes, this approach is computationally inefficient. Each replanning iteration recomputes the entire search tree, disregarding previous search efforts, leading to redundant vertex expansions. For a grid of size $n \times m$, the worst-case time complexity of A^* is

$O(nm \log(nm))$ per search, assuming a binary heap priority queue. In dynamic environments with frequent updates—such as discovering new obstacles every few seconds—this repeated computation severely impacts responsiveness, delaying the UGV’s ability to adapt to changes in real-time, a critical requirement for navigation in hostile, unpredictable terrains.

To address the inefficiencies of A^* in dynamic environments, incremental search methods offer a promising alternative by reusing information from previous searches to accelerate replanning. Several such methods have been proposed in the literature, each tailored to handle changes in the graph (*e.g.*, edge cost changes, vertex additions, or deletions) more efficiently than replanning from scratch. Lifelong Planning A^* (LPA^*) [32], an incremental version of A^* , reuses parts of the previous search tree that remain valid after map updates, focusing only on vertices affected by changes. D^* (Dynamic A^*) [34] and its variant D^* Lite [35] are other notable incremental methods, widely used in robotics for real-time navigation. D^* searches backward from the goal to the start, updating costs as the environment changes, and is particularly effective for robot navigation in unknown terrain. D^* Lite, a simplified version, combines D^* ’s incremental updates with heuristic search, offering improved efficiency for goal-directed navigation.

Among the incremental search methods, LPA^* serves a dual advantage—incremental updates and heuristic guidance—making LPA^* more efficient than D^* and D^* Lite in scenarios with frequent, small changes, as it avoids the overhead of backward searches (D^*) and simplifies the update process (D^* Lite). Furthermore, LPA^* ’s integration with the tactical $\kappa(\alpha)$ function, as used in our framework, allows seamless adaptation of stealth-inducing behavior, ensuring the UGV coasts shelters even as the map evolves, a critical feature for hostile environments.

2.5 Proposed Global Tactical LPA^* Path Planner

The Lifelong Planning A^* (LPA^*) algorithm extends the classical A^* framework to address the dynamic nature of UGV navigation in environments where the map is built incrementally, as detailed in [32]. In the context of this research, LPA^* computes an optimal path from a start node r_0 to a goal set $\mathcal{G}$ on a 2D occupancy grid $\mathfrak{M}^O : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$, adapting to newly discovered obstacles without requiring a complete recomputation of the search tree. Unlike A^* , which replans from scratch, LPA^* reuses information from previous searches, making it particularly suited for real-time navigation in hostile environments where the map evolves as the UGV explores. The algorithm operates on a finite graph defined by the grid, where vertices represent grid cells, and edges connect neighboring cells in eight directions, with costs $c(u, v)$ reflecting the traversal cost between vertices u and v , scaled by the tactical $\kappa(\alpha)$ function to induce stealth-inducing behavior.

The cost function in LPA^* mirrors that of A^* , defined as:

$$f_k \triangleq g_k + h_k, \quad k \in \{0, \dots, n_p\}, \quad (2.20)$$

where g_k represents the cost-to-come from the start node to node k , h_k is the heuristic cost-to-go estimating the cost from k to $\mathcal{G}$, and n_p is the number of waypoints. The functions g_k and h_k are the same as those in A^* (equations (2.4) and (2.5)), scaled by $\kappa(\alpha)$ and μ_2 respectively, with their optimality and admissibility established in Section 2.3.2. LPA^* prioritizes nodes in a queue based on a key $k(s) = [\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))]$, compared lexicographically, where $rhs(s)$ is a one-step lookahead estimate of the start distance, distinct from the g -value.

A key concept in LPA^* is local consistency. A node s is *locally consistent* if its g -value equals its rhs -value, i.e., $g(s) = rhs(s)$, indicating that the node's cost-to-come aligns with

the best possible cost through its predecessors [32]. Conversely, a node is *locally inconsistent* if $g(s) \neq rhs(s)$, signaling a discrepancy that LPA^* must resolve to ensure path optimality. The rhs -value is defined as:

$$rhs(s) = \begin{cases} 0 & \text{if } s = r_0, \\ \min_{s' \in \text{pred}(s)} (g(s') + c(s', s)) & \text{otherwise,} \end{cases} \quad (2.21)$$

where $\text{pred}(s)$ denotes the predecessors of s , and $c(s', s)$ is the $\kappa(\alpha)$ -scaled cost between s' and s . Unlike the g -value, which represents the accumulated cost along the current search tree path to s , the rhs -value provides a lookahead estimate based on the g -values of the predecessors of s , making it potentially more informed and enabling LPA^* to identify inconsistencies caused by map updates without recomputing unaffected regions.

2.5.1 Execution Framework of the Tactical LPA^* Algorithm

The tactical LPA^* algorithm initializes a priority queue with the start node r_0 , setting $g(r_0) = \infty$ and $rhs(r_0) = 0$, making r_0 locally inconsistent ($g(r_0) \neq rhs(r_0)$). All other nodes are initialized with infinite g - and rhs -values, and their heuristics h are precomputed as per Section 2.3.3. The priority queue contains locally inconsistent nodes, prioritized by the key $k(s) = [\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))]$, compared lexicographically [32]. The algorithm iterates by removing the node u with the smallest key, adjusting its g -value to resolve inconsistencies: if $g(u) > rhs(u)$, $g(u)$ is set to $rhs(u)$; if $g(u) < rhs(u)$, $g(u)$ is set to ∞ . It then updates the rhs -values of u 's successors, adding or removing them from the queue if their local consistency changes.

Unlike A^* , which continues until the priority queue is empty or the goal is reached, LPA^* terminates when the goal node $s \in \mathcal{G}$ is locally consistent ($g(s) = rhs(s)$) and its key

is no less than the smallest key in the queue, ensuring the path to $\mathcal{G}$ is optimal without expanding unnecessary nodes [32]. When the map $\mathfrak{M}^{\mathcal{O}}$ updates (*e.g.*, new obstacles are detected), LPA^* adjusts the *rhs*-values of affected nodes, reinserts them into the queue if inconsistent, and resumes the loop, avoiding A^* 's full recomputation. Upon termination, the path $\mathscr{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$ is reconstructed by backtracking from s to r_0 using parent pointers, following predecessors that minimize $g(s') + c(s', s)$. This 2D path is then projected onto the terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, b] \times [0, d], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$, where b, d represent the width and depth of the occupancy grid, mapping each waypoint $r_i = (x_i, y_i)$ to the closest 3D point $(x_j, y_j, z_j) \in \mathfrak{M}^{\mathcal{T}}$ via nearest neighbor search, yielding a 3D path accounting for terrain elevation.

Algorithm 3 Pre-Search for LPA^* : Initialization

```

1: Input: 2D occupancy grid  $\mathfrak{M}^{\mathcal{O}} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ , start node  $r_0$ , goal set  $\mathcal{G}$ , parameters
    $\mu_1, \mu_2, \mu_3$ , scan limit  $\epsilon$ 
2: Output: Heuristic map  $h : \mathfrak{M}^{\mathcal{O}} \rightarrow \mathbb{R}_{\geq 0}$ ,  $\kappa$  map  $\kappa : \mathfrak{M}^{\mathcal{O}} \rightarrow (0, 1]$ , initial  $g$  and rhs values
3: function PRELPASTAR( $\mathfrak{M}^{\mathcal{O}}, r_0, \mathcal{G}, \mu_1, \mu_2, \mu_3, \epsilon$ )
4:   for each node  $k \in \mathfrak{M}^{\mathcal{O}}$  do
5:     if  $k$  is not an obstacle then
6:        $\alpha_k \leftarrow \min_{o \in \mathcal{O}} d_2(k, o)$  within distance  $\epsilon$  ▷ Scan for nearest obstacle
7:        $\kappa(k) \leftarrow 1 - \mu_1 e^{4\mu_1\mu_3 - (\mu_3\alpha_k + \mu_1/\alpha_k)^2}$  ▷ Compute  $\kappa$  for node  $k$ 
8:        $h(k) \leftarrow (1 - \mu_2)d_2(k, \mathcal{G})$  ▷ Compute heuristic scaled by  $\mu_2$ 
9:     else
10:       $h(k) \leftarrow \infty$  ▷ Obstacles have infinite heuristic cost
11:       $\kappa(k) \leftarrow 1$  ▷ Default  $\kappa$  for obstacles
12:     end if
13:      $g(k) \leftarrow \infty$  ▷ Initialize  $g$ -values
14:     if  $k = r_0$  then
15:        $rhs(k) \leftarrow 0$  ▷ Set rhs for start node
16:     else
17:        $rhs(k) \leftarrow \infty$  ▷ Initialize other rhs-values
18:     end if
19:   end for
20:   return  $h, \kappa, g, rhs$ 
21: end function

```

Algorithm 4 Tactical LPA^* Search

```

1: Input: 2D occupancy grid  $\mathfrak{M}^O : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ , terrain map  $\mathfrak{M}^T$ , start node  $r_0$ ,
   goal set  $\mathcal{G}$ , heuristic map  $h$ ,  $\kappa$  map  $\kappa$ , initial  $g$  and  $rhs$  values
2: Output: Reference path  $\mathcal{W} \triangleq \{r_i\}_{i \in \mathbb{W}}$ 
3: function TACTICALLPASTARSEARCH( $\mathfrak{M}^O, \mathfrak{M}^T, r_0, \mathcal{G}, h, \kappa, g, rhs$ )
4:   Initialize priority queue  $Q \leftarrow \{(r_0, [\min(g(r_0), rhs(r_0)) + h(r_0); \min(g(r_0), rhs(r_0))])\}$ 
5:   while  $Q \neq \emptyset$  and not ( $u \in \mathcal{G}$  and  $g(u) = rhs(u)$  and  $Q.TopKey() \geq$ 
    $[\min(g(u), rhs(u)) + h(u); \min(g(u), rhs(u))])$  do
6:      $u \leftarrow Q.Pop()$ 
7:     if  $g(u) > rhs(u)$  then
8:        $g(u) \leftarrow rhs(u)$ 
9:     else
10:       $g(u) \leftarrow \infty$ 
11:    end if
12:    for each neighbor  $v$  of  $u$  in 8-connected grid do
13:      if  $v$  is not an obstacle then
14:         $rhs(v) \leftarrow \min_{s' \in \text{pred}(v)} (g(s') + \kappa(v)d_2(s', v))$ 
15:        if  $g(v) \neq rhs(v)$  then
16:           $Q.Insert(v, [\min(g(v), rhs(v)) + h(v); \min(g(v), rhs(v))])$ 
17:        else
18:           $Q.Remove(v)$ 
19:        end if
20:      end if
21:    end for
22:    if map  $\mathfrak{M}^O$  updates then
23:      for each affected node  $v$  do
24:         $rhs(v) \leftarrow \min_{s' \in \text{pred}(v)} (g(s') + \kappa(v)d_2(s', v));$  Update  $Q$ 
25:      end for
26:    end if
27:  end while
28:  if  $u \notin \mathcal{G}$  then
29:    return  $\emptyset$ 
30:  end if
31:   $\mathcal{W} \leftarrow$  Backtrack from  $u$  to  $r_0$ ; Map to  $\mathfrak{M}^T$  via nearest neighbor search
32:  return  $\mathcal{W}$ 
33: end function

```

2.5.2 Data Structure for the Tactical LPA^* Priority Queue

The tactical LPA^* algorithm relies on a priority queue to manage locally inconsistent nodes, prioritized by their keys $k(s) = [\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))]$, ensuring that the node with the smallest key is expanded first during the search. An efficient implementation of this queue is critical for real-time performance in large grids, such as the $500 \text{ m} \times 500 \text{ m}$ environment used in our simulations. Following the approach in [32], we adopt a *binary min-heap* as the data structure for the priority queue. In this structure, nodes are organized in a complete binary tree, where each parent’s key is less than or equal to its children’s keys, guaranteeing that the root holds the node with the smallest key.

The binary min-heap supports three key operations required by LPA^* : *extract-min* (removing the node with the smallest key), *insert* (adding a new node when it becomes locally inconsistent), and *decrease-key* (updating a node’s key when its g or rhs value changes due to map updates). For a heap with n nodes, these operations have a time complexity of $O(\log n)$, achieved through percolation procedures that maintain the heap property after each operation. This logarithmic complexity ensures efficient handling of large grids, where the number of nodes can be on the order of $n \times m$ (e.g., $500 \times 500 = 250,000$ cells in our setup). The binary min-heap’s simplicity and predictable performance make it well-suited for real-time UGV navigation, where frequent map updates (e.g., new obstacles detected every few seconds) necessitate rapid adjustments to the queue.

Whereas the binary min-heap is a robust choice, it is not the only viable data structure for LPA^* ’s priority queue. Alternatives include the Fibonacci heap, which offers an amortized $O(1)$ time for decrease-key operations, compared to $O(\log n)$ for a binary min-heap, but at the cost of increased implementation complexity and higher constant factors, making it less practical for typical path planning scenarios. A pairing heap provides a simpler alternative to

the Fibonacci heap with good practical performance, though its theoretical bounds are less favorable. A sorted list, while straightforward, incurs $O(n)$ time for insert and extract-min operations, rendering it inefficient for large grids. Given the balance of efficiency, simplicity, and practical performance, the binary min-heap remains the preferred choice for our tactical LPA^* implementation, ensuring responsive path planning in dynamic, hostile environments.

2.5.3 Numerical Results for Tactical LPA^* Planner

The tactical LPA^* path planner has been evaluated through simulations in a virtual environment shown in Figure 2.6, generated using Blender [33]. This environment, spanning a $500 \text{ m} \times 500 \text{ m}$ area, emulates off-road terrain with uneven topography similar to the setup in Section 2.3.4 but with updated obstacle placements to further challenge path planning. The UGV navigates from a start point to a goal set $\mathcal{G}$, a disk of radius 5 m within a 2D occupancy grid $\mathfrak{M}^{\mathcal{O}} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ and a terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, 500] \times [0, 500], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$. The algorithm was implemented in C++ on Ubuntu 20.04, using an Intel Core i7-9750H CPU, consistent with prior experiments.

Simulations were conducted across five stealth levels (0 to 4), each defined by the parameter triplet (μ_1, μ_2, μ_3) from Table 2.1, ranging from reckless (Level 0: $\mu_1 = 2.00, \mu_2 = 0.00, \mu_3 = 2.00$) to highly tactical (Level 4: $\mu_1 = 1.00, \mu_2 = 0.90, \mu_3 = 0.70$) as shown in Figure 2.7. Both LPA^* and A^* were evaluated to compare their performance in this static, fully known map scenario, revealing comparable path lengths, which suggests that LPA^* 's incremental updates do not compromise path quality in static conditions. However, the primary focus of this study is LPA^* 's applicability in dynamic environments, where its ability to adapt to incremental map updates provides a significant advantage. The results are cat-

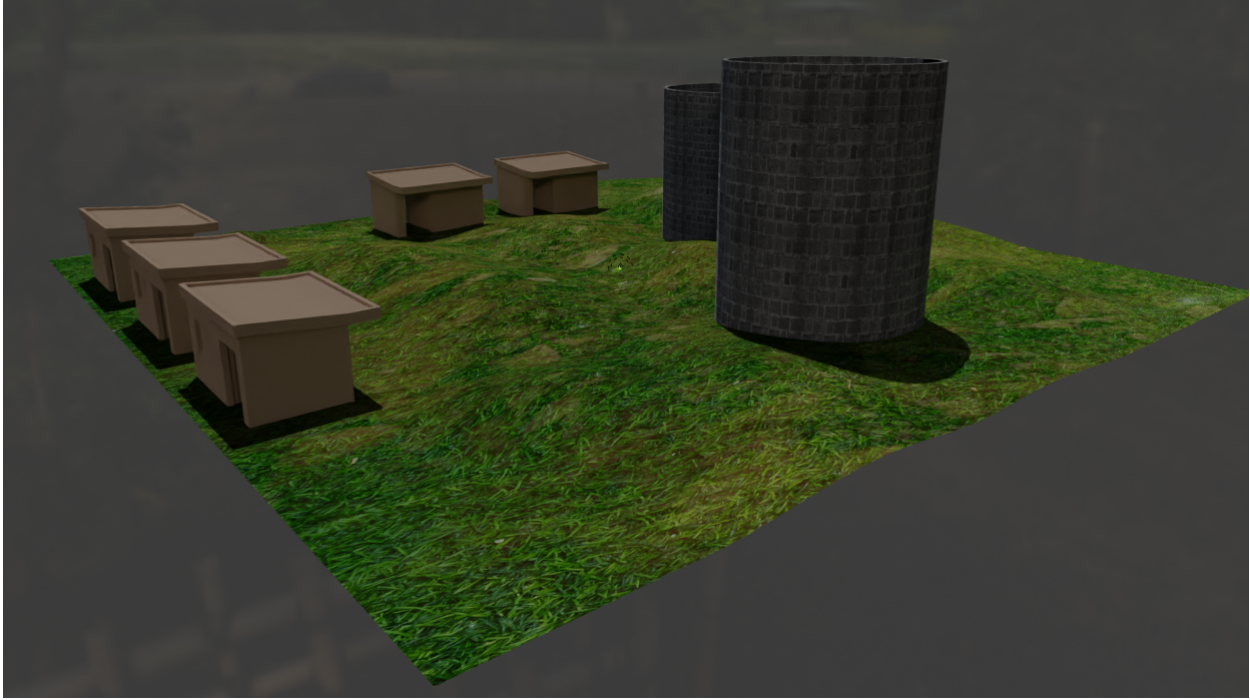


Figure 2.6: Virtual test environment developed in Blender for evaluating tactical and terrain-aware navigation strategies based on Lifelong Planning A* (LPA^*). The simulation incorporates realistic terrain undulations and heterogeneous obstacles representing shelters

egorized into path length, shelter proximity, and path entropy, analyzed below to highlight their tactical implications.

Path Length Analysis

The path lengths for LPA^* and A^* across stealth levels are compared in Figure 2.8. At Stealth Level 0 (reckless), LPA^* yields a path length of 451.51 m, while A^* yields 464.37 m. As stealth increases, path lengths grow: at Level 1, LPA^* is 452.98 m, A^* is 462.62 m; at Level 2, LPA^* is 521.40 m, A^* is 492.94 m; at Level 3, LPA^* is 600.12 m, A^* is 590.47 m; and at Level 4 (most tactical), LPA^* reaches 632.42 m, A^* reaches 597.10 m. The trend mirrors prior findings, with path lengths increasing by 40.1% for LPA^* as stealth level rises from 0 to 4. This increase reflects the UGV's detours to coast shelters, prioritizing stealth

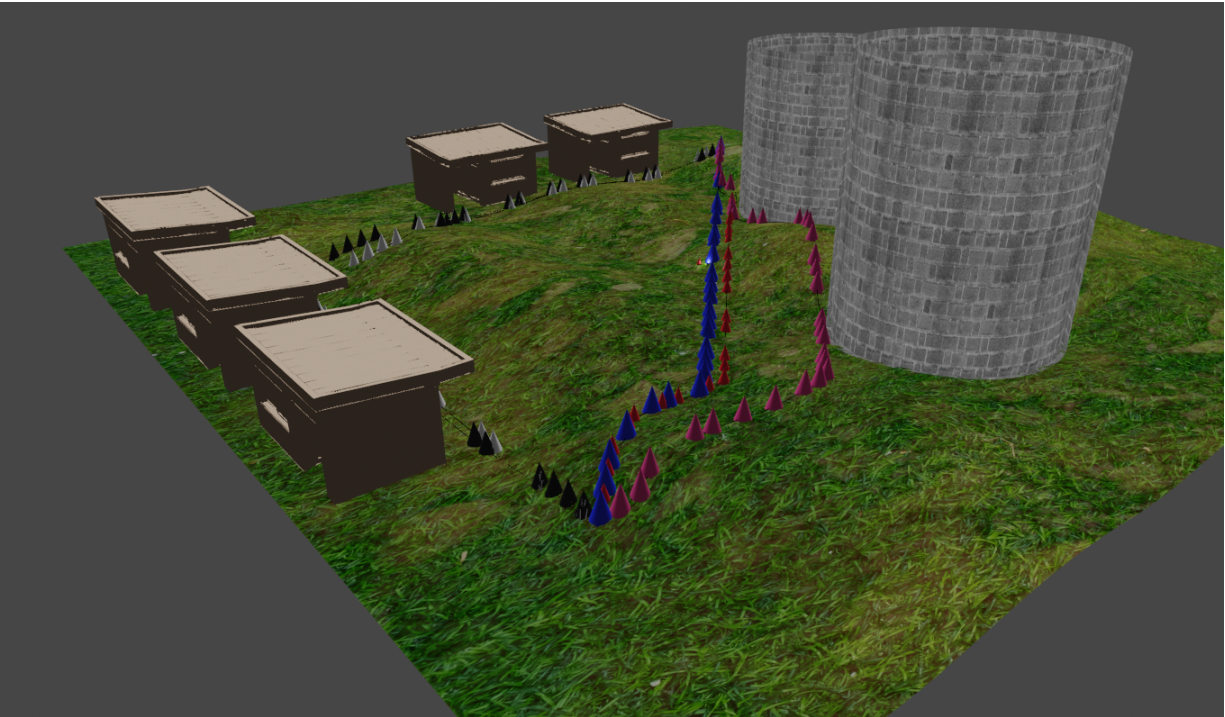


Figure 2.7: Simulated paths using LPA^* . The black path represents the most tactical trajectory, prioritizing concealment and obstacle cover. The blue path reflects a reckless strategy, favoring direct goal pursuit with minimal regard for tactical considerations. The magenta path demonstrates a semi-tactical approach, balancing stealth and goal-directedness

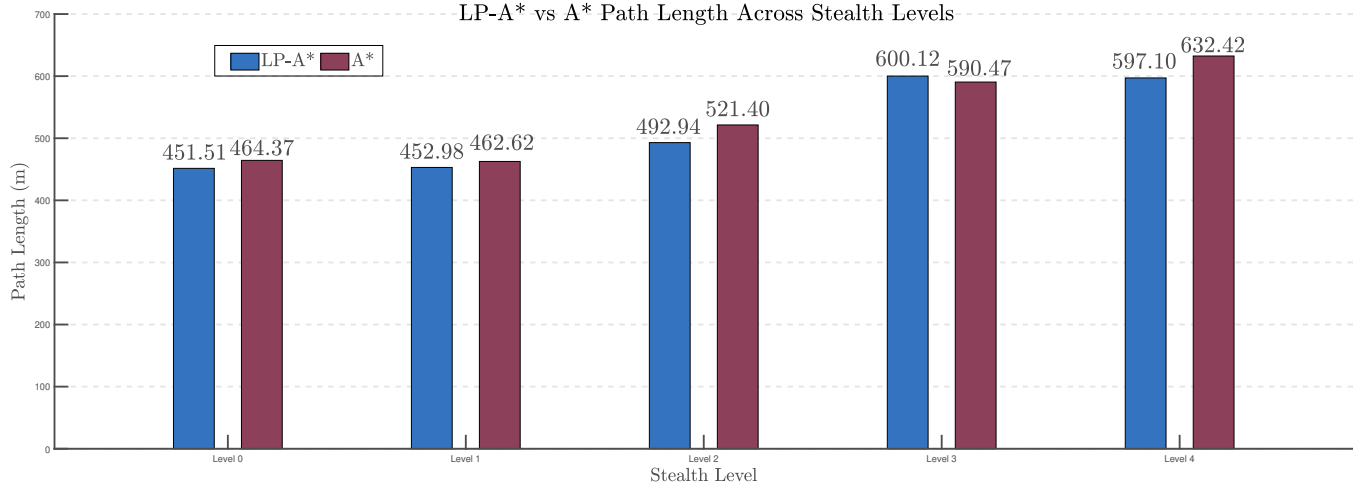


Figure 2.8: Path length comparison between LPA^* and A^* across stealth levels: The bar chart shows increasing path lengths as stealth level rises, reflecting detours to coast shelters.

over distance optimization.

Assuming a constant velocity of 5 m/s, the travel time for LPA^* at Level 0 is $t_0 = 451.51/5 = 90.30$ s, rising to $t_4 = 632.42/5 = 126.48$ s at Level 4—an increase of 36.18 s (40.1%). Fuel consumption, proportional to distance ($F \propto L$), also increases by the same percentages, impacting mission logistics. These results align with earlier findings, confirming that higher stealth levels lead to longer paths, necessitating more time and fuel, a trade-off justified by enhanced safety in hostile environments where minimizing detection is critical.

Shelter Proximity Analysis

The distance to the nearest obstacle, a key indicator of stealth, is analyzed for LPA^* across stealth levels in Figure 2.9, presented as a box plot. At Stealth Level 0, the median distance is approximately 60 m, with a wide interquartile range (IQR) indicating varied proximity to shelters. As stealth level increases, the median distance decreases: at Level 1, it drops to 50 m; at Level 2, to 40 m; at Level 3, to 30 m; and at Level 4, to 20 m, with a

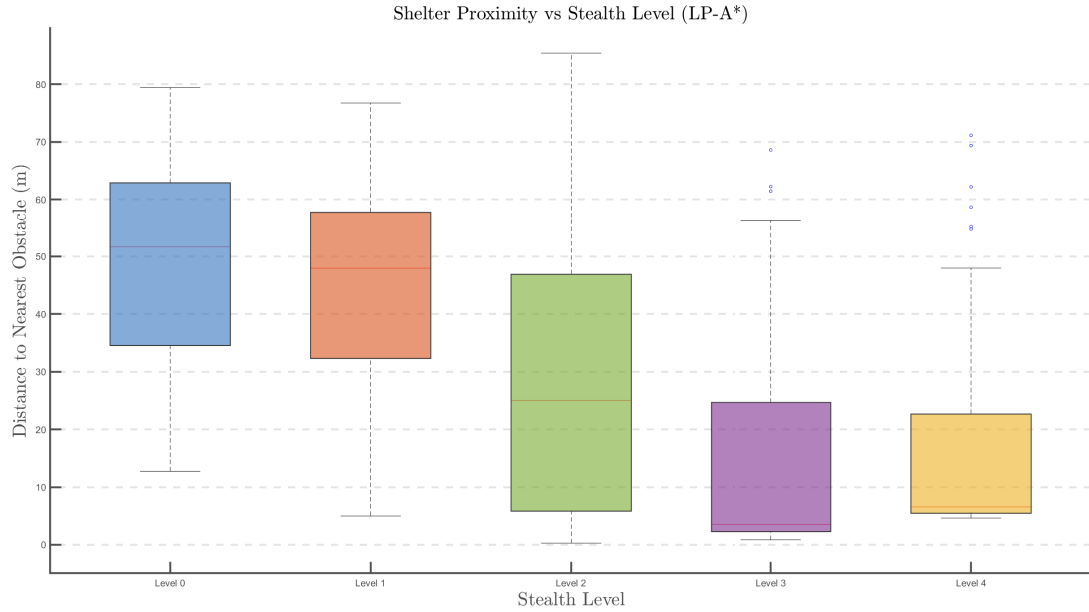


Figure 2.9: Distance to nearest obstacle across stealth levels for LPA^* : The box plot shows decreasing median distances as stealth level increases, indicating closer shelter-coasting behavior.

tighter IQR, reflecting more consistent shelter-coasting behavior. Outliers at higher stealth levels suggest occasional deviations, possibly due to terrain constraints or sparse obstacle placement.

Physically, this trend indicates that the UGV navigates closer to shelters as stealth level increases, reducing its visibility and exposure to detection—a critical advantage in hostile environments. For army personnel, this proximity enhances mission safety by minimizing the UGV’s detectability, allowing it to operate covertly in threat-prone areas. Mathematically, the $\kappa(\alpha)$ function rewards paths closer to obstacles (smaller α), and the box plot confirms this behavior, with median distances aligning with the function’s design to favor tactical paths. However, closer proximity may increase the risk of collision, requiring more precise control systems or enhanced sensors, and could affect navigation stability on uneven terrain, aspects not directly measured here but warranting consideration in future studies.

Path Entropy Analysis

To assess the unpredictability of tactical paths, the entropy, specifically the Shannon entropy, was evaluated for path directions across stealth levels using LPA^* , as shown in Figure 2.10. The Shannon entropy, defined as

$$H = - \sum_{i=1}^n p_i \log_2 p_i, \in [0, \log_2(n)], \quad (2.22)$$

measures uncertainty in bits, where $p_i \in [0, 1]$ represents the probability of event i and $\sum_{i=1}^n p_i = 1$. To this end, the direction vectors along the obtained waypoints were calculated and segregated into 8 discrete bins, corresponding to the 8-connected grid. The probability of direction vectors in each bin was determined, and the entropy was subsequently computed. At Stealth Level 0, the entropy measures approximately 1 bit, indicating relatively predictable motion. As the stealth level increases, the entropy rises: 1.1 bits at Level 1, 1.2 bits at Level 2, 1.4 bits at Level 3, and 1.5 bits at Level 4, reflecting greater directional uncertainty.

Physically, higher entropy indicates that tactical paths involve more frequent and varied direction changes to coast obstacles, making the UGV's motion less predictable. For an opponent using a motion tracking algorithm, this unpredictability disrupts model fitting and prediction convergence, as the increased uncertainty (higher entropy) complicates trajectory forecasting. For army personnel, this is a significant advantage, as it reduces the likelihood of the UGV being tracked, enhancing its survivability in hostile environments. Mathematically, while the $\kappa(\alpha)$ function rewards proximity to obstacles, this study reveals an emergent effect: tactical paths naturally become more chaotic, increasing entropy without explicit optimization for unpredictability. This finding suggests a potential future research direction—deliberately maximizing path entropy to further confuse motion prediction mod-

els, combining shelter-coasting with intentional directional chaos to create paths that are both stealthy and unpredictable, thereby enhancing tactical effectiveness.

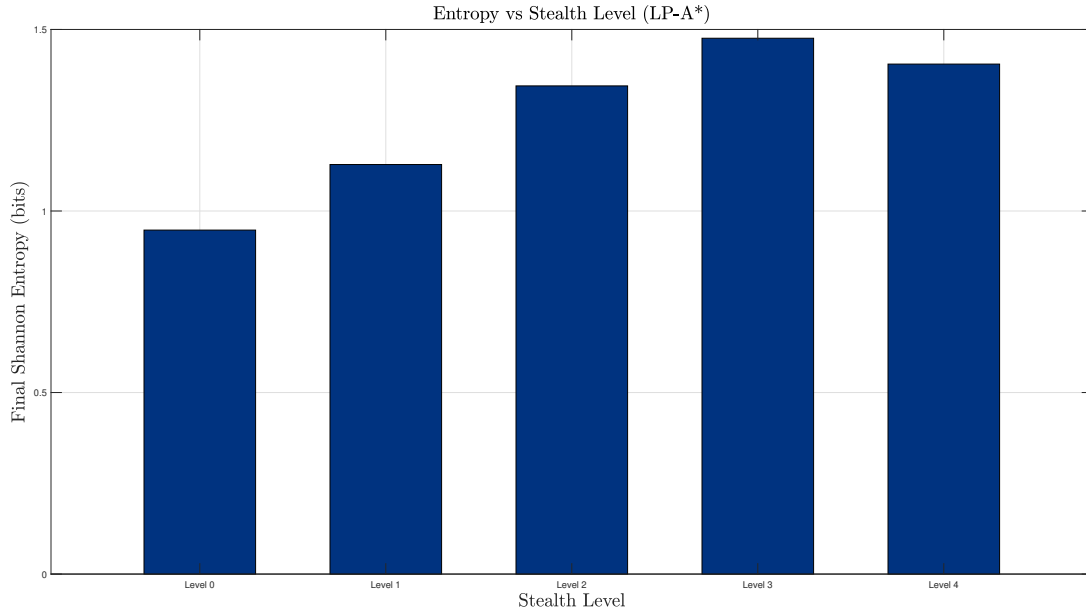


Figure 2.10: Shannon entropy of path directions across stealth levels for LPA^* : The bar chart shows increasing entropy as stealth level rises, indicating greater directional unpredictability.

2.6 Dual mode obstacle interaction in tactical path planning

In the tactical path planning framework developed thus far, obstacles within the environment have been uniformly treated as shelters, leveraging the $\kappa(\alpha)$ function to encourage the UGV to coast them for concealment, as detailed in Sections 2.3.1 and 2.5. However, not all obstacles in hostile environments warrant the same interaction strategy. Certain obstacles, such as walls or structures, serve as safe shelters, providing cover to minimize detection by adversarial sensors. Conversely, other obstacles, such as landmines or explosive traps, pose

significant risks, where proximity could lead to catastrophic failure of the UGV and mission. This section introduces a dual-mode obstacle interaction paradigm, distinguishing between obstacles to coast (*e.g.*, safe shelters) and those to avoid (*e.g.*, landmines), while ensuring the path remains optimal within the LPA^* framework. The implications of this dual-mode approach are explored and the need for heuristic redesign is evaluated in this section.

Consider an environment as shown in Figure 2.11 (3D view) and Figure 2.12 (top view) containing both safe shelters (*e.g.*, walls or structures) and hazardous obstacles (*e.g.*, landmines). Coasting a shelter aligns with the tactical objective of minimizing detection, as the UGV can utilize the shelter as a shield against external threats, reducing its visibility and exposure. However, applying the same coasting strategy to a landmine would be disastrous, as proximity to such an obstacle risks destruction, undermining the mission's success.

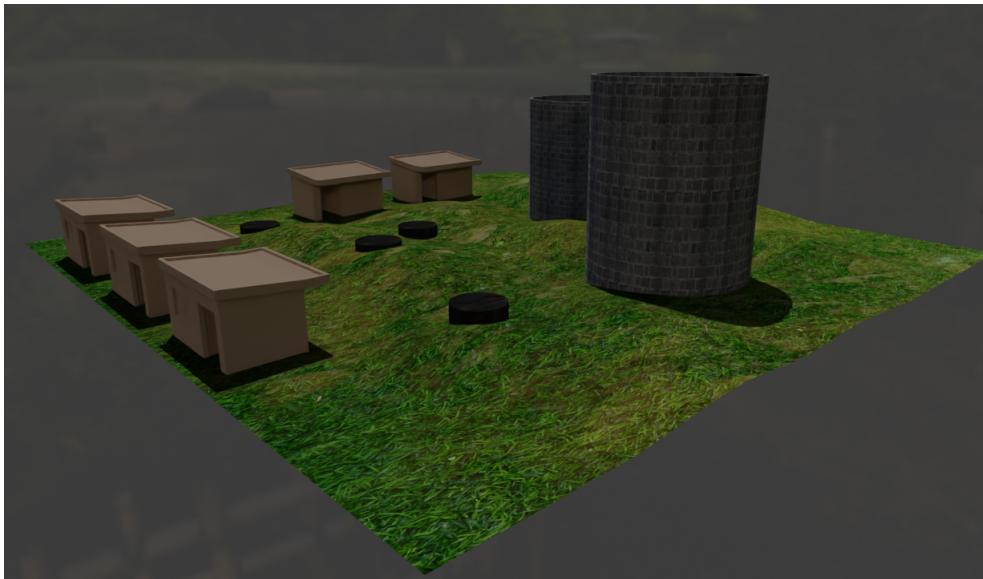


Figure 2.11: Blender generated virtual environment with safe shelters and dangerous mines. This simulated world is used to evaluate the efficiency of the proposed dual obstacle interactive path planner exhibiting both concealment seeking and mine-evasive characteristics

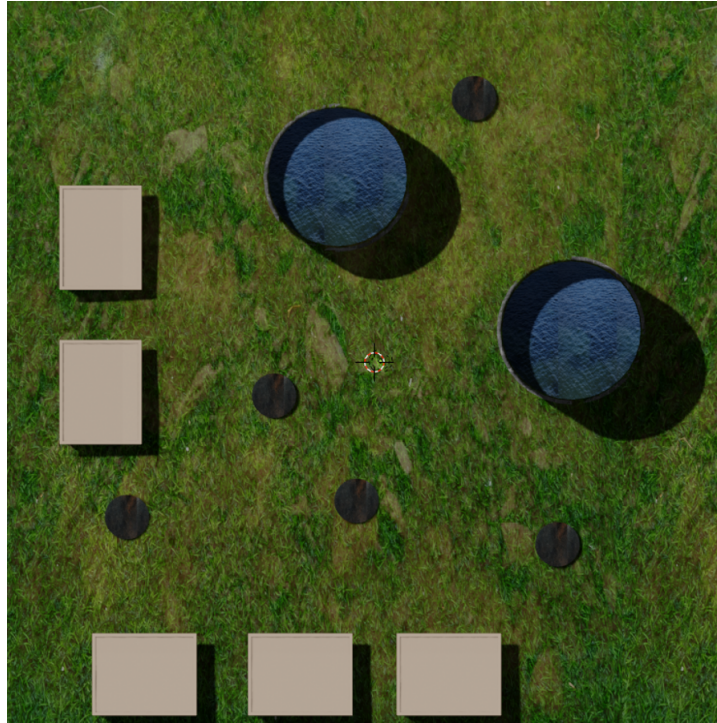


Figure 2.12: Top view of the Blender generated world with safe shelters and mines used to test the dual obstacle interactive path planner

To address this challenge of dual obstacle set interactions, obstacles are first classified into two distinct categories: coasting obstacles $\mathcal{O}_c$ (*e.g.*, shelters) and avoidance obstacles $\mathcal{O}_a$ (*e.g.*, landmines). Coasting obstacles are handled as before, using the $\kappa(\alpha_c)$ function, where $\alpha_c(k) = \min_{o \in \mathcal{O}_c} d_2(k, o)$ denotes the distance to the nearest coasting obstacle for node k , reducing traversal costs to encourage proximity and enhance stealth. Avoidance obstacles, however, require a mechanism to ensure the path never passes near or over them, as even a slight deviation could be fatal.

2.6.1 Heuristic Redesign in Dual-Mode Interaction

Incorporating the dual-mode obstacle interaction into the LPA^* framework requires careful consideration of the heuristic design to balance tactical objectives with computational

efficiency. A comprehensive redesign of the heuristic to explicitly account for obstacle categories—such as adding a penalty term based on proximity to avoidance obstacles $\mathcal{O}_a$ or integrating a probabilistic risk model would introduce significant computational overhead. For instance, computing the distance to the nearest avoidance obstacle for each node would increase the planner’s time complexity to $O(mn)$ per iteration, where m is the number of nodes and n is the grid size. In real-time UGV navigation within hostile environments, this overhead could delay critical path updates, rendering the planner impractical for dynamic settings.

To address this, a simpler approach is adopted: the heuristic cost for nodes associated with avoidance obstacles (*e.g.*, landmines) or their immediate vicinity (*e.g.*, neighboring cells, ensuring a safe buffer) is set to infinity. Specifically, $h_k = \infty$ is assigned to any node $(x, y) \in \mathcal{O}_a$ or within its vicinity, ensuring that the total cost $f_k = g_k + h_k$ becomes infinite. Consequently, such nodes are effectively excluded from the priority queue, as LPA^* prioritizes nodes with the lowest f_k values. For all other nodes, the original heuristic $h_k = (1 - \mu_2)d_2(\hat{r}_k, \mathcal{G})$ is retained, preserving its role in balancing goal pursuit with tactical behavior. This method avoids complex heuristic redesign, maintaining the planner’s computational efficiency while ensuring paths never traverse avoidance obstacles. Since nodes in $\mathcal{O}_a$ are not part of the optimal path, this approach does not violate the principle of admissibility, allowing the planner to operate on a subgraph of the grid where hazardous obstacles are excluded.

2.6.2 Case Study Setup: Mine Evasion in Military Applications

The dual-mode LPA^* algorithm is evaluated in a simulated environment previously established in Section 2.5.3, featuring a 500 m $\times$ 500 m grid with uneven terrain, and safe

shelters, generated using Blender [33]. In this case study, the environment is augmented with mines as shown in Figure 2.12, classified as avoidance obstacles $\mathcal{O}_a$, alongside the existing coasting obstacles $\mathcal{O}_c$ (e.g., shelters). The mines, represented as small black cylinders, are scattered across the grid, posing a hazard that the UGV must avoid. The algorithm is implemented in C++ on Ubuntu 20.04 using an Intel Core i7-9750H CPU, consistent with prior experiments. Nodes associated with mines or their immediate vicinity are assigned an infinite heuristic cost ($h_k = \infty$), ensuring they are excluded from the path, while the $\kappa(\alpha_c)$ function encourages coasting of shelters, as per Equation (2.6).

The results are illustrated in Figures 2.13, comparing reckless and tactical paths with mines. The reckless path (blue) navigates closer to mines, risking destruction, while the tactical path (red) successfully avoids mines by detouring around them, maintaining a safe distance, and coasts shelters to minimize visibility. These results demonstrate the dual-mode approach’s effectiveness: the planner ensures safety by avoiding hazardous obstacles while achieving tactical objectives through shelter-coasting, making it suitable for military applications where both stealth and survivability are critical.

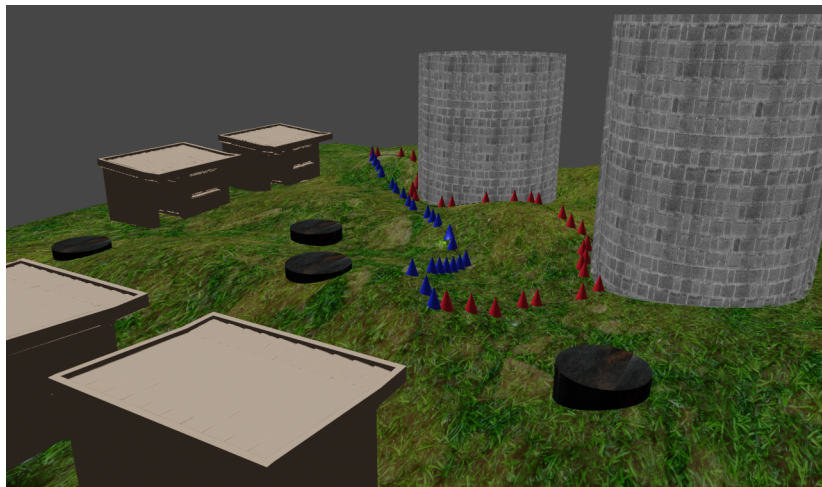


Figure 2.13: Simulated result of the dual obstacle aware LPA^* path planner. The green path represents a reckless goal pursuing path whilst being mine evasive whereas the red path exhibits higher stealth as well as mine evasiveness

2.7 Conclusions

This chapter has developed and evaluated a comprehensive framework for tactical global path planning, integrating the A^* and LPA^* algorithms with stealth-inducing modifications and a dual-mode obstacle interaction paradigm, tailored for UGV navigation in hostile environments. The tactical A^* algorithm, as detailed in Section 2.3.3, effectively balances stealth and efficiency in static environments, as evidenced by the numerical results in Section 2.3.4. The simulations revealed that increasing stealth levels, driven by the $\kappa(\alpha)$ function, lead to longer paths—up to a 13% increase in path length from 456 m to 516 m—reflecting detours to coast shelters, which reduce the UGV’s detectability at the cost of increased travel time and fuel consumption.

The LPA^* algorithm, introduced in Section 2.5, addresses the limitations of A^* in dynamic environments by reusing previous search results, ensuring rapid adaptation to incremental map updates—a critical feature for navigation in unpredictable terrains. The numerical results in Section 2.5.3 demonstrated LPA^* ’s ability to maintain comparable path quality to A^* in static conditions while offering significant advantages in dynamic scenarios. Path lengths increased by 40.1% with higher stealth levels, mirroring the trade-off observed with A^* , but the shelter proximity analysis confirmed closer coasting behavior (median distance decreasing from 60 m to 20 m), enhancing stealth. The path entropy analysis revealed an emergent benefit: tactical paths exhibit higher directional uncertainty (entropy rising from 1 to 1.5 bits), disrupting opponent motion prediction models, a finding that opens avenues for future research into deliberately maximizing entropy to further enhance unpredictability.

The dual-mode obstacle interaction framework, presented in Section 2.6, extends the tactical planning capability by distinguishing between coasting obstacles (shelters) and avoidance obstacles (mines). By setting the heuristic cost of avoidance obstacle nodes to infinity,

as described in Section 2.6.1, the planner ensures safe navigation without compromising computational efficiency and path optimality. The case study in Section 2.6.2 validated this approach, showing that tactical paths successfully avoid mines while coasting shelters, achieving both stealth and survivability. This dual-mode strategy is particularly valuable for military applications, where UGVs must operate in environments with mixed obstacles, balancing the need for concealment with the imperative to avoid hazards. Future work could explore adaptive weighting of the $\kappa(\alpha_c)$ functions to dynamically adjust the trade-off between stealth and safety, as well as investigate entropy-driven path planning to further confound adversarial tracking systems, thereby enhancing the tactical effectiveness of autonomous navigation in contested environments.

This chapter concludes the development of algorithms to produce tactical waypoints for UGV navigation. The next chapter will elaborate on the trajectory planner, which builds upon these waypoints by smoothing the path and producing time-scaled reference state vectors. These vectors, comprising position, velocity, orientation, and angular velocity, will serve as the reference state vector to be passed to the UGV's control system, enabling precise tracking in real-world conditions.

Chapter 3

Trajectory planning

Contributions

This chapter presents a cubic polynomial-based spline trajectory planner to generate collision-free reference *op-tempo* translational and angular velocities.

- A novel terrain-aware attitude planning algorithm and parallel transport
- A tactical weighing function to generate *op-tempo* translation reference velocities
- An analytical approach to generate reference angular velocities
- A fast, safe corridor generation algorithm and an optimization-based safe ellipsoid generation algorithm for collision avoidance guarantees

3.1 Introduction

The path planning module, as delineated in Chapter 2, produces a sequence of spatial waypoints endowed with tactical characteristics within the UGV's configuration space. These waypoints function as spatial coordinates or “guideposts” to which the UGV is compelled to adhere within acceptable tolerances. However, these waypoints, typically of a lower resolution, lack temporal parameterization. In contrast, the control system mandates a temporally parameterized reference vector to generate the requisite control signals. Ergo, a nexus is necessitated that connects the spatial waypoints into a temporally parameterized vector, which

the control system can efficaciously track. This time parametrized vector, which not only smoothenes the spatial waypoints produced by the path planner but also dictates the *tempus* of traversal, is referred to as the *reference trajectory*, and the algorithmic design responsible for its generation is referred to as *trajectory planning*.

Within the hierarchical structure of the autonomy stack, as delineated in Chapter 1, trajectory planning is positioned in the lower rungs, constituting the final stage of the guidance system and close proximity to the control system. As such, it not only signifies the *ultima ratio* for instilling “intelligence” within the system but presents another challenge: the requirement of higher computational speed. Recalling the broader aims of this research, that is to develop a UGV capable of terrain-aware and tactical navigation, and with the additional requirement of higher computational speed, it becomes imperative to strike a balance between tactical efficacy, terrain awareness, and computational efficiency. In this research, collision avoidance and part of the tactical aspect are indeed handled by the path planning module, however, the trajectory planner enhances these aspects in a finer resolution and the UGV’s vicinity with more advanced techniques. Consequently, three requirements of the trajectory planning module within the autonomy stack emerge- firstly, to ensure that the reference trajectory adapts to the topographical nuances of the terrain; secondly, to imbue the UGV with tactical capabilities; and thirdly, to sustain the computational efficiency requisite for real-time operations. It is noteworthy that, whereas “tactical” planning in Chapter 2 refers to the stealth capabilities of the UGV, *i.e.*, modulating the shape of the path to seek shelter around obstacles, the trajectory planner, on the other hand, also imbues the system with tactical capabilities via regulating speed as a function of its distance to the obstacles. These features, namely the obstacle-seeking characteristic alongside the *op-tempo* velocity-regulation, increase the trajectory’s entropy whilst enabling the UGV to seamlessly arrive at the goal set.

The subsequent sections elaborate on the design philosophy underlying trajectory planning, examine the limitations of existing approaches, and present the development of a cubic polynomial-based trajectory that meets the expected requirements. Finally, the chapter presents numerical simulations that demonstrate the applicability of the proposed trajectory planning system.

3.1.1 Problem Formulation of Trajectory Planning

The trajectory planning problem entails synthesizing a time-parameterized reference trajectory $\mathcal{R}(t)$ that maps a sequence of spatial waypoints $\mathcal{W}(r) \in \mathcal{C}$, generated by the path planner (Chapter 2), into a dynamically feasible state vector for the UGV's control system, ensuring terrain-aware, tactical navigation in $\mathbb{R}^3$. Let $\mathcal{C} \subset \mathbb{R}^3 \times \mathbb{H}$ denote the UGV's configuration in space, with initial configuration $r_0 \in \mathcal{C}$ and goal set $\mathcal{G} \subset \mathcal{C}$. The trajectory is defined as $\mathcal{R}(t) \triangleq [r(t), \dot{r}(t), \Theta(t), \Omega(t)] \in \mathbb{R}^n$, where $r(t) \in \mathbb{R}^3$ is the position, $\dot{r}(t) \in \mathbb{R}^3$ is the translational velocity, $\Theta(t) \in \mathbb{H}$ is the orientation represented using unit quaternions, and $\Omega(t) \in \mathbb{R}^3$ is the angular velocity, for $t \in [0, \infty)$. The problem is formally expressed as:

$$\mathcal{R}(t) = f_{\mathcal{R}}(\mathcal{W}(r)), \quad f_{\mathcal{R}} : \mathbb{R}^3 \rightarrow \mathbb{R}^n \times [0, \infty), \text{ s.t. } r(t) \in \mathcal{C}_{\text{free}}, \quad \Theta(t) \in \eta, \quad \dot{r}(t), \quad \Omega(t) \in \mathcal{K}, \quad (3.1)$$

where $\mathcal{C}_{\text{free}} \subset \mathcal{C}$ is the obstacle-free configuration space, $\eta \subset \mathbb{H}$ ensures terrain-aligned orientations, and $\mathcal{K}$ is the feasible set of angular velocities. The function $f_{\mathcal{R}}$ must produce a C^2 -continuous trajectory to ensure smoothness, incorporate *op-tempo* velocity regulation for tactical speed modulation, and integrate terrain topography via the attitude planner (Section 3.4), all while maintaining computational efficiency for real-time control [13].

3.1.2 Design philosophy of trajectory planning

Let $\mathbb{I} \triangleq \{O; X, Y, Z\}$ denote a right-handed gravity-aligned fixed reference inertial frame, and let $\mathbb{J} \triangleq \{O'; X', Y', Z'\}$ represent the right-handed local reference frame attached to the vehicle's center of gravity, as depicted in Figure 3.1. Let $\mathcal{C} \subset \mathbb{R}^3 \times \mathbb{H}$ represent the configuration space of the UGV, where $r_0 \in \mathcal{C}$ denotes the UGV's initial configuration and $\mathcal{G} \subset \mathcal{C}$ denotes the goal set. The sequence of way-points generated by the path planner, $\mathcal{W}(r) \in \mathcal{C}$, spatially connects the initial position r_0 to the goal set $\mathcal{G}$. The temporally parameterized reference *trajectory*, $\mathcal{R}$, is thus expressed as $\mathcal{R}(t)$.

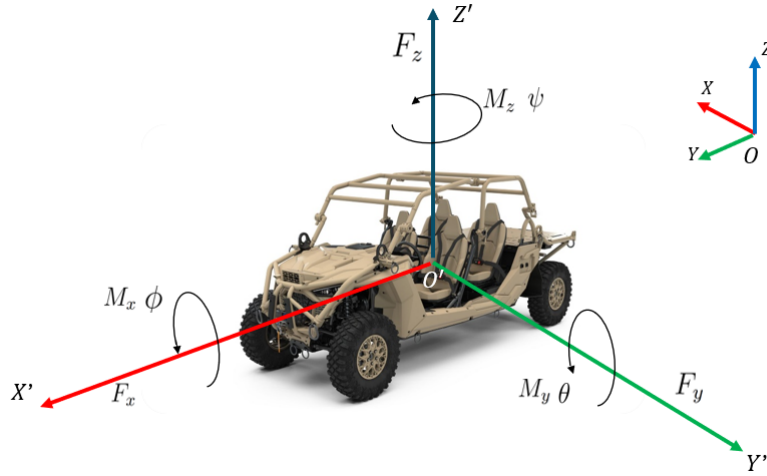


Figure 3.1: Frame of reference: Gravity-aligned fixed inertial reference frame and body fixed reference frame

Typically, the reference trajectory is characterized as a vector of states within the UGV's state-space, which includes not only the time-dependent spatial coordinates but also orientation and derivatives in both translation and rotation [13]. While it is conventional to represent the trajectory within the state-space, some researchers have developed trajectories within the control space [36], and others have explored a synthesis of state-space and control space [1, 2, 37]. In this research, the trajectory is designed within the state-space and defined

as $\mathcal{R}(t) \triangleq [r(t), \dot{r}(t), \Theta(t), \Omega(t)] \in \mathbb{R}^n$. Orientation in this study is deliberately represented using unit quaternions [38] or Euler parameters [39] to ensure singularity-free computations. The problem of trajectory planning can now be formally expressed as:

$$\mathcal{R}(t) = f_{\mathcal{R}}(\mathcal{W}(r)), \quad f_{\mathcal{R}} : \mathbb{R}^3 \rightarrow \mathbb{R}^n \times [0, \infty) \quad (3.2)$$

Equation 3.2 encapsulates the essence of trajectory planning, positing it as a function that maps a set of spatial waypoints into a multidimensional trajectory space over time. Solving this equation indeed produces the required reference trajectory. However, applying this solution in practical scenarios reveals several constraints and limitations that must be rigorously managed. For example, the position must belong to the obstacle-free set within $\mathcal{C}$. Furthermore, the orientations of the UGV on an uneven terrain within the planning window must account for the terrain topography, which is crucial in the present research. Such constraints are often defined by reachable or feasible sets, such as $r(t) \in \epsilon$ and $\Theta(t) \in \eta$, which abstractly represent real-world feasibility. The general philosophy behind this trajectory planning initiative is to craft trajectories that respect these constraints while simultaneously meeting the demands for fast computation and real-time responsiveness. It is noteworthy that the trajectory outputs are specifically designed to interface seamlessly with the control systems employed in the UGV. The UGV used in the present research is a Clearpath Husky, whose in-built controller requires time-parameterized translational and angular velocities.

The trajectory planning challenge addressed in this research is thus formally defined by the following questions:

1. **Collision-Free Navigation:** How to ensure that the reference pose trajectory $r_{ref}(t)$ is continuous and remains free from collisions throughout its course?

2. **Kinematic and Dynamic Feasibility:** How to maintain smoothness, *i.e.* C^2 continuity for $r_{ref}(t)$ as well as kinematic feasibility for the planned translational velocities $\dot{r}_{ref}(t)$ and angular velocities $\Omega_{ref}(t)$?
3. **Terrain-Aware Planning:** How can terrain topography data be effectively integrated to ensure that the reference trajectory is accurately reflective of real-world three-dimensional spaces, thereby enhancing terrain-aware trajectory planning?
4. **Tactical trajectory planning:** How to regulate the velocities to enable the UGV with *op-tempo* motion regulation capability?
5. **Algorithm Design and Efficiency:** How to design the algorithm for $f_{\mathcal{R}} : \mathbb{R}^3 \rightarrow \mathbb{R}^n \times [0, \infty)$ to be both swift and reliable, ensuring it meets the stringent demands for real-time applicability and computational efficiency?

The following section presents a survey of both problem formulations as well as solution strategies that is pertinent to the present trajectory planning problem.

3.2 Literature review

The trajectory planning problem can be broadly analyzed from two perspectives: **problem formulation** and **solution methodology**. The formulation involves defining the trajectory vector and establishing constraints that govern reachable and non-reachable states for each system state. On the other hand, solution methodologies encompass the mathematical and computational techniques used to solve the posed problem such as optimization, parametric curves, learning approaches *etc.* In essence, trajectory planning must balance the criteria that define system behavior with the available computational resources. A stricter

adherence to specific constraints, such as collision avoidance, kinematic feasibility, or actuator limitations, typically necessitates more sophisticated solution approaches, such as constrained optimization techniques. These methods can guarantee that the trajectory meets the imposed criteria, but they often introduce additional computational costs, which can compromise the efficiency required for real-time applications. Conversely, when computational speed is prioritized over strict adherence to optimality, alternative approaches—such as unconstrained methods, heuristic techniques, or sampling-based algorithms—can be employed. These methods may offer faster solutions but potentially sacrifice precision or the ability to meet certain constraints. Given the requirements of this research as outlined in Section 3.1.2, the objective of this survey is to seek a methodology that has aids in accomplishing the larger goal of terrain aware trajectory planning. The following subsections will explore existing solution approaches, including spline-based trajectories, Bézier curves, potential field methods, and optimization techniques, as well as a discussion on the formulation of constraints such as collision avoidance, kinematic feasibility, and motor torque limits.

The trajectory planning problem for off-road UGVs has been addressed in several ways. Broadly, they are approached via deterministic techniques like optimization methods, parameteric curve fitting, state space and control space sampling, and dynamic programming ([13, 36, 37, 40]). Recently, data-driven techniques have found prominence due to their ability to handle uncertainties that are not explicitly modeled as in [41, 42, 43]. Motion primitives, on the other hand, represent predefined trajectories under specific operating conditions. Although these approaches can serve as viable solutions [44], the fast tactical planning and replanning demands of the present research, necessitate a more dynamic approach.

Hierarchical motion planning, wherein a global, coarse path planning module is com-

plemented by a local, finer-scale trajectory planning framework, is widely adopted across numerous studies. This prominence stems from the critical importance of ensuring geometric feasibility over longer distances, contrasted with the immediate safety and dynamic requirements in the vicinity of the UGV. One of the main factors accounted in trajectory planning is collision avoidance guarantees.

3.2.1 Review on collision avoidance strategies in local planners:

The authors in [45] proposed a discrete waypoint generation approach (akin to the methodology under consideration in this dissertation), followed by an optimization-based trajectory planning algorithm. Specifically, the trajectory planning problem is framed as a minimum-time optimization challenge, factoring in collision avoidance by constructing a safe corridor along the reference waypoints and incorporating kinematic constraints, though the problem is modeled in $\mathbb{R}^2$. Similarly, the authors in [46] employs a comparable strategy, beginning with a sequence of reference waypoints and generating a safe corridor for collision avoidance. However, unlike the approach in [45], they introduce multiple layers of “path smoothing” prior to time scaling. In this context, the reference waypoints, treated as the centerline of the corridor, are smoothed via an optimization process to minimize distance from the corridor walls, followed by additional Bezier curve-based smoothing. These curves are further refined using polynomials parameterized by the spline variable, solved through a quadratic programming approach that adheres to kinematic constraints imposed on the first and second derivatives of the polynomials at key points. Ultimately, the velocity profile is derived as a function of the spline parameter. It is worth noting here that the spline trajectory proposed by the authors is not modeled as an explicit function of time. Furthermore, since the algorithm is confined to $\mathbb{R}^2$, constraints on the second derivatives of the spline do not directly translate to centripetal acceleration limits due to the fact that, the curvature in

$\mathbb{R}^3$ is not necessarily aligned with the UGV's pitch axis. Along similar lines of hierarchical motion planning with safe-space generation for trajectory planning, the authors in [1, 2] introduced an optimization-based trajectory planning algorithm. In contrast to previous studies, their local trajectory planner is formulated as a convex optimization problem, where the collision avoidance constraint is modeled by maximizing a safe "ellipsoid" around the UAV. This is followed by generating half-planes at points sampled on the ellipsoid to create the convex constraints for the original optimization formulation. Likewise, the authors in [47] employed a hierarchical motion planning framework with an A*-based path planning module in $\mathbb{R}^2$. Their approach tackles the trajectory problem in three levels: initial path smoothing through optimization subject to collision avoidance via a maximum "bounding box," followed by cubic polynomial fitting to model path curvature, and finally, time sampling for speed profiling. As per the approach in [45], the spline parameter is not explicitly linked to the time variable and the constraints on second derivatives of the curve do not directly translate to acceleration constraints in $\mathbb{R}^3$. A common thread in all these works is the design philosophy of collision-free spaces - they are either modeled as corridors or convex spaces like ellipsoids, octahedrons, or bounding boxes. The authors in [48] proposed a Bezier curve for based path smoothening algorithm for $\mathbb{R}^2$, but rather than accounting for collision avoidance via explicitly modeling the free space, they propose to perform "collision checks" by evaluating the polylines connecting each pair of knot points with the obstacles. Whereas such an approach would indeed prove to be guaranteed when the obstacles are convex, the extension of this methodology to nonconvex obstacle sets, especially in real-time, is perhaps wanting of further investigation. A common theme among these works is the design of collision-free spaces, modeled either as corridors or convex spaces such as ellipsoids, octahedrons, or bounding boxes. The authors in [48], on the other hand, proposed a collision avoidance strategy by performing "collision checks" as opposed to explicitly modeling collision-free spaces. The trajectory modeled using Bezier curves are evaluated by fitting

polylines connecting knot points with obstacle sets. While this method may be reliable for convex obstacles, extending it to nonconvex obstacle sets, especially in real-time, requires further exploration.

The main takeaway is that collision avoidance, formulated as safe corridors or convex spaces, holds promise for terrain-aware tactical trajectory planning. However, it is essential to recognize that most solutions have been applied in $\mathbb{R}^2$, while this research aims to extend them to $\mathbb{R}^3$. The primary challenge stems from the inherently nonconvex "floor" boundary created by the terrain the vehicle traverses. Another critical consideration is that, although speed profiling with polynomial curves offers computational efficiency, it is typically modeled without explicit time parameterization, relying solely on the spline parameter.

3.2.2 Review on terrain aware trajectory planning strategies:

Whereas collision avoidance is undeniably crucial in trajectory planning, it is equally essential to ensure that the trajectory is feasible for the controller to track. Various approaches have been adopted to account for this requirement. For instance, the authors in [36] addressed local trajectory planning within the control space through control sampling. Specifically, a fixed number of samples from the control space are propagated forward in time using a vehicle model, and the cost of each trajectory is evaluated based on user-defined criteria. The trajectory with the lowest cost is selected and applied to the controller. The authors in [49] propose a technique using a library of trajectories for various control actions and states. The sampled trajectories are evaluated using an optimization approach, however, in the state space. The authors in [50] approached the problem within the control space, similar to the approach in [36], but employed an optimization technique like [49], explicitly

incorporating vehicle dynamics constraints into the model. Following a similar framework to [20, 36] proposed a “minimum cost” trajectory planner by solving for multiple quintic polynomial-based trajectories in $\mathbb{R}^2$, evaluating their cost based on the derivatives of the polynomials, and retaining the trajectory with the minimal cost. This work was largely inspired by [51], where a kinematically feasible trajectory planning algorithm was introduced in the Frénet frame in $\mathbb{R}^2$. The philosophy of trajectory planning in the control space was also employed by [52] wherein a higher-level path planner provides long-range reference waypoints, followed by local trajectory planning based on an optimization framework that minimizes a user-defined cost function to derive the optimal control policy. This control policy is subject to state, kinodynamic, and pose constraints. It is noteworthy that the constraints are enforced on variables that depend on the vehicle’s orientation. Unlike state estimation, where orientations can be derived from onboard gyroscopes, trajectory planning requires predicting a vector of future states and controls. Thus, vehicle orientation must be forecasted ahead of time, which implies a requirement of an understanding of the terrain topography, which indeed is the case here. The vehicle orientation is determined by constructing surface normals from the terrain data, stored as a 2.5D map. These surface normals are then used to predict the vehicle’s orientation, specifically its roll and pitch angles.

Up to this point, the problem has been addressed primarily within the control space, focusing on obtaining the “best” control inputs either through explicit optimization methods or by calculating minimum-cost trajectories via sampling, while adhering to specific constraints. However, the trajectory planning problem can also be formulated in the state space, where the system’s states are optimized, presenting different advantages and challenges in trajectory planning. For example, the authors in [53] proposed a polynomial-based trajectory planner, solved as a two-point boundary value problem. The curve was modeled as a parametric curve by explicitly formulating the curvature as a cubic polynomial in $\mathbb{R}^2$.

Similarly, the authors in [54] also introduced a polynomial-based trajectory, but in stark contrast to the other works presented up to this point, they explicitly modeled the time parameter and solved it via an optimization problem. The problem accounts for velocity and acceleration constraints as saturation limits; however, the velocity and acceleration are modeled using a first-order Euler forward approximation method.

On the other hand, the approach shown in [55] solved the trajectory planning in $\mathbb{R}^2$ as an optimal control problem by minimizing the control effort, smoothening the control effort to reduce jerky behaviors, and a spatially dependent traversability cost. The authors in [37] elaborated on the optimal control framework and modeled the trajectory planning problem in the combined state-control space using a model predictive control (MPC) approach. Here, they explicitly account for dynamic constraints, with an emphasis on a neural network-based tire force model that serves as a surrogate for terramechanics-based force models. Similarly, the authors in [56] proposed a motion planning strategy in $\mathbb{R}^3$ using an optimal control method. The cost function, in this case, was designed to minimize total power consumption. The constraints, however, were modeled for local 2D planar kinematics but were designed to account for the UGV's pose in the inertial frame by modeling the rotation matrix using the UGV's roll, pitch, and yaw angles. As seen in [52], the orientation was obtained by relying on the DEM-based terrain map. Specifically, the forward Euler difference and arctangents based on terrain height in longitudinal and lateral directions were employed to determine the roll and pitch angles.

Unlike a “one-shot” optimization approach as seen earlier, the authors in [57] proposed a trajectory planner using two optimization steps. The first step was solved as a dynamic programming problem to obtain a coarse trajectory. More specifically, spatio-temporal trajectory points were sampled in the Frenet space at consecutive time intervals, and the cost for each point was evaluated based on collision safety, stability, smoothness, and traversability.

The traversability cost, in particular, accounted for static stability in terms of orientation limits, terrain slope, and height. As seen earlier, vehicle orientation was again obtained by relying on the terrain map. The approach followed here involved projecting the UGV's footprint or support polygon onto the terrain map in the form of a geodesic plane, followed by constructing surface normals to form a matrix in $SO(3)$. A similar approach was also observed in [40]. The second layer of trajectory planning was a constrained optimization problem, designed to minimize tracking error with respect to the initial coarse trajectory and control effort, subject to mobility constraints modeled as saturation limits on the vehicle states. The authors in [58] used a similar concept of a support polygon. The key issue addressed was that velocity regulation is required as the UGV traverses various orientations of the terrain. As a result, orientation calculation becomes necessary. The terrain map, represented as a DEM, was modeled as a web of triangles. The pitch and roll angles were then evaluated across the centroids and vertices of each triangular element. These obtained angles were subsequently used to formulate a nonlinear velocity model that would reduce the vehicle's speed when the terrain is "uneven," while keeping it at a nominal value when the terrain is relatively flat.

At this point, it is clear that the orientation of the UGV on uneven terrain plays a crucial role in stability, mobility, and traversal cost. However, the approaches observed so far have been heavily dependent on pre-existing terrain maps. The challenge posed by the present terrain-aware tactical UGV research is that the map is constructed *in situ*. This introduces the likelihood of inevitable uncertainties in both the environmental map and the terrain map. Furthermore, unless the terrain map is of high resolution, approaches such as deriving surface normals may result in severe deviations from the actual terrain topography. This issue was echoed by [59], who highlighted that although the rotation matrix representing the UGV's orientation is formulated in $SO(3)$, the matrix is often subject to uncertainty. The direct

consequence of this is that algorithms dependent on these matrices or their equivalent angles could violate constraint formulations or disrupt the planning process entirely. In response to this, the authors in [59] attempt to “fix” the $SO(3)$ matrix and proceed to solve the problem as a Nonlinear Optimal Control Problem (NOCP), similar to the support polygon and zero moment point (ZMP) methods.

Some works have circumvented the problem of explicit dependence on the UGV orientation by other techniques. For example, the authors in [60] proposed an optimal control trajectory planner. Albeit solving the problem in $\mathbb{R}^2$, the optimal control problem seeks a trajectory that minimizes a “risk” factor that is modeled using the elevation information from the terrain map. The authors in [29] created the trajectory planner in 3 levels. In the first level, an A* algorithm produces guiding waypoints that help in the sampling of nodes for the subsequent RRT layer. The sampled nodes are in $\mathbb{R}^3$; however, the node to be added to the RRT tree is determined based on the cost on a local plane in $\mathbb{R}^2$ subject to dynamic constraints. The new node’s cost is evaluated by a neural network trained on optimal trajectories obtained via offline optimization methods. Furthermore, the final trajectory is further refined via perturbation to obtain more optimal trajectories if found. A similar approach is seen in [61](37) but in $\mathbb{R}^2$. Here, the RRT is guided via the forces of a potential field, unlike the A* as in [29]. The node appended to the tree is evaluated for heading costs and the final trajectory fine tuning is replaced via polynomial fitting that is solved as an optimization problem by explicitly modeling speed and acceleration constraints.

From the literature survey above and in consideration of the specific requirements of this research, several key conclusions can be drawn.

- First, to achieve fast trajectory planning, polynomial curves, cubic polynomials in particular, offer a significant computational advantage over other methods. However, neither the polynomial curves nor other parametric curves explored explicitly account

for time parameterization.

- Second, relying on terrain maps to deduce vehicle orientation within a fixed inertial frame may introduce uncertainties when applied in real-time systems.
- Third, none of the surveyed trajectory planning algorithms address *tactical trajectory planning*, except for [24], as discussed in Chapter 2, though it cannot be applied here due to its reliance on a deep reinforcement learning approach.
- Fourth, while collision-free planning using corridors proves more feasible, these corridors are not explicitly designed to be perpendicular to the local terrain surface.

Given these gaps in the state of the art, it is essential to develop the algorithms outlined in the subsequent sections to meet the broader objectives of this research.

3.3 Proposed cubic polynomial trajectory planner

Consider the piecewise continuous cubic spline trajectory represented by $\mathcal{R} : \mathbb{R} \rightarrow \mathbb{R}^3$, where:

$$\mathcal{R}(\xi) \triangleq [x(\xi), y(\xi), z(\xi)]^T, \quad (3.3)$$

and $\xi \in [0, L]$ denotes a spatial parameter representing the arc length along the spline. For each $\xi \in [0, L]$, the components $x(\xi)$, $y(\xi)$, and $z(\xi)$ of the interpolating spline segment are defined as:

$$x(\xi) = \sum_{i=0}^3 a_i \xi^i, \quad y(\xi) = \sum_{i=0}^3 b_i \xi^i, \quad z(\xi) = \sum_{i=0}^3 c_i \xi^i, \quad (3.4)$$

where $a_i, b_i, c_i \in \mathbb{R}$ are the spline coefficients for each segment. Determining these coefficients involves solving a tridiagonal linear system based on Hermite interpolation [62], ensuring C^1 continuity (continuous position and first derivatives) across segment boundaries. Given a

sequence of waypoints $\mathcal{W}(r) \in \mathcal{C}$ from the path planner, Hermite interpolation enforces position and tangent constraints at each waypoint, computed efficiently using established numerical libraries, such as MATLAB's `spline()` function or the GSL library in C++. The resulting spline provides a smooth, terrain-aware path that respects the UGV's configuration space constraints while facilitating tactical modulation of curvature and orientation.

3.4 Attitude Planner

The cubic spline $r(\xi(t))$, $t \in [0, \infty)$, encapsulates the spatial and temporal evolution of an unmanned ground vehicle (UGV) navigating uneven terrain. Determining the UGV's orientation along this polynomial trajectory is critical for deriving reference translational and angular velocities, which can enforce orientation constraints to prevent roll-over or nose-diving. Let ${}^{\mathbb{I}}R(\xi) \in SO(3)$, $\xi \in [0, \infty)$, denote the rotation matrix mapping the UGV's body-fixed frame $\mathbb{J}$ to the global frame $\mathbb{I}$. The Euler angles or unit quaternions derived from ${}^{\mathbb{I}}R(\xi)$ at any point ξ define the UGV's orientation, enabling precise control. The three columns of ${}^{\mathbb{I}}R(\xi)$ are constructed as the unit tangent, normal, and binormal vectors to $r(\xi)$, $\xi \in [0, \infty)$.

Since the cubic polynomial (3.4) is a twice-differentiable curve in $\mathbb{R}^3$, parameterized by ξ , the unit *tangent vector* is defined as

$$T(\xi) \triangleq \left\| \frac{dr(\xi)}{d\xi} \right\|^{-1} \frac{dr(\xi)}{d\xi}, \quad \xi \in [0, \infty). \quad (3.5)$$

This vector represents the UGV's instantaneous heading in $\mathbb{R}^3$. Given the UGV's non-zero constant translational velocity along the spline, (3.5) is well-defined for all $t \in [0, \infty)$. The

unit *normal vector* is

$$N(\xi) \triangleq \frac{1}{\gamma(\xi)} \frac{dT(\xi)}{d\xi}, \quad \xi \in [0, \infty), \quad (3.6)$$

where the *curvature* $\gamma(\xi) \triangleq \left\| \frac{dT(\xi)}{d\xi} \right\|$ quantifies the spline's bending. The orthogonality condition $\langle T(\xi), N(\xi) \rangle = 0$ holds for all ξ . The *binormal vector* is

$$B(\xi) \triangleq T(\xi)^\times N(\xi), \quad \xi \in [0, \infty), \quad (3.7)$$

where $(\cdot)^\times$ denotes the cross product operator. The set $\{r(\xi), T(\xi), N(\xi), B(\xi)\}$ forms the Frénet-Serret frame [63]. However, at inflection points where $\gamma(\xi) \rightarrow 0$, the Frénet-Serret frame undergoes abrupt reorientation about $T(\xi)$, causing the osculating plane to twist. This makes $\mathbb{I}R(\xi)^\mathbb{J} = [T(\xi), N(\xi), B(\xi)]$ impractical for stable UGV orientation. To address this, we employ rotation-minimizing frames (RMFs), which minimize torsion and ensure smooth frame propagation. In this research, we explore two RMF methods: Wang's double reflection method and a proposed double projection method, detailed in the following subsections.

3.4.1 The Double Reflection Method

To compute a rotation-minimizing frame (RMF) for the UGV's orientation along the cubic spline $r(\xi)$, $\xi \in [0, \infty)$, we employ the double reflection method proposed by Wang *et al.* [64]. This method propagates an initial orthonormal frame to minimize torsion, ensuring a smooth orientation that avoids the twisting inherent in the Frénet-Serret frame, as described in Section 3.4. The rotation matrix $\mathbb{I}R(\xi)^\mathbb{J} \in SO(3)$ comprises columns $[T(\xi), N(\xi), B(\xi)]$, where $T(\xi)$ is the unit tangent vector from (??), and $N(\xi)$ and $B(\xi)$ are the normal and binormal vectors. Starting from an initial frame $\mathbb{I}R(\xi_0)^\mathbb{J} = [T(\xi_0), N(\xi_0), B(\xi_0)]$ at ξ_0 , the method uses two reflections to compute subsequent frames, achieving a fourth-order global approximation error.

The procedure, outlined in Algorithm 5, operates on a sequence of points $r(\xi_i)$, $i = 0, \dots, n$. For each ξ_i , the tangent $T(\xi_{i+1})$ is computed via (3.5). The first reflection vector is $v_1 = r(\xi_{i+1}) - r(\xi_i)$, representing the displacement along the spline. The dot product $c_1 = \langle v_1, v_1 \rangle$ computes the squared length of v_1 , geometrically defining the scale of the displacement. The previous binormal $B(\xi_i)$ and tangent $T(\xi_i)$ are reflected across the plane perpendicular to v_1 :

$$B^L(\xi_i) = B(\xi_i) - \frac{2\langle v_1, B(\xi_i) \rangle}{c_1} v_1$$

where $\langle v_1, B(\xi_i) \rangle$ and $\langle v_1, T(\xi_i) \rangle$ measure the alignment of $B(\xi_i)$ and $T(\xi_i)$ with the displacement, and the reflection flips these vectors across the plane, producing a temporary left-handed frame. Visually, this aligns the frame closer to the curve's path. The second reflection vector $v_2 = T(\xi_{i+1}) - T^L(\xi_i)$ captures the difference between the new tangent and the reflected tangent, with $c_2 = \langle v_2, v_2 \rangle$ as its squared length. The binormal is then reflected across the plane perpendicular to v_2 :

$$B(\xi_{i+1}) = B^L(\xi_i) - \frac{2\langle v_2, B^L(\xi_i) \rangle}{c_2} v_2, \quad (3.8)$$

restoring a right-handed frame. The normal is computed as $N(\xi_{i+1}) = B(\xi_{i+1}) \times T(\xi_{i+1})$, where the cross product ensures orthogonality, geometrically defining the plane perpendicular to both $B(\xi_{i+1})$ and $T(\xi_{i+1})$. The updated frame is ${}^{\mathbb{I}}R(\xi_{i+1})^{\mathbb{J}} = [T(\xi_{i+1}), N(\xi_{i+1}), B(\xi_{i+1})]$. These reflections minimize rotation about $T(\xi)$, akin to smoothly adjusting the UGV's roll and pitch to follow the spline's geometry.

Algorithm 5 Double Reflection RMF Generation [64]

Input: Spline points $\{r(\xi_i)\}_{i=0}^n$, initial frame $\mathbb{J}(\xi_0) = \{r(\xi_0); T(\xi_0), N(\xi_0), B(\xi_0)\}$
Output: Frames $\mathbb{J}(\xi_i) = \{r(\xi_i); T(\xi_i), N(\xi_i), B(\xi_i)\}$, $i = 0, \dots, n$

```

1: function RMF( $\mathbb{J}(\xi_0), \{r(\xi_i)\}$ )
2:   for  $i = 0$  to  $n - 1$  do
3:      $v_1 \leftarrow r(\xi_{i+1}) - r(\xi_i)$  ▷ Displacement vector
4:      $c_1 \leftarrow \langle v_1, v_1 \rangle$ 
5:      $B^L(\xi_i) \leftarrow B(\xi_i) - \frac{2\langle v_1, B(\xi_i) \rangle}{c_1} v_1$  ▷ First reflection
6:      $T^L(\xi_i) \leftarrow T(\xi_i) - \frac{2\langle v_1, T(\xi_i) \rangle}{c_1} v_1$ 
7:      $v_2 \leftarrow T(\xi_{i+1}) - T^L(\xi_i)$  ▷ Tangent difference
8:      $c_2 \leftarrow \langle v_2, v_2 \rangle$ 
9:      $B(\xi_{i+1}) \leftarrow B^L(\xi_i) - \frac{2\langle v_2, B^L(\xi_i) \rangle}{c_2} v_2$  ▷ Second reflection
10:     $N(\xi_{i+1}) \leftarrow B(\xi_{i+1}) \times T(\xi_{i+1})$  ▷ Normal via cross product
11:     $\mathbb{J}(\xi_{i+1}) \leftarrow \{r(\xi_{i+1}); T(\xi_{i+1}), N(\xi_{i+1}), B(\xi_{i+1})\}$ 
12:   end for
13: end function

```

3.4.2 Proposed Double Projection Method

To address the twisting limitations of the Frénet-Serret frame for UGV orientation along the cubic spline $r(\xi)$, $\xi \in [0, \infty)$, we propose another method called double projection method to compute a rotation-minimizing frame (RMF). This method also propagates an initial orthonormal frame to minimize torsion, using sequential projections to compute subsequent frames, offering computational simplicity over reflection-based approaches.

The procedure, detailed in Algorithm 6, iterates over sampled points ξ_i , $i = 1, \dots, n$. For each ξ_i , the tangent $T(\xi_i)$ is computed using (3.5), representing the UGV's heading. The previous normal $N(\xi_{i-1})$ is projected onto the osculating plane formed by $N(\xi_{i-1})$ and $T(\xi_i)$ and subsequently, the component in the shifted $N(\xi_i)$ along the $T(\xi_i)$ is removed to preserve orthogonality between $T(\xi_i)$ and $N(\xi_i)$.

$$N_{\text{perp}} = N(\xi_{i-1}) - \langle N(\xi_{i-1}), T(\xi_i) \rangle T(\xi_i), \quad N(\xi_i) = \frac{N_{\text{perp}}}{\|N_{\text{perp}}\|}, \quad (3.9)$$

where $\langle N(\xi_{i-1}), T(\xi_i) \rangle$ measures the alignment of $N(\xi_{i-1})$ with the new heading. Subtracting this component ensures N_{perp} is orthogonal to $T(\xi_i)$, geometrically aligning the normal to the curve's path, and normalization maintains unit length. The previous binormal $B(\xi_{i-1})$ is then projected onto the plane perpendicular to both $T(\xi_i)$ and $N(\xi_i)$:

$$B_{\text{perp}} = B(\xi_{i-1}) - \langle B(\xi_{i-1}), T(\xi_i) \rangle T(\xi_i) - \langle B(\xi_{i-1}), N(\xi_i) \rangle N(\xi_i), \quad B(\xi_i) = \frac{B_{\text{perp}}}{\|B_{\text{perp}}\|}. \quad (3.10)$$

The dot products $\langle B(\xi_{i-1}), T(\xi_i) \rangle$ and $\langle B(\xi_{i-1}), N(\xi_i) \rangle$ quantify the binormal's alignment with $T(\xi_i)$ and $N(\xi_i)$, respectively. Subtracting these components ensures B_{perp} is orthogonal to both, visually stabilizing the UGV's roll axis relative to the new frame, and normalization yields $B(\xi_i)$. The updated frame is $\mathbb{I}R(\xi_i) = [T(\xi_i), N(\xi_i), B(\xi_i)]$, minimizing rotation about $T(\xi_i)$ for smooth orientation.

Algorithm 6 Double Projection RMF Generation

Input: Spline points $\{\xi_i\}_{i=1}^n$, spline functions $r_x(\xi), r_y(\xi), r_z(\xi)$, initial frame $\mathbb{I}(\xi_1) = \{r(\xi_1); T_0, N_0, B_0\}$

Output: Frames $\mathbb{I}(\xi_i) = \{r(\xi_i); T(\xi_i), N(\xi_i), B(\xi_i)\}$, $i = 1, \dots, n$

```

1: function RMF( $\mathbb{I}(\xi_1), \{\xi_i\}, r_x, r_y, r_z$ )
2:    $\mathbb{I}R(\xi_1) \leftarrow [T_0, N_0, B_0]$  ▷ Initialize first frame
3:   for  $i = 2$  to  $n$  do
4:      $T(\xi_i) \leftarrow \left\| \frac{dr(\xi_i)}{d\xi} \right\|^{-1} \frac{dr(\xi_i)}{d\xi}$  ▷ Compute tangent
5:      $N_{\text{perp}} \leftarrow N(\xi_{i-1}) - \langle N(\xi_{i-1}), T(\xi_i) \rangle T(\xi_i)$  ▷ Project normal
6:      $N(\xi_i) \leftarrow \frac{N_{\text{perp}}}{\|N_{\text{perp}}\|}$ 
7:      $B_{\text{perp}} \leftarrow B(\xi_{i-1}) - \langle B(\xi_{i-1}), T(\xi_i) \rangle T(\xi_i) - \langle B(\xi_{i-1}), N(\xi_i) \rangle N(\xi_i)$  ▷ Project
       binormal
8:      $B(\xi_i) \leftarrow \frac{B_{\text{perp}}}{\|B_{\text{perp}}\|}$ 
9:      $\mathbb{I}R(\xi_i) \leftarrow [T(\xi_i), N(\xi_i), B(\xi_i)]$  ▷ Update frame
10:  end for
11:  return  $\{\mathbb{I}R(\xi_i)\}_{i=1}^n$ 
12: end function

```

3.5 Spline Parameterization and Time Scaling

Time scaling is essential for *op-tempo* planning, enabling user-defined velocity profiles that dynamically adjust the UGV's speed to meet tactical mission requirements. Unlike minimum snap or minimum jerk planners, which prioritize smoothness with fixed velocity profiles and limited adaptability for real-time speed variations [65], this approach supports variable, user-specified velocities critical for *op-tempo*-driven navigation. For example, Singh *et al.* The authors in [66] employ non-linear time scaling to modify velocity and acceleration, ensuring no-slip and ground contact constraints on uneven terrain, a flexibility vital for maintaining stability in dynamic environments. Such planners lack the dynamic response needed for tactical scenarios, where the UGV must slow near obstacles or accelerate in safe zones. The explicit need for *op-tempo* time scaling stems from mapping the Hermite-interpolated spline $\mathcal{R}(\xi)$, parameterized by arc length $\xi \in [0, L]$, to a time-dependent trajectory $\mathcal{R}(t)$ that enforces kinematic consistency.

The arc length of the piecewise continuous cubic spline is given by:

$$L(\xi) = \int_0^\xi \|r'(\zeta)\| \, d\zeta, \quad \xi \in [0, L], \quad (3.11)$$

where

$$\|r'(\xi)\| \triangleq \sqrt{x'(\xi)^2 + y'(\xi)^2 + z'(\xi)^2}. \quad (3.12)$$

Imposing a user-defined traversal velocity $V > 0$, the rate of change of arc length with respect to time must satisfy:

$$V = \frac{dL(\xi(t))}{dt} = \frac{\partial L(\xi)}{\partial \xi} \frac{d\xi(t)}{dt}, \quad t \in [0, \infty). \quad (3.13)$$

Applying the Leibniz rule for differentiation under the integral:

$$\frac{\partial L(\xi)}{\partial \xi} = \|r'(\xi)\|, \quad (3.14)$$

substituting into Equation (3.13) yields:

$$V = \|r'(\xi(t))\| \frac{d\xi(t)}{dt}. \quad (3.15)$$

To compute the inverse relationship, solve for the time derivative:

$$\frac{dt}{d\xi} = \frac{1}{\frac{d\xi}{dt}} = \frac{\|r'(\xi)\|}{V}, \quad (3.16)$$

and integrate with respect to ξ :

$$t(\xi) = \int_0^\xi \frac{\|r'(\zeta)\|}{V} d\zeta, \quad \xi \in [0, L], \quad (3.17)$$

where $V = \text{sat}(\eta d_2(\bar{\xi}, \mathcal{O}), V_{\min}, V_{\max})$ adjusts speed based on obstacle distance d_2 at segment start $\bar{\xi}$, with $\eta > 0$, $V_{\max} \geq V_{\min} > 0$, and $\text{sat}(V, V_{\min}, V_{\max}) \triangleq \min\{\max\{V, V_{\min}\}, V_{\max}\}$.

3.5.1 *Op-Tempo* Tactical Velocity Planning

Tactical velocity planning, within the *op-tempo* framework, addresses the challenge of modulating the UGV's translational velocity to ensure safe, conservative navigation across uneven terrain while adhering to mission-driven tactical objectives. For safety-oriented driving, the strategy prioritizes deceleration in regions of high trajectory curvature or extreme elevation changes, mitigating risks of instability or loss of traction, and regulates speed based on proximity to obstacles, slowing near hazards and accelerating in safer zones. This ap-

proach contrasts with conventional velocity planners that maintain constant or pre-defined speeds, lacking the adaptability required for dynamic, obstacle-rich environments. The UGV platform used in this research, a Clearpath Husky with specified maximum velocity and acceleration constraints, ideally operates at a nominal velocity $v_{\text{nominal}} > 0$, adjusted dynamically based on orientation and environmental factors to balance tactical efficacy and safety. Orientation data, expressed as Euler angles $\phi, \theta, \psi \in \mathbb{R}$, are derived from the sequence of rotation matrices $R(t_k) \in SO(3)$ provided by the attitude planner (Section 3.4) over a future time horizon along the spline trajectory $\mathcal{R}(\xi)$. Obstacle distance $d_{\mathcal{O}}(\xi) > 0$, measured as the Euclidean distance from the spline point $r(\xi)$ and occupancy grid $\mathfrak{M}^{\mathcal{O}}$, informs proximity-based speed adjustments. Ergo, the tactical velocity problem is formulated as designing a user-defined velocity profile $V(\xi)$ that modulates v_{nominal} using a weighing function $\chi(\xi)$, ensuring kinematic consistency within the time scaling framework (Section 3.5) while respecting the UGV's dynamic limits. The formulation leverages the attitude planner's foresight to anticipate terrain-induced orientation changes and obstacle encounters, enabling proactive velocity modulation. Unlike minimum snap or minimum jerk planners, which enforce fixed velocity profiles optimized for smoothness, this approach dynamically adapts $V(\xi)$ to environmental and tactical cues, critical for *op-tempo* navigation. The UGV's maximum velocity V_{max} and acceleration constraints ensure that $V(\xi) \leq V_{\text{max}}$, with the nominal velocity v_{nominal} serving as a baseline, typically below V_{max} to allow headroom for tactical adjustments. The tactical velocity is thus defined as:

$$V(\xi) = \chi(\xi) \cdot v_{\text{nominal}}, \quad (3.18)$$

$$\chi(\xi) = 1 - \beta_1 \left(1 - \exp \left(-\beta_2 \phi(\xi)^2 - \beta_3 \theta(\xi)^2 - \beta_4 \psi(\xi)^2 - \frac{\beta_5}{d_{\mathcal{O}}(\xi)^2} \right) \right), \quad (3.19)$$

where $v_{\text{nominal}} > 0$ is the baseline velocity from the time scaling ODE (Section 3.5), $\phi(\xi), \theta(\xi), \psi(\xi)$ are Euler angles extracted from $R(t_k)$ at spline parameter ξ , $d_{\mathcal{O}}(\xi)$ is the obstacle distance, and $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5 > 0$ are user-defined parameters tuning the sensitivity to orientation and proximity.

The exponential form in Equation (3.19) ensures a smooth, non-linear response to curvature and obstacle distance, avoiding abrupt velocity changes that could destabilize the UGV. For large ϕ, θ, ψ (indicating high curvature or elevation) or small $d_{\mathcal{O}}$ (near obstacles), the exponent $-\beta_2\phi^2 - \beta_3\theta^2 - \beta_4\psi^2 - \frac{\beta_5}{d_{\mathcal{O}}^2}$ becomes more negative, reducing χ and thus $V(\xi)$. Conversely, small angles or large distances yield $\chi \approx 1$, allowing $V(\xi) \approx v_{\text{nominal}}$. The parameter β_1 scales the weighing effect, while $\beta_2, \beta_3, \beta_4$ adjust sensitivity to roll, pitch, and yaw, respectively, and β_5 governs obstacle proximity influence. This formulation, integrated with the time-scaling, produces a trajectory $\mathcal{R}(t)$ that respects the UGV's kinematic and dynamic constraints while enabling *op-tempo*-driven, safety-oriented navigation. The design ensures computational efficiency by evaluating $\chi(\xi)$ at discrete spline points, leveraging the attitude planner's pre-computed orientations, and supports real-time control via the Clearpath Husky's interface for time-parameterized velocities.

3.6 Angular Velocity Planner

The reference state vector of the UGV comprises the angular velocity vector $\omega = [\omega_x, \omega_y, \omega_z]^T \in \mathbb{R}^3$, critical for maintaining orientation stability across uneven terrain. Unlike planar UGV planners that equate angular velocity with yaw rate ω_z , 3D angular velocity planning is essential to account for terrain-induced roll and pitch effects, particularly in the off-road tactical navigation. Whereas nonlinear model predictive control (NMPC) [67] computes the full ω vector for 3D pose tracking, explicit methods for deriving 3D angular velocities re-

main limited in UGV literature. Given the tactical *op-tempo* velocity planning framework using splines (Section 3.5.1), a complementary 3D angular velocity planner is developed to ensure accuracy and computational efficiency. The attitude planner (Section 3.4) provides a discrete sequence of rotation matrices in $SO(3)$ as a function of time, as shown in Figure 3.2. The angular velocity planning problem is thus formulated as determining ω at each timestamp, given a sequence of rotation matrices $R(t) \in SO(3)$ and time vector $t \in [0, \infty)$, expressed as:

$$\mathbb{J}\omega = f(R, t) : \mathbb{R}^{3 \times 3} \times [0, \infty) \rightarrow \mathbb{R}^3. \quad (3.20)$$

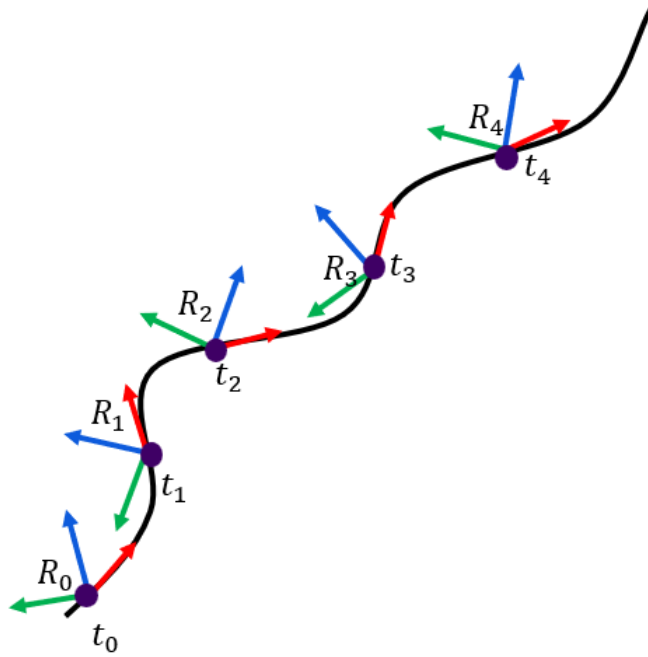


Figure 3.2: Interpolated spline points with the $SO(3)$ matrices representing the UGV orientation at every time instant in the future horizon.

The rotation matrices produced by the attitude planner (Section 3.4) represent the UGV's orientation over uneven terrain across a future time horizon. Each rotation matrix can be used to extract the underlying orientation, expressed as either Euler angles $\phi, \theta, \psi \in \mathbb{R}$ or

unit quaternions $q \in \mathbb{H}, \|q\| = 1$. Once the orientation angles are obtained, the angular velocity $\omega \in \mathbb{R}^3$ can be calculated using:

$$\omega = \frac{1}{2}J(q)\dot{q}, \quad (3.21)$$

for quaternions, where $J(q) \in \mathbb{R}^{3 \times 4}$ is the Jacobian matrix, or:

$$\omega = \begin{bmatrix} 0 & \sin \theta & \cos \theta \\ 0 & -\sin \theta \sin \phi & \cos \theta \cos \phi \\ 0 & \sin \phi & -\cos \phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}, \quad (3.22)$$

for Euler angles. It can be seen that the problem then reduces to determining the vector $\dot{q}(t)$ or $[\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ at each timestamp. To this end, the following methods are evaluated for their numerical accuracy and computational efficiency.

3.6.1 Forward Euler Angular Interpolation

The Forward Euler angular interpolation method computes the time derivatives of Euler angles $\phi, \theta, \psi \in \mathbb{R}$ to determine the angular velocity $\omega \in \mathbb{R}^3$ using a first-order finite difference approximation. Given a discrete sequence of rotation matrices $R(t_k) \in SO(3)$ from the attitude planner (Section 3.4) and corresponding timestamps $t_k, k \in \{0, \dots, N\}$, the Euler angles at each timestep are extracted, and their rates $\dot{\phi}, \dot{\theta}, \dot{\psi}$ are approximated by forward differencing. This method is computationally efficient, leveraging the simplicity of linear approximations, but may introduce numerical errors, particularly for lower time resolutions, which is preferred to maintain a tradeoff for faster computations, or rapidly changing orientations over uneven terrain. The angular velocity is then calculated using the Euler angle

transformation matrix as in Equation 3.22. The Forward Euler approximation is given by:

$$\begin{bmatrix} \dot{\phi}(t_k) \\ \dot{\theta}(t_k) \\ \dot{\psi}(t_k) \end{bmatrix} \approx \begin{bmatrix} \frac{\phi(t_{k+1}) - \phi(t_k)}{t_{k+1} - t_k} \\ \frac{\theta(t_{k+1}) - \theta(t_k)}{t_{k+1} - t_k} \\ \frac{\psi(t_{k+1}) - \psi(t_k)}{t_{k+1} - t_k} \end{bmatrix}, \quad \omega(t_k) = T(\phi(t_k), \theta(t_k)) \begin{bmatrix} \dot{\phi}(t_k) \\ \dot{\theta}(t_k) \\ \dot{\psi}(t_k) \end{bmatrix}, \quad (3.23)$$

where $T(\phi, \theta) \in \mathbb{R}^{3 \times 3}$ is the Euler angle transformation matrix for a 3-2-1 sequence, defined in Equation (3.22).

3.6.2 Rodrigues axis-angle method

The Rodrigues axis-angle method computes the angular velocity $\omega \in \mathbb{R}^3$ by determining the instantaneous rotation axis and angle between consecutive rotation matrices in a discrete sequence $R(t_k) \in SO(3)$ provided by the attitude planner (Section 3.4) at timestamps $t_k, k \in \{0, \dots, N\}$. This approach extracts the axis-angle representation from the relative rotation matrix between $R(t_k)$ and $R(t_{k+1})$, using the Rodrigues formula to derive the angular velocity as the rotation axis scaled by the angle rate. By leveraging the logarithmic map on $SO(3)$, it offers improved numerical stability compared to Forward Euler interpolation. The angular velocity is calculated as:

$$\omega(t_k) = \frac{\theta_k}{t_{k+1} - t_k} n_k \quad (3.24)$$

$$\theta_k = \cos^{-1} \left(\frac{\text{tr}(R(t_k)^T R(t_{k+1})) - 1}{2} \right), \quad (3.25)$$

$$n_k = \frac{1}{2 \sin \theta_k} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}, \quad (3.26)$$

where $\theta_k \in [0, \pi]$ is the rotation angle, $n_k \in \mathbb{R}^3$, $\|n_k\| = 1$ is the rotation axis, $R(t_k)^T R(t_{k+1}) = [R_{ij}]$, and $\omega(t_k) = 0$ if $|\theta_k| < \epsilon$ for a small threshold ϵ .

3.6.3 SLERP Interpolation

Spherical Linear Interpolation (SLERP) generates a smooth transition between two unit quaternions representing UGV orientations, leveraging the advantages of quaternion-based orientation over Euler angles for robust navigation over uneven terrain. Quaternions, defined in $\mathbb{H} \triangleq \{q = w + xi + yj + zk \mid w, x, y, z \in \mathbb{R}, i^2 = j^2 = k^2 = ijk = -1\}$, with unit norm $\|q\| = 1$, avoid gimbal lock—a singularity in Euler angle representations where degrees of freedom are lost at certain orientations (*e.g.*, pitch near $\pm\pi/2$). Additionally, quaternions provide a compact, numerically stable representation, ensuring smooth interpolation without discontinuities, which is critical for tactical UGV maneuvers. SLERP is chosen for its geodesic interpolation on the unit sphere $S^3 \subset \mathbb{H}$, guaranteeing constant angular velocity along the shortest path between orientations, unlike linear interpolation, which distorts rotational dynamics. Given a sequence of rotation matrices $R(t_k) \in SO(3)$ from the attitude planner (Section 3.4), converted to unit quaternions $p, q \in \mathbb{H}$, $\|p\| = \|q\| = 1$ at timestamps t_k, t_{k+1} , SLERP computes the interpolated quaternion as a function of time to derive the “smoother” angular velocity $\omega \in \mathbb{R}^3$. The time derivative $\dot{q}(t)$ is derived as follows:

$$\tilde{q}(t) = p \otimes (p^{-1} \otimes q)^{h(t)} \quad (3.27)$$

$$h(t) = \frac{t - t_k}{t_{k+1} - t_k}, \quad (3.28)$$

$$\text{Let } r = p^{-1} \otimes q, \quad (3.29)$$

$$\text{then } \tilde{q}(t) = p \otimes r^{h(t)}. \quad (3.30)$$

Differentiating $\tilde{q}(t)$ with respect to time using the product rule, since p is constant:

$$\dot{\tilde{q}}(t) = p \otimes \frac{d}{dt} (r^{h(t)}) \quad (3.31)$$

$$\frac{d}{dt} (r^{h(t)}) = r^{h(t)} \otimes \log r \cdot \frac{dh(t)}{dt}. \quad (3.32)$$

Therefore,

$$\dot{\tilde{q}}(t) = \tilde{q}(t) \otimes \log (p^{-1} \otimes q) \cdot \frac{1}{t_{k+1} - t_k}, \quad (3.33)$$

where $\otimes$ denotes quaternion multiplication, p^{-1} is the quaternion inverse, $\log : \mathbb{H} \rightarrow \mathbb{R}^4$ is the quaternion logarithm, and $h(t) \in [0, 1]$.

3.7 Proposed collision avoidance algorithms

The curvature of the spline trajectory, $\mathcal{R}(t)$, is derived as outlined in Section 3.3. Although the sequence of waypoints produced by the global path planner is guaranteed to be collision-free, the interpolated spline trajectory does not inherently ensure collision avoidance due to its continuous nature. A robust trajectory planner must therefore incorporate collision safety, enabling the generation of an alternative reference vector when the initial trajectory risks collision. Previous works have explored various approaches to this challenge. For instance, the authors in [1, 2] enhanced the safety region around a UAV by modeling the safe space as an ellipsoid in $\mathbb{R}^3$ and constructing a convex tetrahedron around the expanded ellipsoid to constrain the UAV's pose. Similarly, [47] defined a minimum bounding box around a UGV in $\mathbb{R}^2$ to ensure safe navigation. Applying an ellipsoidal model to a ground vehicle appears plausible for the present problem, where the coordinates of the spline trajectory could be checked against their position relative to the ellipsoid and replanned accordingly. While

maximizing safe spaces offers an optimal solution with potential extensions to advanced frameworks like Model Predictive Control (MPC), an alternative approach of constructing collision checks, such as safe walls around the UGV, could be computationally efficient. In this research, two algorithms are proposed to ensure collision avoidance, namely: (1) a safe corridor generation approach, and (2) maximization of the largest convex volume.

3.7.1 Safety corridor generation via point cloud partitioning

A safe corridor is generated by treating the initially interpolated spline trajectory, $\mathcal{R}(t)$, as a “candidate spline.” If no obstacles intersect the safe corridor, the candidate spline is used to generate the reference translational and angular velocities required by the UGV’s controller. However, if an intersection occurs between obstacles and the safe corridor, seed waypoints are added to the local trajectory segment such that the newly added waypoints are at a safe distance from the corridor’s boundaries. This approach prioritizes computational efficiency while ensuring collision safety, avoiding the computational complexity associated with geometric constraints such as tetrahedra or bounding boxes. Furthermore, the trajectory planner is designed without the convexity requirements typical of optimization-based methods, enhancing its practical applicability in dynamic, off-road environments.

The proposed safe corridor generation algorithm can be formally expressed as:

$$\Upsilon = f_{\text{corridor}}(\mathfrak{M}^{\mathcal{O}}, \mathcal{R}(\xi_i), [X'_i, Y'_i, Z'_i]), \quad \Upsilon \in \{0, 1\}, \quad (3.34)$$

where $\mathfrak{M}^{\mathcal{O}}$ represents the voxel map of obstacles, $\mathcal{R}(\xi_i)$ denotes the candidate spline trajectory, and $[X'_i, Y'_i, Z'_i]$ in $SO(3)$ represents the axes of the UGV’s local reference frame $\mathbb{J}$. The Boolean variable Υ returns 0 if $\mathcal{O} \cap \mathcal{R}(\xi_i) = \emptyset$ and 1 otherwise, indicating a potential collision. The following enumeration delineates the process to determine a potential collision.

1. **Local Obstacle Set Extraction:** From the UGV's position vector in the future horizon, $r(\xi_i)$, along the spline trajectory expressed in the inertial frame $\mathbb{I}$, a local obstacle set $\mathbb{I}\mathcal{O}_i^\xi$ is extracted from $\mathfrak{M}^\mathcal{O}$. This involves identifying the nearest neighboring obstacles within a sphere of radius $r_\mathcal{O}$ around the spline position $r(\xi_i)$ using a k-nearest neighbor (kNN) search technique [68].
2. **Obstacle Set Partitioning:** The local obstacle set is transformed into the UGV's reference frame $\mathbb{J}$. The transformed local obstacle set, $\mathbb{J}\mathcal{O}_i^\xi$, is partitioned into two subsets based on the vehicle's Z' axis as follows:
 - $\mathbb{J}\mathcal{O}_i^{\xi+}$: Includes obstacles such that $q_j \cdot Z' < 0$, where $q_j \in \mathbb{J}\mathcal{O}_i^{\xi+}$ represents each element in the obstacle set lying on the positive Y' axis.
 - $\mathbb{J}\mathcal{O}_i^{\xi-}$: Includes obstacles such that $q_j \cdot Z' > 0$, where $q_j \in \mathbb{J}\mathcal{O}_i^{\xi-}$ represents each element in the obstacle set lying on the negative Y' axis.
3. **Closest Obstacle Identification:** After partitioning, the algorithm identifies the closest obstacle within the sets $\mathbb{J}\mathcal{O}_i^{\xi+}$ and $\mathbb{J}\mathcal{O}_i^{\xi-}$, calculating the distances $d_\mathcal{O}^+$ and $d_\mathcal{O}^-$ respectively. If no obstacles are present in a set, a threshold distance $\epsilon_\mathcal{O}$ is assigned to the respective distance variable.
4. **Plane Parameters Determination:** Using the distance variables and the UGV's orientation, the parameters for two planes parallel to the $X' - Z'$ plane are determined. These planes, positioned at distances $d_\mathcal{O}^+$ and $d_\mathcal{O}^-$ from the UGV, define the safe corridor boundaries. Their equations are:

$$\text{plane's } \Pi^+ : X' \cdot x + Y' \cdot y + Z' \cdot (z - d_\mathcal{O}^+) = 0, \quad (3.35)$$

$$\Pi^- : X' \cdot x + Y' \cdot y + Z' \cdot (z + d_\mathcal{O}^-) = 0. \quad (3.36)$$

5. **Collision Boolean Determination:** Once the planes Π^+ and Π^- are defined, the displacement vectors from the spline trajectory $\mathcal{R}(\xi_i)$ to the planes are calculated. These displacements, denoted $\vec{d}(\Pi^+, \mathcal{R}(\xi_i))$ and $\vec{d}(\Pi^-, \mathcal{R}(\xi_i))$, represent the shortest vectors from points on the trajectory to the respective planes. The algorithm checks for a change in the directional component of these vectors along the planes' normal vector Z' . A change in the sign of the scalar projections indicates that the trajectory crosses a plane, implying a potential collision. In such cases, the function returns $\Upsilon = 1$; otherwise, it returns $\Upsilon = 0$, indicating no collision.

Algorithm 7 Safety Corridor Generation for UGV Navigation

Input: Obstacle set $\mathcal{O}$, trajectory $\mathcal{R}(\xi)$, local reference frame $[X'_i, Y'_i, Z'_i]$
Output: Boolean collision indicator Υ

```

1: function SAFETYCORRIDOR( $\mathcal{O}, \mathcal{R}, [X'_i, Y'_i, Z'_i]$ )
2:   Extract local obstacle set  $\mathbb{I}\mathcal{O}_i^\xi$  using kNN search
3:   Transform  $\mathbb{I}\mathcal{O}_i^\xi$  to vehicle's frame  $\mathbb{J}\mathcal{O}_i^\xi$ 
4:   Partition  $\mathbb{J}\mathcal{O}_i^\xi$  into  $\mathbb{J}\mathcal{O}_i^{\xi+}$  and  $\mathbb{J}\mathcal{O}_i^{\xi-}$  that lie on either side of the UGV's  $X'Z'$  plane
5:   Identify closest obstacles and calculate distances  $d_{\mathcal{O}}^+, d_{\mathcal{O}}^-$ 
6:   Determine planes  $\Pi^+$  and  $\Pi^-$  at distances  $d_{\mathcal{O}}^+$  and  $d_{\mathcal{O}}^-$ 
7:   Calculate displacement vectors  $\vec{d}(\Pi^+, \mathcal{R}(\xi))$  and  $\vec{d}(\Pi^-, \mathcal{R}(\xi))$ 
8:   Check directional components along  $Z'$ 
9:   if change in sign of displacement vectors then
10:      $\Upsilon \leftarrow 0$  ▷ Collision detected
11:   else
12:      $\Upsilon \leftarrow 1$  ▷ No collision
13:   end if
14:   return  $\Upsilon$ 
15: end function

```

3.7.2 Safe Convex Volume Maximization for Collision Avoidance

The second approach builds on the algorithm developed by the authors in [1, 2], which maximizes a convex shape around a UAV, adapted here for an off-road UGV navigating uneven terrain. The position vectors of the UGV along the spline trajectory, $r(\xi_i)$, are

coordinates in $\mathbb{R}^3$. The problem of identifying imminent collisions can be defined as checking whether the set of points constituting the trajectory satisfies a specific condition in $\mathbb{R}^3$. As long as the coordinates of these trajectory position vectors remain within a traversable region of the configuration space defined in $\mathbb{R}^3$, the trajectory is safe. Collisions occur when these coordinates enter non-traversable regions. To address this, a volumetric boundary is established around the UGV in $\mathbb{R}^3$ and expanded outward until it encounters obstacles, if any. If no obstacles are detected, the expansion is halted at a predefined limit. Given the requirement for safe volume expansion, a convex shape is chosen, leveraging the availability of efficient solvers for convex optimization problems. Formulating this volume expansion as a convex optimization problem is advantageous due to the extensive development of convex solvers, as noted in [1, 2]. However, this approach may introduce computational bottlenecks. Nevertheless, the algorithm shows promise for application in this context and is therefore investigated.

The ellipsoid is defined as:

$$\mathcal{E} \triangleq \{x \in \mathbb{R}^3 \mid x^T P x + q^T x + r \leq 0\}, \quad (3.37)$$

where $P \in \mathbb{R}^{3 \times 3}$ is a symmetric positive definite matrix defining the ellipsoid's shape, $q \in \mathbb{R}^3$ influences its center, and $r \in \mathbb{R}$ is a scalar offset. The UGV's boundary points, $\mathcal{X} = \{x_j \in \mathbb{R}^3 \mid j = 1, \dots, N\}$, are the vertices of a rectangular prism centered at the UGV's position. Obstacle points, $\mathcal{Y} = \{y_i \in \mathbb{R}^3 \mid i = 1, \dots, M\}$, are extracted from the voxel map $\mathfrak{M}^\mathcal{O}$ within a sphere of radius $r_\mathcal{O}$. The goal is to find an ellipsoid $\mathcal{E}$ such that $\mathcal{X} \subseteq \mathcal{E}$ and $\mathcal{Y} \cap \mathcal{E} = \emptyset$, ensuring the UGV is separated from obstacles.

The problem is formulated as a quadratically constrained quadratic program (QCQP) to separate the set, $\mathcal{X}$ and $\mathcal{Y}$. The optimization variables are the ellipsoid parameters, struc-

tured as a vector $z = [P_{11}, P_{22}, P_{33}, P_{12}, P_{13}, P_{23}, q_1, q_2, q_3, r]^T \in \mathbb{R}^{10}$, where P is symmetric ($P_{ij} = P_{ji}$). The QCQP is:

$$\text{minimize: } f_0(x) = 1, \quad (3.38)$$

$$\text{subject to: } f_j(x) = x_j^T P x_j + q^T x_j + r \leq 0, \quad \forall x_j \in \mathcal{X}, \quad (3.39)$$

$$f_i(y) = y_i^T P y_i + q^T y_i + r \geq 1, \quad \forall y_i \in \mathcal{Y}, \quad (3.40)$$

$$P \succ 0, \quad (3.41)$$

where $P \succ 0$ ensures positive definiteness, guaranteeing that $\mathcal{E}$ is an ellipsoid [69].

The SDP is solved using the SDPA library [70], which employs a primal-dual interior point method to handle the positive semidefinite constraints efficiently. The solver subsequently provides the ellipsoid parameters z , from which P , q , and r are extracted. The actual center of the optimal ellipsoid center is then computed,

$$x_c = -\frac{1}{2}P^{-1}q. \quad (3.42)$$

Once the ellipsoid is defined, the trajectory points $\mathcal{R}(\xi_i)$ are evaluated against the condition:

$$\mathcal{R}^T(\xi_i)P\mathcal{R}(\xi_i) + q^T\mathcal{R}(\xi_i) + r \leq 0. \quad (3.43)$$

If all points satisfy this inequality, the trajectory is collision-free. If any point violates it, indicating it lies outside $\mathcal{E}$, new seed waypoints are introduced to generate another candidate trajectory. This approach diverges from [1, 2], which generates affine half-planes to approximate the ellipsoid for FMPC constraints (Section 5.2.2). By focusing solely on the ellipsoid, computational complexity is reduced, making the method suitable for real-time UGV navigation in dynamic environments. The SDPA library's robust handling of SDP constraints

ensures reliable solutions, though computational overhead remains a consideration compared to the safe corridor approach in Section 3.7.1.

3.8 Numerical case studies

The sequence of waypoints $\mathcal{W}(r)$ produced by the path planner for varying levels of stealth is shown in Figure 2.3. As delineated earlier, the trajectory planning module appends to the path planning module by smoothening the waypoints, adhering to collision safety as well as kinematic feasibility, plans the attitude along the trajectory and finally produces a time-scaled trajectory conforming to the tactical regulation of the velocity using the $\chi(\xi)$ velocity weighing function as described in Section 3.5.1. To this end, the spline based trajectory planning algorithms were implemented on the results obtained from the path planning module. Specifically, the case represented by the blue points in the Figure 3.3 were chosen for generating the trajectory $\mathcal{R}(t)$. The Hermite interpolated spline is shown in Figure 3.4

3.8.1 Results on Attitude Planning

With the cubic spline trajectory $r(\xi(t))$, $\xi \in [0, 1]$, $t \in [0, \infty)$, defined as per Section ??, the UGV's attitude along the spline is computed to derive the rotation matrix ${}^{\mathbb{I}}R(\xi)^{\mathbb{J}} \in SO(3)$, which maps the UGV's body-fixed frame $\mathbb{J}$ to the inertial frame $\mathbb{I}$. Initially, the Frénet-Serret frame, comprising the tangent $T(\xi) \in \mathbb{R}^3$, normal $N(\xi) \in \mathbb{R}^3$, and binormal $B(\xi) \in \mathbb{R}^3$ vectors as defined in Section 3.4, is computed and illustrated in Figure 3.5. The figure reveals significant reorientation of $N(\xi)$ and $B(\xi)$ at inflection points where the curvature $\gamma(\xi) \rightarrow 0$. This twisting, a known issue in $\mathbb{R}^2$ trajectories, is exacerbated in $\mathbb{R}^3$ due to

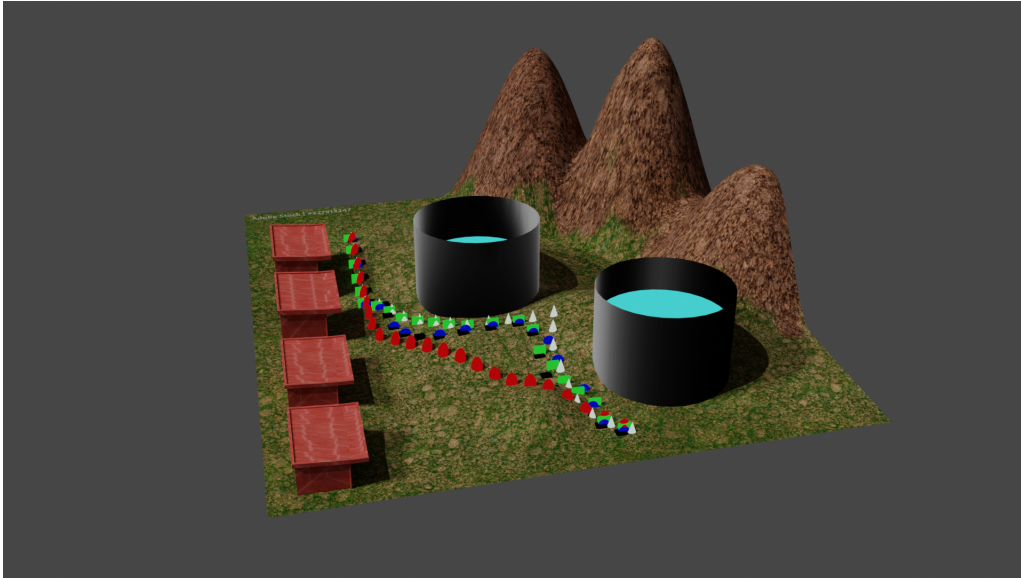


Figure 3.3: Reference waypoints: Sequence of waypoints $\mathcal{W}(r)$ produced by the path planner with different levels of tactical parameters as in Chapter 2. The red markers represent the most reckless path whereas the white markers represent the most tactical path

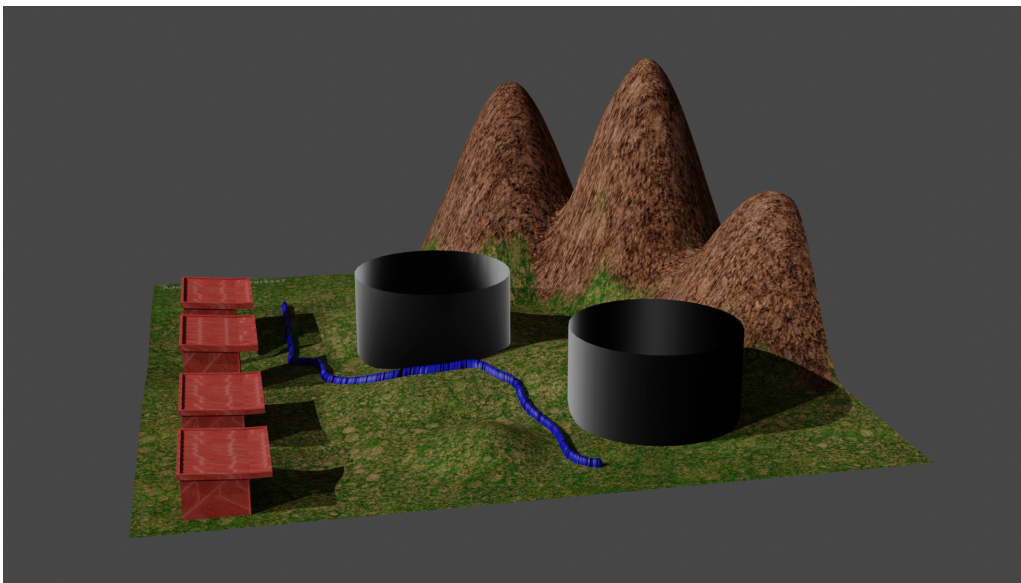


Figure 3.4: Path smoothening and time scaling: the figure shows the C^2 continuous and smoothened waypoints received from the path planner.

additional curvature components, leading to abrupt changes in the UGV's orientation that can make the direct use of the $T(\xi)$, $N(\xi)$ and $B(\xi)$ axes as the UGV orientation axes impractical.

To mitigate this, rotation-minimizing frames (RMFs) are employed, as detailed in Sections 3.4.1 and 3.4.2. The RMFs generate vectors $\hat{T}(\xi)$, $\hat{N}(\xi)$, and $\hat{B}(\xi)$, forming a matrix ${}^{\mathbb{I}}R(\xi)^{\mathbb{J}}$ that remains in $SO(3)$, verified by a determinant of 1. Applying RMFs to the same spline trajectory, as shown in Figure 3.6, demonstrates a significant reduction in reorientation. Unlike the Frénet-Serret frame, the RMF vectors $\hat{N}(\xi)$ and $\hat{B}(\xi)$ exhibit smooth transitions, minimizing twisting and ensuring a stable UGV orientation along the trajectory.

Reorientation of Frames

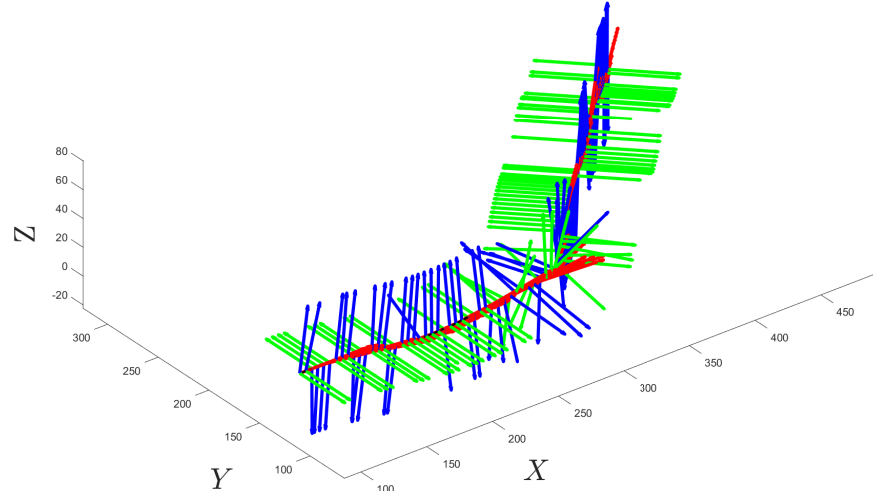


Figure 3.5: Frénet-Serret frames determined along the spline trajectory: The plot illustrates how the Frénet-Serret frames undergo twisting at the inflection points of the trajectory, causing reorientation of the normal and binormal axes.

The effectiveness of RMFs is quantitatively assessed using curvature $\gamma(\xi)$ and torsion $\tau(\xi)$, computed via two methods: inherent spline geometry and values derived from the orientation matrix ${}^{\mathbb{I}}R(\xi)^{\mathbb{J}}$.

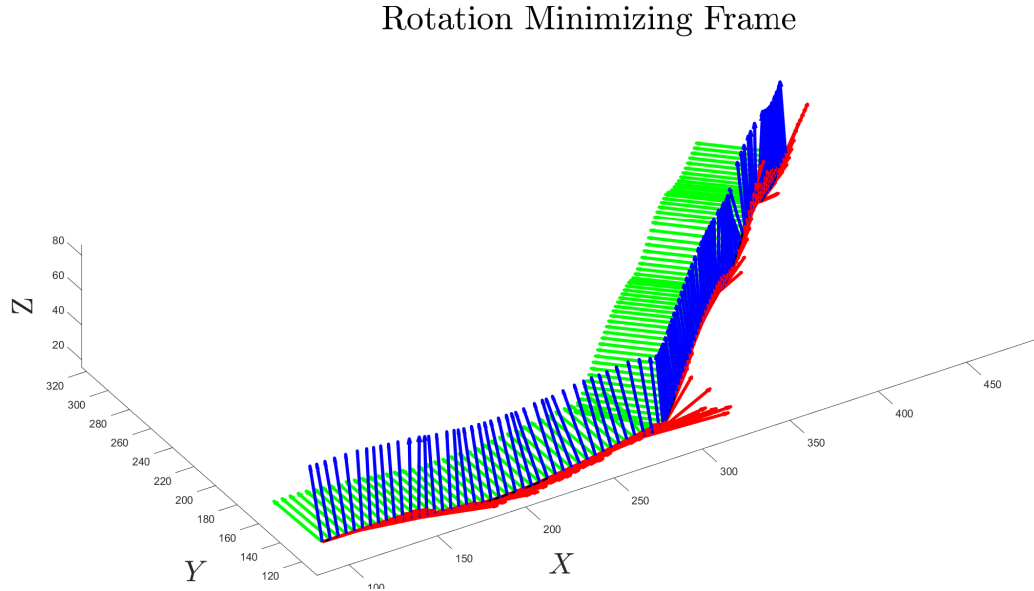


Figure 3.6: Rotation-minimizing frames to determine the UGV orientation along the spline trajectory: The plot shows the application of rotation-minimizing frames (RMFs) to maintain smooth UGV orientation along the trajectory, minimizing unnecessary twisting and ensuring stable orientation.

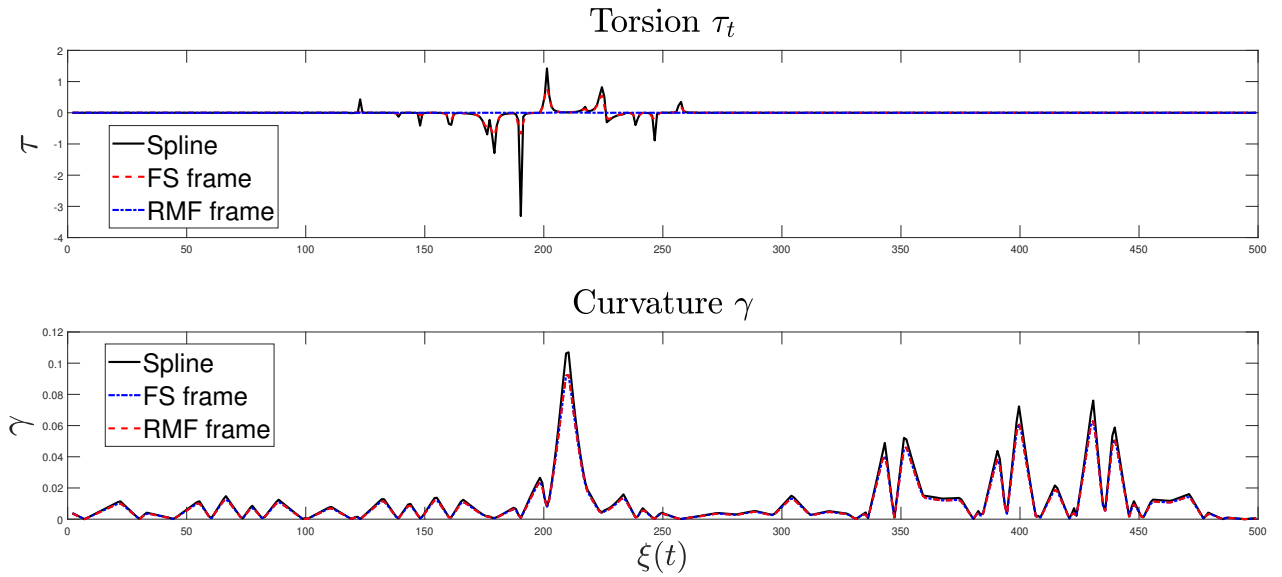


Figure 3.7: Torsion and curvature comparison for the spline trajectory: The top plot shows the torsion values, while the bottom plot depicts the curvature along the spline trajectory. The near-zero torsion for RMFs demonstrates their effectiveness in minimizing twisting, making them suitable for UGV orientation.

The results, depicted in Figure 3.7, show that RMFs achieve near-zero torsion ($\tau(\xi) \approx 0$) across the trajectory, in contrast to the Frénet-Serret frame, which exhibits significant torsion spikes at inflection points. The curvature $\gamma(\xi)$ remains consistent for both methods, indicating that RMFs preserve the spline’s geometric properties while eliminating torsional effects. This reduction in torsion confirms that RMFs are a suitable choice for representing UGV orientation, as they ensure smooth and predictable pose changes critical for controller stability in off-road navigation.

Further analysis compares the performance of the two RMF methods: Wang’s double reflection (Section 3.4.1) and the proposed double projection (Section 3.4.2). Both methods produce comparable torsion values, differing by an order of 10^{-3} with the proposed double projection method exhibiting almost zero torsion despite elevated curvature, as shown in Figure 3.8, indicating enhanced effectiveness in minimizing twisting. However, computational efficiency, a key requirement for real-time replanning in tactical scenarios, differs significantly. The simulation time for both algorithms was evaluated across varying resolutions of the time vector (from 10 to 1000 sample points along the spline), as shown in Figure 3.9. The proposed double projection method consistently exhibits lower computational time compared to Wang’s double reflection method, with the gap widening as resolution increases. At 1000 sample points, the double projection method is approximately 20% faster, likely due to fewer operations per iteration — projections versus reflections and cross products — reducing the computational burden. This efficiency makes the double projection method a more viable option for real-time deployment on resource-constrained UGV platforms like the Clearpath Husky, where rapid replanning is essential for dynamic environments.

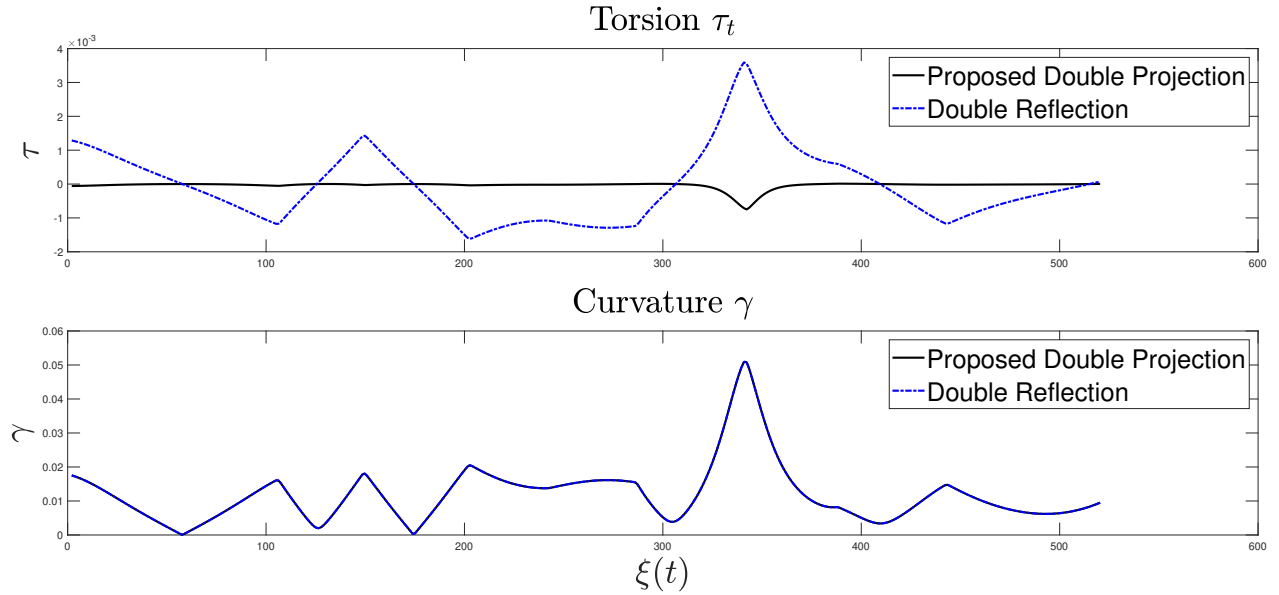


Figure 3.8: The torsion (top) and curvature (bottom) of the splines using Wang’s double reflection technique compared to those obtained using the proposed double projection technique indicate reduced torsion values almost equal to zero but preservation of the splines curvature

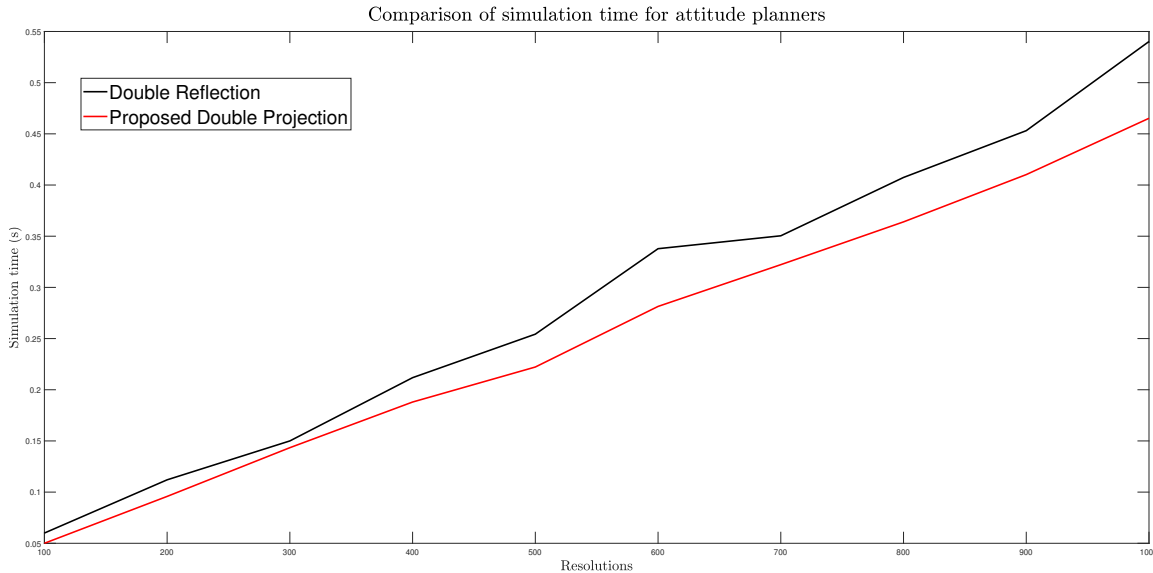


Figure 3.9: Comparison of simulation time for attitude planners: The plot compares the computational time of Wang’s double reflection method (literature) and the proposed double projection method across varying time vector resolutions. The double projection method consistently shows lower computational time, suggesting better suitability for real-time applications.

3.8.2 Results on Spline Parameterization

Following the Hermite interpolation and attitude planning along the spline trajectory, as described in Sections 3.3 and 3.4, the time-scaling algorithm from Section 3.5 is applied to parameterize the position vectors $r(\xi(t))$, $\xi \in [0, 1]$, $t \in [0, \infty)$, as a function of time. The parameter $\xi(t)$ is computed by solving the integral 3.17, relating arc length to time via $\frac{dt}{d\xi} = \frac{\|r'(\xi)\|}{V}$. Initially, a constant velocity $V = 5 \text{ ms}^{-1}$ is imposed to evaluate the baseline performance.

The resulting velocity profile, shown in Figure 3.11, confirms a constant magnitude of 5 ms^{-1} , aligned with the tangent direction at each point on the spline. The velocity components in the inertial frame $\mathbb{I}$, denoted $v_x(t)$, $v_y(t)$, $v_z(t)$, and as functions of ξ , are presented in Figure 3.10. The component $v_z(t)$ in the inertial frame, exhibits peaks corresponding to elevation changes, reflecting the spline's 3D geometry.

A variable velocity profile is then generated using the op-tempo weighing function $\chi(\xi)$ from Section 3.5.1, with parameters $\beta_1 = 0.5$, $\beta_2 = \beta_3 = \beta_4 = 1.5$, and $\beta_5 = 0.1$. The nominal velocity v_{nominal} is set to the maximum allowable velocity $V_{\text{max}} = 5 \text{ ms}^{-1}$, and the tactical velocity is computed as $V(\xi) = \chi(\xi) \cdot v_{\text{nominal}}$. Figure 3.12 plots $V(\xi)$ against the spline's elevation $z(\xi)$, showing a decrease to approximately 4 ms^{-1} during elevation climbs and an increase to v_{nominal} on flat terrain. Figure 3.13 shows $V(\xi)$ decreasing near shelters (*e.g.*, where $d_{\mathcal{O}}(\xi) < 30 \text{ m}$) and increasing when $d_{\mathcal{O}}(\xi) > 40 \text{ m}$.

The velocity components in both inertial frame $\mathbb{I}$ and body frame $\mathbb{J}$ are analyzed in Figure ?? for uneven terrain and Figure 3.14 for flat terrain. On uneven terrain, the inertial frame components $v_x(t)$, $v_y(t)$, $v_z(t)$ reflect the trajectory's geometry, while body frame components $\hat{v}_y(t)$ and $\hat{v}_z(t)$ remain near zero, confirming non-holonomic constraints. On flat terrain, $v_z(t)$ and $\hat{v}_z(t)$ are minimal, validating the frame transformation.

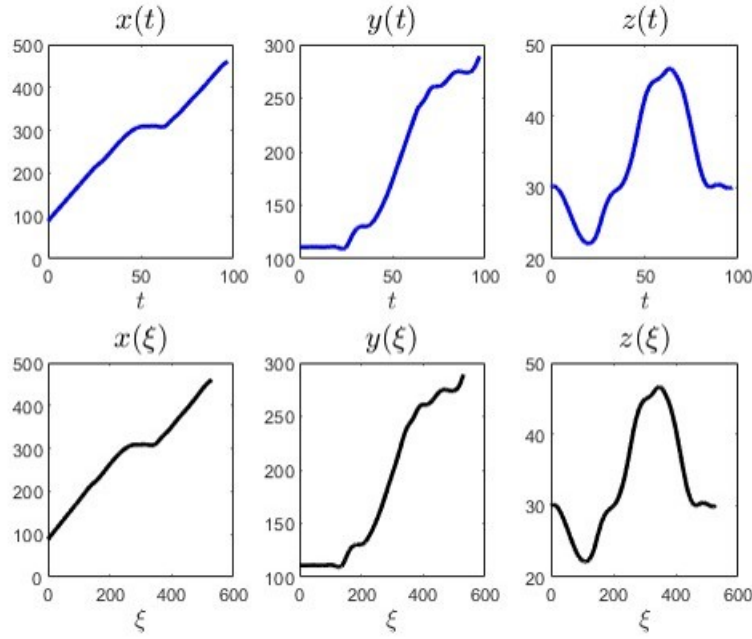


Figure 3.10: Components of the velocity vector for constant velocity: The top row shows the components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$ as a function of time, while the bottom row shows them as a function of the spline parameter ξ .

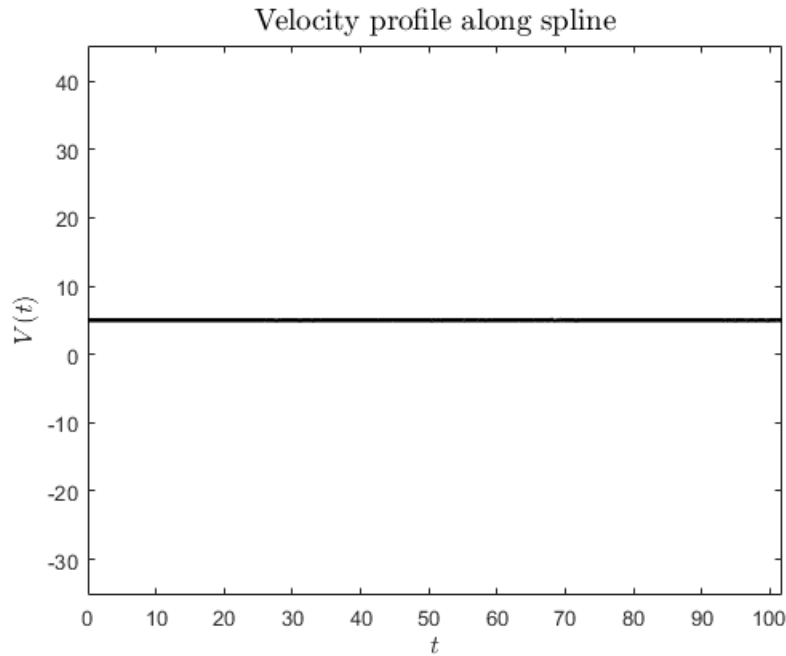


Figure 3.11: Velocity profile along the spline for constant velocity: The graph shows a constant velocity of 5 ms⁻¹ over time, representing uniform motion along the cubic spline.

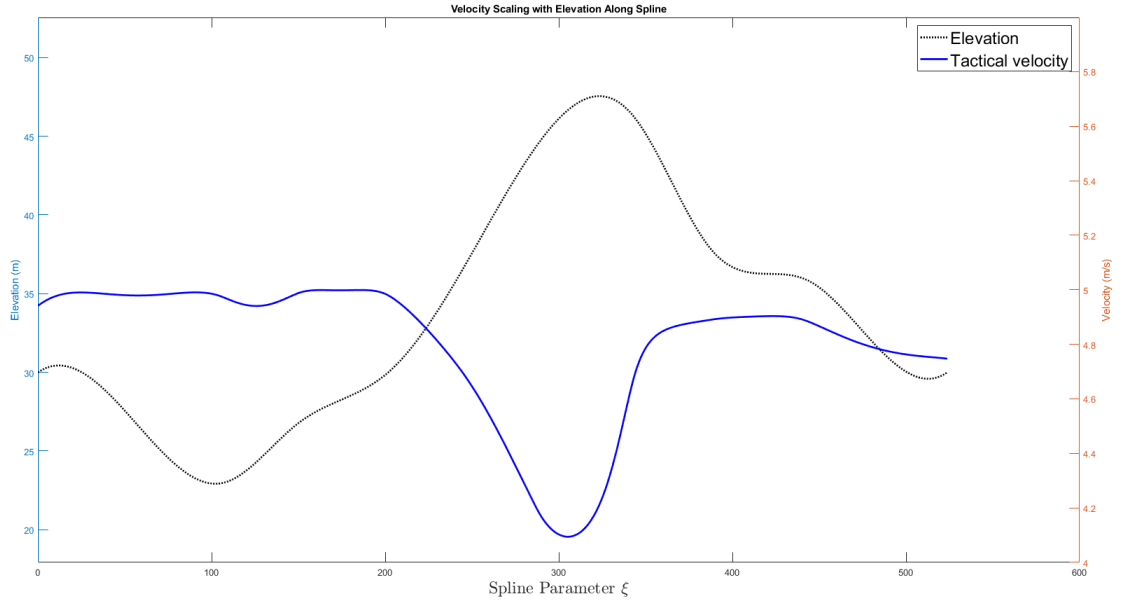


Figure 3.12: Velocity scaling with elevation along the spline: The plot shows the tactical velocity $V(\xi)$ modulated by the op-tempo function $\chi(\xi)$, decreasing during elevation climbs and increasing on flat terrain.

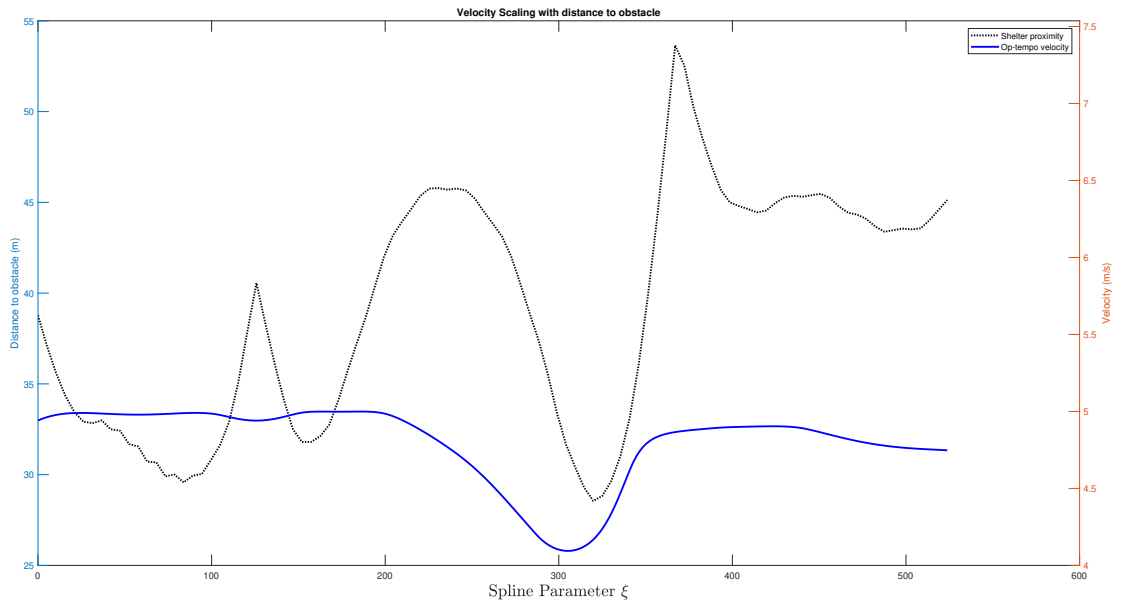


Figure 3.13: Velocity scaling with shelter proximity along the spline: The plot shows the tactical velocity $V(\xi)$ decreasing near shelters (low $d_O(\xi)$) and increasing when farther away.

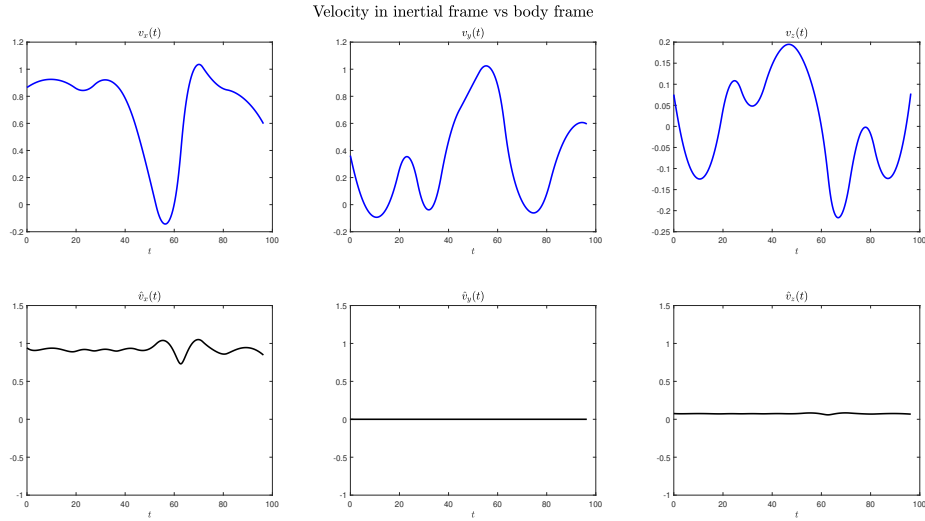


Figure 3.14: Velocity components in inertial and body frames on uneven terrain: The top row shows components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$, while the bottom row shows $\hat{v}_x(t)$, $\hat{v}_y(t)$, $\hat{v}_z(t)$ in the body frame $\mathbb{J}$, with near-zero $\hat{v}_y(t)$ and $\hat{v}_z(t)$ confirming non-holonomic constraints.

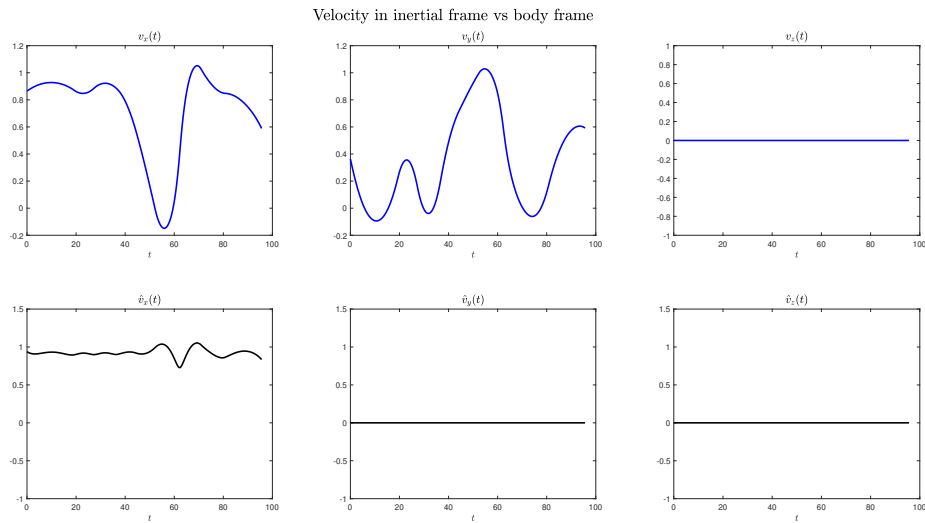


Figure 3.15: Velocity components in inertial and body frames on flat terrain: The top row shows components $v_x(t)$, $v_y(t)$, $v_z(t)$ in the inertial frame $\mathbb{I}$, while the bottom row shows $\hat{v}_x(t)$, $\hat{v}_y(t)$, $\hat{v}_z(t)$ in the body frame $\mathbb{J}$, with minimal $v_z(t)$ and $\hat{v}_z(t)$ due to flat terrain.

3.8.3 Results on Reference Angular Velocity Planning

The angular velocity planner, as outlined in Section 3.6, computes the angular velocity $\omega(t) = [\omega_x(t), \omega_y(t), \omega_z(t)]^T \in \mathbb{R}^3$ from a sequence of rotation matrices $R(t) \in SO(3)$ provided by the attitude planner at discrete timestamps t . Four methods—SLERP, SQUAD, Rodrigues, and Forward Euler—are evaluated for accuracy and computational efficiency. To establish a ground truth, Euler angles are analytically defined as $\phi(t) = \frac{\pi}{4} \sin(t)$, $\theta(t) = \frac{\pi}{6} \cos(t)$, $\psi(t) = \frac{\pi}{3} \sin(t)$, with their derivatives computed as $\dot{\phi}(t) = \frac{\pi}{4} \cos(t)$, $\dot{\theta}(t) = -\frac{\pi}{6} \sin(t)$, $\dot{\psi}(t) = \frac{\pi}{3} \cos(t)$. The ground truth angular velocity is obtained using the Jacobian transform in (3.22). These Euler angles and the time vector t are then used as inputs to the four algorithms, simulating the process of extracting angles from $R(t)$ as in prior sections, and the resulting $\omega(t)$ is compared to the ground truth.

Figure 3.16 shows the angular velocities $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ computed by each method at a high time vector resolution, alongside the ground truth. All methods converge closely to the ground truth, accurately tracking the sinusoidal variations, with RMS errors below 10^{-3} . However, at lower resolutions, as shown in Figure 3.17, the methods struggle to capture rapid changes in $\omega(t)$, particularly for $\omega_x(t)$ and $\omega_z(t)$, where Forward Euler exhibits the largest deviations due to its first-order approximation.

The accuracy and computational efficiency of the methods are further analyzed across varying resolutions and it was observed that the mean error between each method's $\omega(t)$ and the ground truth decreases with increasing resolution. SLERP and Rodrigues achieve nearly identical errors, while Forward Euler has the highest error. Notably, Rodrigues' computational time is consistently lower than SLERP's across all resolutions. SQUAD's computational time on the other hand rises sharply due to its iterative SLERP-of-SLERP approach, while Forward Euler remains the fastest but with significantly higher error. At a

resolution of 40 points, Rodrigues achieves an RMS error of approximately 10^{-3} with a computational time of 0.08 seconds, slightly outperforming SLERP's 0.1 seconds for the same error. Furthermore, use of Rodrigues' method circumvents the calculation of orientation angles by leveraging the $SO(3)$ matrices directly to calculate only one axis angle. These results thus indicate that Rodrigues offers the best combination of accuracy and computational efficiency for real-time UGV navigation. Its near-identical accuracy to SLERP, coupled with lower computational time, makes it the preferred method, especially at lower resolutions suitable for resource-constrained platforms like the Clearpath Husky. Higher resolutions improve accuracy but increase computational demands, impacting upstream processes like RMF calculation and time scaling. Rodrigues' efficiency ensures tactical responsiveness while maintaining accuracy.

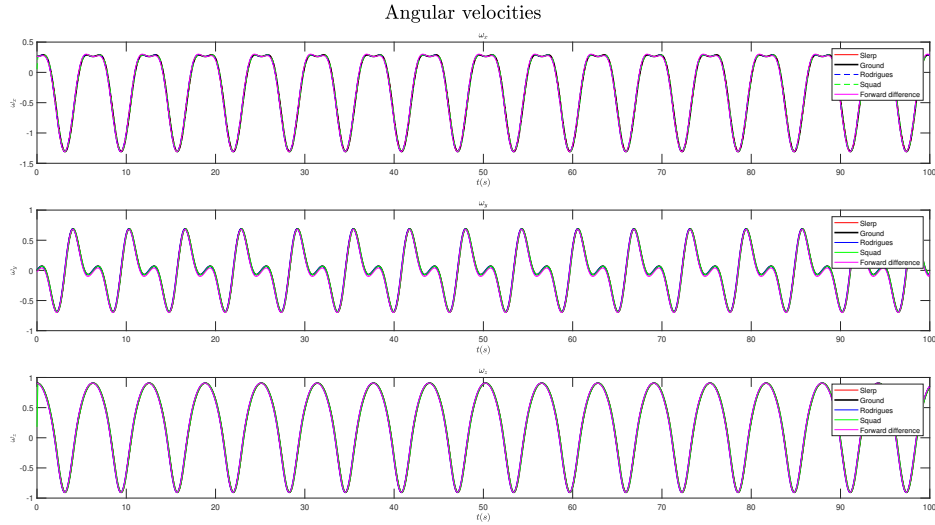


Figure 3.16: Angular velocities at high resolution: The plots show $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ computed by SLERP, SQUAD, Rodrigues, and Forward Euler at 640 time points, compared to the ground truth. All methods converge closely at high resolution, but struggle at lower resolutions.

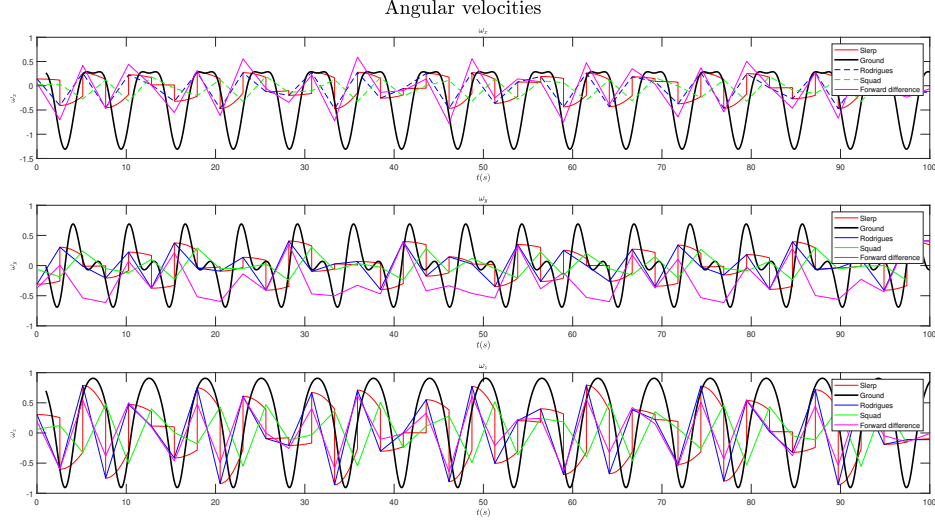


Figure 3.17: Angular velocities at low resolution: The plots show $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ computed by SLERP, SQUAD, Rodrigues, and Forward Euler at 40 time points, compared to the ground truth. Rodrigues, SLERP and SQUAD track the ground truth more effectively than Forward Euler.

3.8.4 Results on Collision Avoidance

Following the generation of the spline trajectory $\mathcal{R}(\xi(t))$, the collision avoidance module ensures safe navigation by generating alternative reference vectors when obstacles are detected. This section evaluates two approaches: the safety corridor generation algorithm (Section 3.7.1) and the safe convex volume maximization using ellipsoids (Section 3.7.2). The simulated environment, shown in Figure 3.18, includes four obstacles with distinct shapes (circles, squares, and ellipses), one incorporating noise to mimic real-world point cloud irregularities, posing a realistic challenge for the UGV's trajectory planning.

For the safety corridor approach, the algorithm analyzes the local obstacle set using the UGV's reference frame $\mathbb{J}$ from the attitude planner, partitioning it to compute distances $d_{\mathcal{O}}^+$ and $d_{\mathcal{O}}^-$ to the nearest obstacles. These distances define the corridor boundaries, ensuring

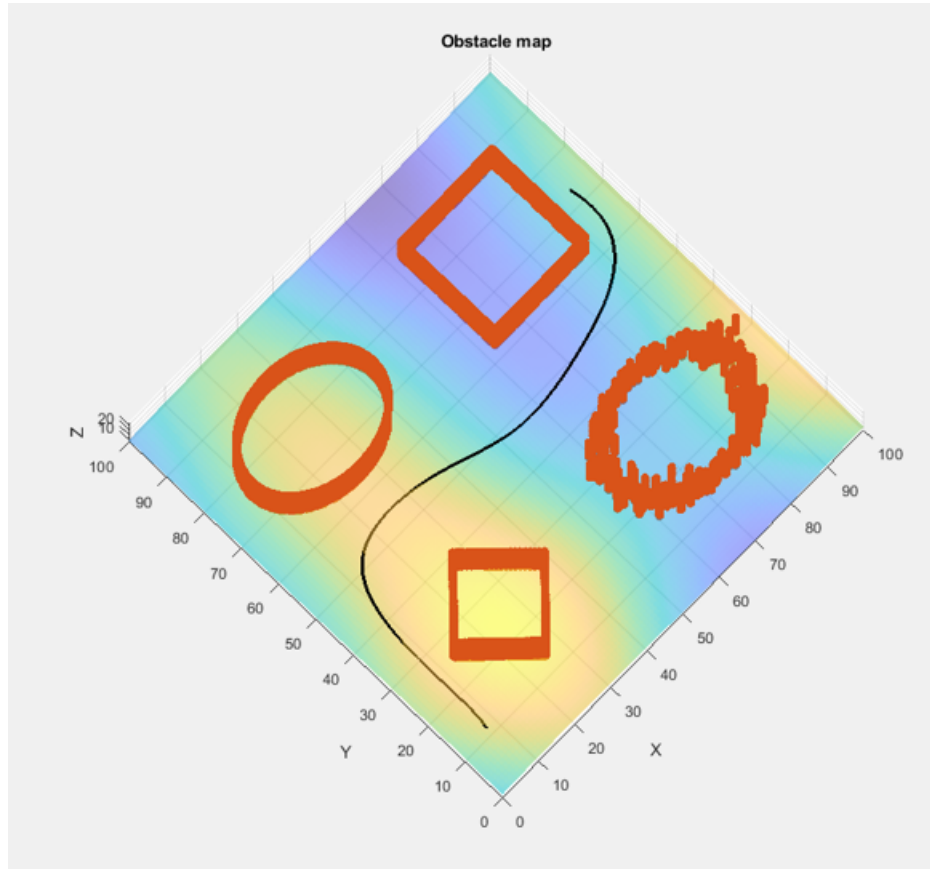


Figure 3.18: Obstacle map with various geometric shapes and a noise-inclusive model to simulate real-world conditions: The map displays different geometric obstacles (circles, squares, and ellipses) with noise added to simulate environmental uncertainty, providing a complex terrain for UGV path planning and obstacle avoidance.

the trajectory remains free from obstructions. Figure 3.19 illustrates the resulting safety corridor, with the space between the blue planes indicating safe navigation zones and red/black areas marking potential collisions. The trajectory dynamically adjusts within the corridor, maintaining a safe path around obstacles, such as rerouting near the noisy elliptical obstacle at $(x, y) \approx (80, 80)$. The corridor adapts to the trajectory's shape and curvature, encapsulating the UGV's operational space and ensuring collision-free navigation throughout the environment.

The ellipsoid-based approach maximizes a convex volume around the UGV to sepa-

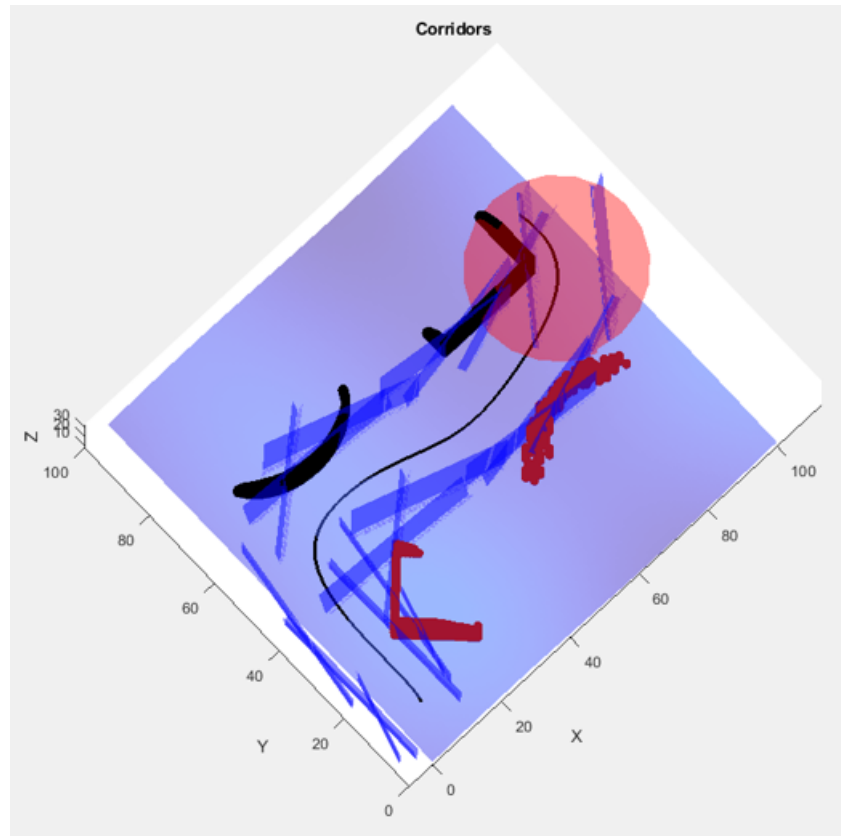


Figure 3.19: Safety corridor generated by the algorithm, highlighting the collision avoidance strategy: The figure shows the safety corridor (in blue) around the trajectory, with red indicating areas of potential collision. The algorithm adjusts the path within the corridor to ensure safe navigation and obstacle avoidance.

rate the trajectory from obstacles. Figure 3.20 shows two ellipsoids centered at points along the trajectory, with a red line connecting their centers from $(x, y, z) \approx (5, 5, -2)$ to $(x, y, z) \approx (15, 15, 2)$, marked by green and red stars, respectively. The ellipsoids, depicted as blue wireframes, enclose the trajectory points, ensuring they remain within a safe volume. Surrounding black dots represent obstacle points, positioned outside the ellipsoids, confirming that the trajectory avoids collisions. Figure 3.21 provides a closer view, highlighting the ellipsoid's intersection at the trajectory midpoint, with the red line segment enclosed within the safe volume. The ellipsoids' shapes adapt to the local obstacle distribution, expanding until they reach a predefined limit or encounter obstacles, ensuring the UGV's path remains collision-free.

These results validate the effectiveness of both collision avoidance methods. The safety corridor approach dynamically adjusts the trajectory to navigate complex environments, while the ellipsoid method ensures safety by enclosing the trajectory within a convex volume, avoiding obstacles even in noisy conditions. Both methods demonstrate practical applicability for off-road UGV navigation in hostile environments.

3.9 Conclusions and Future Work

This chapter developed a comprehensive trajectory planning framework for off-road UGVs in GNSS-denied, hostile environments, addressing the core questions outlined in Section 3.1.2. First, the question of producing continuous trajectories with tactical velocity regulation was addressed through a cubic spline-based approach (Section 3.3) and an op-tempo velocity framework (Section 3.5.1). The spline trajectory $\mathcal{R}(t)$ ensures C^2 continuity, while the weighing function $\chi(\xi)$ modulates velocity based on curvature and obstacle proximity, enabling adaptive speed adjustments critical for tactical navigation. Second, terrain-aware

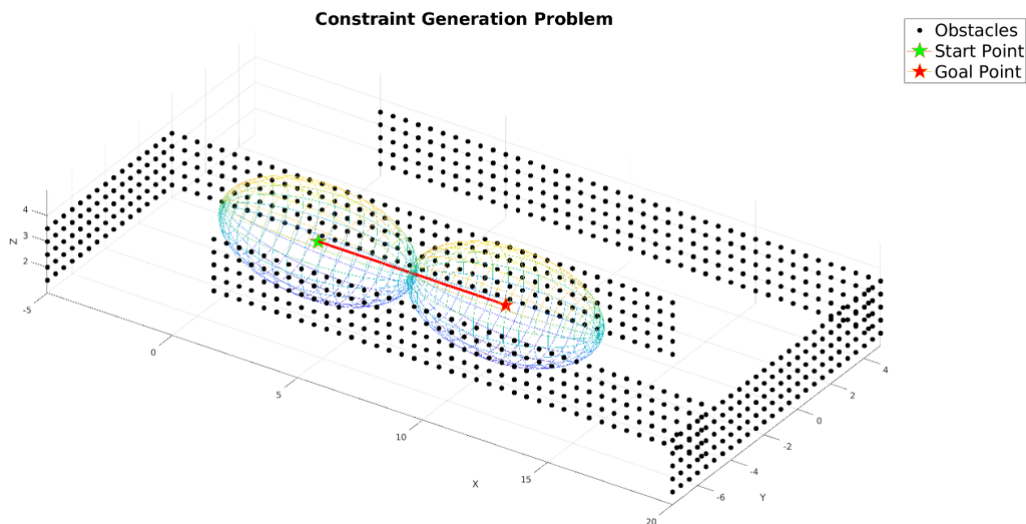


Figure 3.20: Ellipsoid-based collision avoidance: The figure shows two ellipsoids (blue wireframes) centered along the trajectory, connected by a red line from $(x, y, z) \approx (5, 5, -2)$ to $(x, y, z) \approx (15, 15, 2)$, marked by green and red stars. Obstacle points (black dots) lie outside the ellipsoids, ensuring a collision-free path.

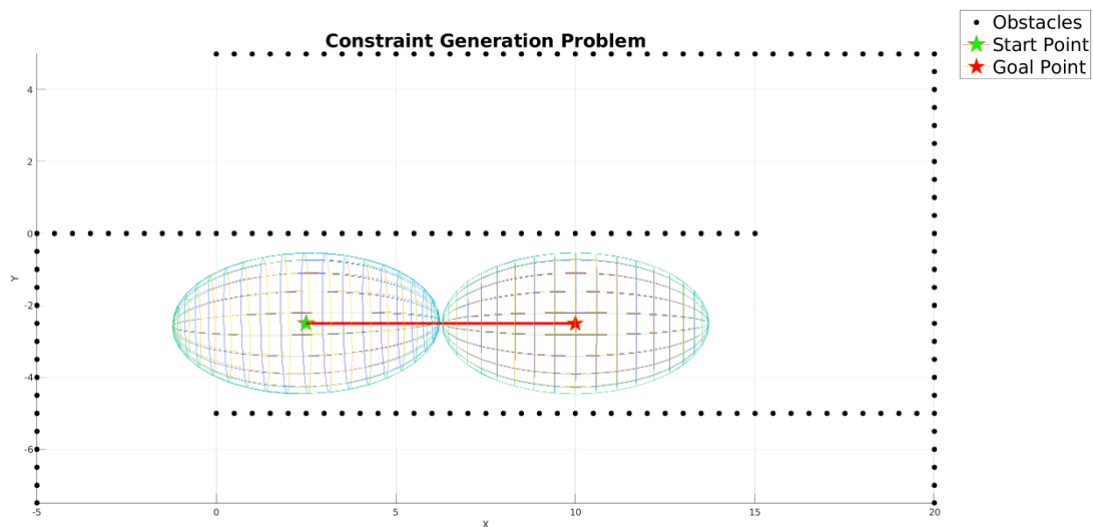


Figure 3.21: Ellipsoid-based collision avoidance (close-up): The figure shows two ellipsoids along the UGV trajectory.

trajectories were achieved by integrating topographical nuances via a novel attitude planning framework (Section 3.4). This framework uses rotation-minimizing frames (RMFs) to compute the UGV’s orientation without reliance on external terrain maps, addressing the challenge of in-situ map uncertainties. Third, the generation of 3D orientations, often overlooked in favor of planar assumptions, was tackled by deriving the rotation matrix ${}^{\mathbb{I}}R(\xi)^{\mathbb{J}} \in SO(3)$ using RMFs, providing a full 3D representation of the UGV’s pose (roll, pitch, yaw) essential for off-road stability. Fourth, collision avoidance was ensured through two algorithms: a safety corridor generation method (Section 3.7.1) and a convex volume maximization approach using ellipsoids (Section 3.7.2), both guaranteeing obstacle-free navigation in complex environments.

The original contributions of this research lie in several novel developments. A key contribution is the use of RMFs for $SO(3)$ planning, introducing a terrain-aware attitude planner that avoids the twisting issues of the Frénet-Serret frame (Section 3.4). Within this framework, a novel double projection algorithm was proposed (Section 3.4.2), which outperforms Wang’s double reflection method by achieving a 20% reduction in computational time at high resolutions (Section 3.8.1), enhancing real-time applicability. Additionally, a fast safety corridor generation algorithm was developed, leveraging point cloud partitioning to dynamically adjust the trajectory, offering computational efficiency over optimization-based methods. The *op-tempo* tactical weighing function $\chi(\xi)$ (Section 3.5.1) introduces a novel velocity regulation mechanism, adapting speed based on terrain and obstacle proximity, a feature underrepresented in prior off-road UGV literature.

Numerical results provide key learnings. The *op-tempo* framework effectively regulates velocities tactically during elevation climbs or near shelters and increasing to v_{nominal} in safer regions (Section 3.8.2). RMF-based attitude planning produces $SO(3)$ matrices with near-zero torsion, ensuring smooth orientation transitions critical for controller stability (Section

3.8.1). The comparison of angular velocity interpolation methods revealed that Rodrigues offers the best balance of accuracy and computational efficiency, outperforming SLERP and making it ideal for real-time deployment (Section 3.8.3). Finally, both collision avoidance algorithms proved effective: the safety corridor method dynamically reroutes the trajectory around obstacles, while the ellipsoid approach encloses the trajectory in a safe convex volume, avoiding collisions even in noisy conditions (Section 3.8.4).

This chapter thus concludes the development and simulation-based validation of motion planning algorithms for off-road UGVs in GNSS-denied, hostile environments. Building on the waypoint generation algorithm from Chapter 2, which produces tactical waypoints from a given map, the trajectory planner developed here generates smooth, time-parameterized trajectories with op-tempo velocity regulation, terrain-aware orientation, and collision avoidance guarantees. However, real-time deployment of these algorithms requires a navigation stack to incrementally construct obstacle and topography maps using onboard sensors. The next two chapters will address this need: first, by elaborating on the development of a navigation stack leveraging real-world sensors to produce the required maps, and second, by deploying these algorithms on the Clearpath Husky platform for experimental testing and validation in real-world conditions.

Chapter 4

Navigation

Contributions

This chapter presents algorithms and experimental results of terrain segmentation and incremental environment mapping using exteroceptive and proprioceptive sensors. The key contributions are:

- A novel geometry based inverse ray tracing to classify terrain points from obstacles
- An inverse sensor model with Bayesian framework to produce maps in the inertial frame
- An iterative convex hull method to produce terrain topography as a set of point clouds
- Real time experimental tests and validation of the navigation module

4.1 Introduction

The preceding chapters of this dissertation have focused on the numerical development and evaluation of tactical motion planning algorithms for an unmanned ground vehicle (UGV) operating in hostile environments. Chapter 2 introduced global path planning algorithms, specifically the tactical A^* and LPA^* planners, which generate stealthy paths by coasting shelters, while Chapter 3 developed a spline-based trajectory planner with *op-tempo* velocity

regulation to ensure dynamic feasibility on uneven terrain. These algorithms were validated through simulations in a virtual environment generated using Blender, where the occupancy grid $\mathfrak{M}^{\mathcal{O}} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ and terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, 500] \times [0, 500], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$ were provided as a priori data. However, real-time deployment in uncharted, GNSS-denied environments presents a significant challenge: the absence of pre-existing maps necessitates incremental, online construction of the environment model, leading to the problem of navigation addressed in this chapter.

Navigation in such conditions requires the UGV to simultaneously construct an environment model and estimate its pose in real time, using only onboard sensors. The motion planning algorithms from Chapters 2 and 3 mandate depth information as a fundamental requirement to identify obstacles and terrain elevation, enabling the generation of paths and trajectories that avoid obstacles while coasting shelters for stealth. Beyond depth, qualitative information such as terrain semantics (*e.g.*, grass, sand) could enhance these algorithms by informing traction and slip models, though this chapter prioritizes the geometric abstraction necessary for immediate deployment. The environment must be represented as a computational abstraction in the global reference frame $\mathbb{I} \triangleq \{O; X, Y, Z\}$, specifically through the occupancy grid $\mathfrak{M}^{\mathcal{O}}$ for obstacle mapping and the terrain topography map $\mathfrak{M}^{\mathcal{T}}$ for elevation profiling. Additionally, accurate real-time vehicle pose estimation in $\mathbb{I}$ is essential, as errors in localization can propagate through the navigation pipeline, compromising the UGV's ability to reach the goal set $\mathcal{G}$.

This chapter, as the experimental counterpart to the numerical studies in Chapters 2 and 3, explores the sensing modalities and algorithms required to translate tactical motion planning into real-time applications. The primary objective is to construct an environment model that captures depth information for obstacle and terrain mapping, while also providing reliable odometry for localization. The chapter presents a review on environment sensing,

develops algorithms for vehicle pose estimation, point cloud segmentation, occupancy grid mapping, and terrain topography mapping, leveraging both geometric and probabilistic approaches. Experimental tests and validation studies are conducted to assess the accuracy of these algorithms in controlled and real-world settings, ensuring the environment model aligns with the needs of the motion planning pipeline. Subsequent sections delineate the sensor suite, theoretical foundations of the algorithms, and empirical results, establishing a robust navigation framework for tactical UGVs in uncharted, GNSS-denied environments.

4.2 Review on Sensing for Tactical Navigation in GNSS-Denied Environments

Successful navigation in GNSS-denied environments, particularly for tactical UGVs operating without *a priori* maps depends heavily on the sensing modality and sensors' reliability. This section reviews the environmental information necessary for these algorithms to be deployed in real-time, surveys the literature on sensing methods and their applications, and identifies the optimal sensor suite for this research.

The requirements of the navigation stack are threefold: (1) a binary occupancy map identifying free and occupied cells to avoid obstacles, (2) a terrain topography map to capture elevation changes for 3D path projection, and (3) obstacle classifications to distinguish between coasting obstacles (shelters) and avoidance obstacles (mines) in the dual-mode framework. Additionally, the $\kappa(\alpha)$ function necessitates the distance to the nearest obstacle α for stealth-inducing behavior, requiring real-time obstacle detection and localization. The trajectory planner, which builds on the global path, also requires terrain-aware data to ensure dynamic feasibility. *Op-tempo* velocity planning adjusts the UGV's speed based on

terrain topography and capable of being extended to terrain properties thus necessitating: (1) terrain elevation and slope to compute orientations in the future horizon, (2) terrain type (*e.g.*, grass, sand, concrete) to estimate traction and slip, and (3) vehicle-terrain interaction parameters (*e.g.*, sinkage, friction) to optimize velocity profiles.

Given the present research's requirement of building the map incrementally whilst relying on sensors on-board alone, the decision regarding the choice of sensors to meet the objectives and seamless integration into the existing framework becomes paramount. To this end, the following section presents a brief survey of the existing sensing modalities and their application within the off-road autonomous navigation framework.

The literature provides a broad spectrum of sensing methods for collecting environmental information relevant to motion planning and navigation, as surveyed in [71]. These methods can be categorized into terrain type classification, terrain topography mapping, terrain property estimation, and vehicle-terrain interaction parameter estimation, each employing various sensors and processing techniques. Terrain type classification identifies surface types (*e.g.*, grass, sand, asphalt), which informs traction and slip models for trajectory planning. Vibration sensors, such as accelerometers and piezoelectric sensors in smart tires, have been widely used for this purpose. For example, Dupont *et al.* in [72] employed a differential measurement unit (DMU) on a wheeled robot, using fast Fourier transform and a probabilistic neural network to classify surfaces like asphalt, gravel, and sand. Similarly, Khaleghian and Taheri in [73] used a triaxial accelerometer inside a tire, combined with an encoder, to measure vibrations and angular speed, employing fuzzy logic to distinguish between grass, soil, and concrete .

Cameras have also been extensively utilized for terrain classification, leveraging visual features to mimic human perception. Filitchkin and Byl in [74] used a camera with Speeded Up Robust Features (SURF) and a support vector machine (SVM) to classify terrain for

a legged robot, while Wellhausen *et al.* in [75] applied supervised semantic segmentation to identify asphalt, gravel, and grass using color features. Microphones offer an alternative by analyzing terrain-induced sounds, as demonstrated by Libby *et al.* in [76], who classified grass, pavement, and gravel using audio signals processed with SVMs. Multi-sensor approaches enhance classification accuracy; for instance, Reina *et al.* in [77] fused data from a stereo camera, wheel encoders, and an IMU on a Clearpath Husky A200 to classify gravel, dirt, and ploughed terrain, using features like color, geometry, and vehicle slip.

Terrain topography mapping, crucial for obstacle detection and elevation profiling, often employs cameras, lidars, and radars. Braun *et al.* in [78] used a rotating stereo camera to generate a point cloud, dividing the map into sectors to assess traversability based on inclination and height. Schilling *et al.* in [79] combined lidar point clouds with camera images, using geometric features and semantic segmentation to classify terrain as safe, risky, or obstacle. Terrain property estimation, such as moisture content or compactness, and vehicle-terrain interaction parameters, like slip and sinkage, are typically derived using estimators. Yang *et al.* in [80] used a sensor suite including GPS, wheel speed sensors, and a camera to estimate terrain parameters like cohesion and friction angle, though this relied on initial values and was not GNSS-denied. Sebastian and Ben-Tzvi in [81] estimated slip using proprioceptive sensors on a tracked robot, training an SVM to predict terrain type based on vehicle states.

The environmental information collected through these sensing methods serves multiple purposes in autonomous navigation, primarily enhancing path planning, trajectory planning, and vehicle control. Terrain type classification data informs path planning by adjusting cost functions based on traversability. For example, Brooks and Iagnemma in [82] used audio and camera data to classify terrain, enabling a planetary rover to select paths with lower slip risk. Similarly, Reina *et al.* in [77] leveraged terrain classification to adjust the Husky A200's path,

avoiding high-slip surfaces like ploughed terrain. Terrain topography maps are directly used to update occupancy grids for path planning, as seen in Chavez-Garcia and Aycard in [83], where lidar and radar data were fused to create a Bayesian occupancy grid map, identifying moving objects and obstacles for real-time navigation. In the context of this research, such information would directly feed into the op-tempo velocity planner, adjusting speeds based on terrain slope and type to ensure dynamic feasibility while maintaining stealth.

Given the GNSS-denied constraint and the absence of a priori maps, the sensor suite for this research must enable real-time localization, mapping, and environmental perception using onboard sensors. The literature highlights the effectiveness of multi-modal sensing for robust navigation in such scenarios, as seen in works like Schilling *et al.* in [79] and Reina *et al.* in [77]. Particularly, the following sensors - Intel Realsense D435i, Intel Realsense T265, and Stereolabs Zed2i camera offer several advantages combining image, depth as well as inertial measurements. The Intel D435i, for instance, provides high-resolution point clouds for obstacle detection and terrain topography mapping, essential for updating the occupancy grid $\mathcal{M}^o$ and terrain topography map $\mathcal{M}^T$ that is needed by the path planner. The ZED 2i also provides this capability with long range stereo vision, whilst simultaneously supporting VSLAM and semantic segmentation for terrain classification, as demonstrated by Wellhausen *et al.* in [75]. The T265 camera on the other hand, provides VSLAM-based odometry albeit with reduced performance in extreme lighting conditions and absence of features. The Clearpath Husky's encoders and IMU on the other hand enable sensor fusion for precise pose estimation, as employed by Reina *et al.* in [77], ensuring accurate state estimation despite the lack of GPS. Its worth noting that, wheareas the literature suggests additional sensors like microphones or smart tires for terrain property estimation [73, 76], these are less critical for the immediate navigation tasks, given the focus on mapping and localization. The current suite is thus optimal, balancing computational efficiency with the comprehensive

data requirements of tactical navigation in uncharted, GNSS-denied environments, though future enhancements could include lidar for denser point clouds in highly cluttered terrains.

4.3 Sensors for tactical navigation

This research employs three sensors: two proprioceptive sensors and one exteroceptive sensor. The Intel RealSense D435*i* camera, an exteroceptive sensor, gathers information about the environment surrounding the UGV. Meanwhile, the two proprioceptive sensors, namely the Intel RealSense T265 camera and the MicroStrain 3DM-GX3-25 inertial measurement unit (IMU), serve as inertial sensors. Together, these three sensors facilitate simultaneous localization and mapping (SLAM). It is noteworthy that, given the primary objective of achieving goal attainment through concealed maneuvers, this mission, unlike typical exploration missions, does not aim to construct a comprehensive map of the environment. Nonetheless, the framework can be readily extended to exploration or reconnaissance missions.

IntelRealsense D435*i* Camera: The D435*i* camera (Figure 4.1) is primarily utilized for acquiring depth information. It is suitable for both indoor and outdoor applications, offering a visibility range from 0.3 to 3 m. With a field of view of $87^\circ \times 58^\circ$, it can produce data at a speed of 90 fps. The minimum observable depth is approximately 0.28 m, making it an excellent choice since the camera on the test UGV platform is positioned about 0.20 m above the ground. Additionally, the camera provides RGB information, albeit at a lower frame rate of 30 fps.



Figure 4.1: The Intel RealSense D435i camera produces color and depth information of surfaces in its field of view in the form of point clouds

Intel RealSense T265 Camera: The Intel RealSense T265 camera (Figure. 4.2) is a compact, off-the-shelf sensor designed for Visual-Inertial Odometry (VIO), providing pose and inertial information critical for localization in GNSS-denied environments. The T265 features two fisheye monochrome cameras with a wide field of view (FOV) of approximately 165° , capturing images at 30 fps with a resolution of 848×800 pixels, aligning with the frame rate of the D435i camera for synchronized data acquisition. It also includes a Bosch BMI055 IMU, delivering 200 Hz gyroscope and 62.5 Hz accelerometer data, enabling high-frequency inertial measurements. The sensor's processing is handled by an integrated Intel Movidius Myriad 2 Vision Processing Unit (VPU), a low-power application-specific integrated circuit (ASIC) optimized for computer vision tasks, consuming less than 1.5 W [84]. The Myriad 2 VPU runs the entire VIO algorithm onboard, offloading computation from the host system and ensuring low CPU utilization, which is advantageous for resource-constrained platforms like the Clearpath Husky.

The T265 employs a VIO algorithm to fuse visual and inertial data, producing 6-degree-of-freedom (6DoF) pose estimates—position (X, Y, Z) and orientation (pitch, yaw, roll)—at 200 Hz. The algorithm begins with feature detection in the stereo fisheye images, identifying and tracking visual landmarks (*e.g.*, corners) using techniques akin to Harris Corner or ORB features [85]. These features are mapped to 3D space via stereo triangulation, providing a sparse map of the environment. The IMU data complements this by offering high-frequency

motion estimates, mitigating the visual odometry’s sensitivity to feature scarcity (*e.g.*, near featureless surfaces like white walls) and addressing scale ambiguity inherent in monocular systems. The VIO algorithm integrates these data streams using a tightly coupled approach, likely employing a Kalman filter or optimization-based method to minimize drift and ensure low-latency tracking [85]. This enables the T265 to output robust pose and inertial information, which this research utilizes for real-time localization in the global frame $\mathbb{I}$, with performance enhanced by integrating external odometry from the Husky’s encoders, as detailed in Section 4.4.1.



Figure 4.2: Intel RealSense T265 tracking camera provides 6DoF pose and inertial information through visual-inertial odometry (VIO), enabling robust localization in GNSS-denied environments.

The following sections elaborate on the modules designed to process the information obtained from the sensors. This process involves determining the location of the UGV with respect to the start point in a globally fixed reference frame, classifying or segmenting the point cloud data produced by the D435i camera into ground points and positive obstacles. Subsequently, two types of maps are constructed: an obstacle map, more commonly referred to as an *occupancy grid*, and a terrain topography map. It is important to note that in this thesis, the terms occupancy grid, voxel map, and obstacle map refer to the same type of map.

4.4 Vehicle pose estimation

In this research, the UGV operates in a GNSS-denied environment, relying solely on on-board sensors to determine its pose relative to the starting point. The pose, $\mathcal{X} \in SE(3)$, encapsulates both the position and orientation of the UGV. Accurate pose estimation is critical as inaccuracies can proliferate throughout the navigation and mapping processes, affecting the UGV's ability to reach the intended goal set $\mathcal{G}$. The Intel Realsense T265 camera, providing pose data via visual odometry accessible through Intel's Software Development Kit, has shown limitations. Specifically, experiments have revealed that surface unevenness, absence of features and lighting conditions leads to significant pose drift over time. This drift, while negligible at low speeds and when traversing on smooth surfaces such as tiles or granite, and more importantly indoors, the drift becomes problematic under the more demanding conditions such as outdoors and more so in rough terrains with high irregularities, which is critical requirement of this research.

4.4.1 Sensor Fusion for Pose Estimation

To mitigate pose drift and enhance accuracy in GNSS-denied environments, a robust sensor fusion strategy is employed, integrating data from three onboard sensors: the Intel RealSense T265 camera, the MicroStrain 3DM-GX3-25 IMU on the Clearpath Husky, and the Husky's wheel encoders. The T265 camera provides 6DoF pose estimates (position and orientation) through Visual-Inertial Odometry (VIO) at 30 Hz, as detailed in Section 4.2, while the MicroStrain 3DM-GX3-25 IMU delivers high-frequency inertial measurements (accelerations and angular rates) at 100 Hz. The wheel encoders supply odometry data by measuring wheel rotations, translated into linear and angular velocities at 50 Hz. These complementary data sources are fused using an Extended Kalman Filter (EKF), implemented

via the ROS `robot_localization` package [86], to produce a more accurate and drift-resistant estimate of the UGV's pose in the global reference frame $\mathbb{I} \triangleq \{O; X, Y, Z\}$.

The EKF maintains a state vector $\mathbf{x}_k = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T \in \mathbb{R}^{12}$ at time step k , representing the UGV's position (x, y, z) , orientation (roll ϕ , pitch θ , yaw ψ), and their respective velocities in $\mathbb{I}$. The state prediction model, accounting for IMU accelerations $\mathbf{a} = [a_x, a_y, a_z]^T$ and angular rates $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$, is given by:

$$\mathbf{x}_{k|k-1} = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \quad (4.1)$$

where $\mathbf{u}_k = [\mathbf{a}, \boldsymbol{\omega}]^T$ is the control input, $\mathbf{f}(\cdot)$ is the nonlinear state transition function (approximated via IMU integration), and $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$ is the process noise with covariance $\mathbf{Q}_k$. The measurement model incorporates observations from the sensors is expressed as:

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, \quad (4.2)$$

where $\mathbf{h}(\cdot)$ maps the state to the measurement space, and $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$ is the measurement noise with covariance $\mathbf{R}_k$. The EKF updates the state estimate by minimizing the error between predicted and observed measurements, using the Kalman gain to weigh the contributions of each sensor based on their noise characteristics.

The ROS EKF package is configured via a YAML file, where the measurement covariances $\mathbf{R}_k$ for the T265 pose, IMU, and encoder data are tuned to reduce pose drift, as illustrated in Figure 4.3, where the EKF-fused path (blue) closely follows the ground truth, compared to the T265's dead reckoning (red), which exhibits noticeable drift over time.

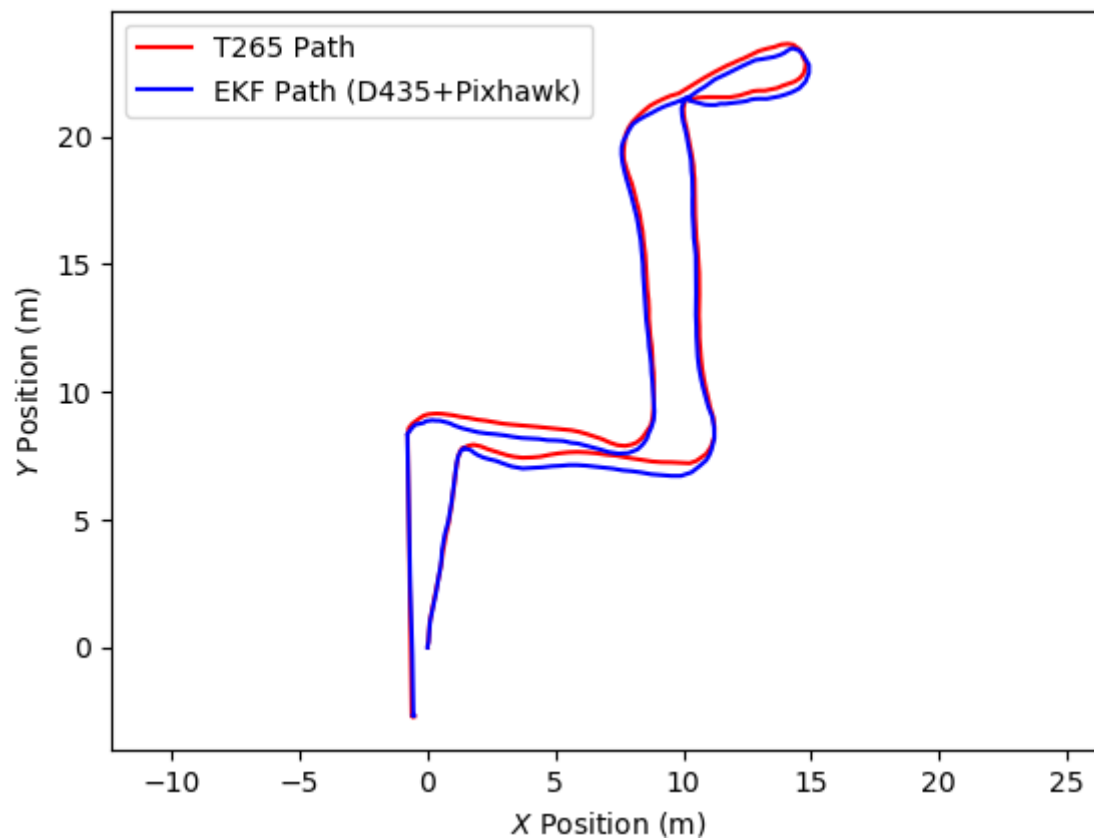


Figure 4.3: Comparison of UGV path using sensor fusion versus dead reckoning: The red line represents the path estimated by the T265 camera alone, while the blue line corresponds to the path obtained through Extended Kalman Filter (EKF) fusion of T265 pose, MicroStrain 3DM-GX3-25 IMU, and wheel encoder data.



Figure 4.4: RGB view of a test terrain with artificial obstacles: The image shows a controlled experimental setup featuring a terrain with visible tracks, simulating real-world conditions for testing terrain segmentation and obstacle detection systems

4.5 Point cloud segmentation

The point clouds produced by the D435i camera contain reflections from both positive obstacles and the traversable surface. In the context of UAVs, it is generally safe to assume that any portion of the point cloud represents an obstacle posing a collision risk. However, for UGVs, it is necessary to segment the dense point cloud into two distinct sets: ground points and obstacles. This segmentation is illustrated in the following figures: Whereas terrain features are readily apparent to the human eye in Figure 4.4, it is evident from Figure 4.5 that distinguishing between terrain and obstacles is crucial for accurately modeling the environment. It is noteworthy, that the efficiency of motion planning and trajectory tracking algorithms in off-road scenarios depends not only on the geometric interpretation of the terrain but also on its physical properties. Given that the overarching research goal is to develop terrain-aware tactical motion planning algorithms, the geometric aspect of the terrain, specifically its topography, is initially considered as the foundation of “terrain

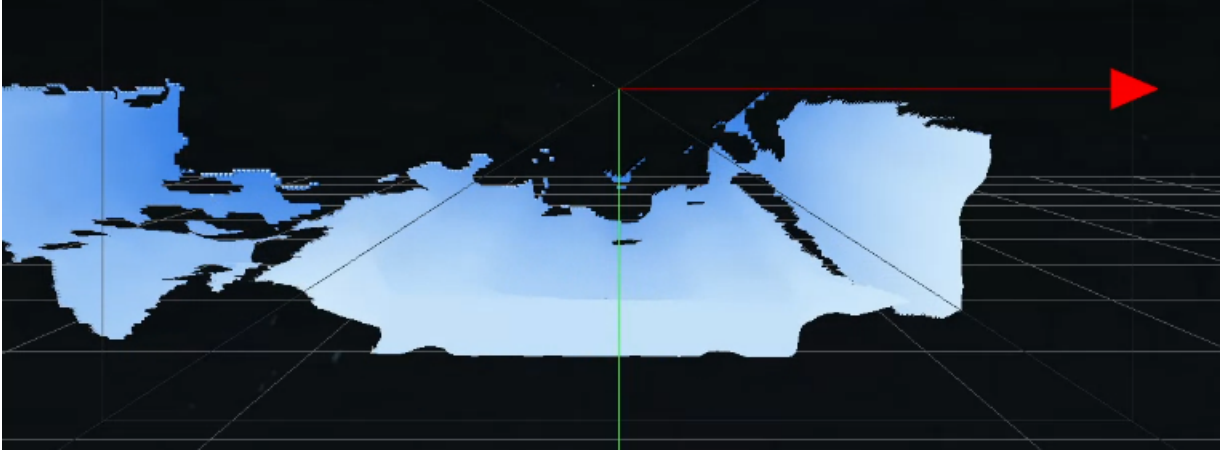


Figure 4.5: Point cloud representation of terrain: A 3D visualization of the terrain captured by the UGV’s sensors, showing the distribution of surface points used for ground segmentation and topography mapping

awareness.” Consequently, the question then arises: given the depth information comprising of both terrain and positive obstacles, how could the point cloud representing the terrain be extracted and to that end, which algorithm should be used that not only extracts the terrain with sufficient fidelity but at lower computational costs? The subsequent sections formally define the problem of terrain segmentation and delineates the proposed solution approach for the purposes of the present research on terrain aware tactical motion planning.

4.5.1 Problem Formulation for Point Cloud Segmentation

The point cloud segmentation is crucial for constructing the occupancy grid $\mathfrak{M}^{\mathcal{O}} : \mathbb{Z}_n \times \mathbb{Z}_m \rightarrow \{0, 1\}$ and terrain topography map $\mathfrak{M}^{\mathcal{T}} \triangleq \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid (x_i, y_i) \in [0, 500] \times [0, 500], z_i = z(x_i, y_i), i \in \{1, \dots, N\}\}$, which the tactical A^* and LPA^* algorithms use to generate stealthy paths toward the goal set $\mathcal{G}$. The segmentation problem thus involves partitioning the point cloud into terrain and obstacle sets, ensuring that the UGV can accurately model the environment in the global reference frame $\mathbb{I} \triangleq \{O; X, Y, Z\}$ for real-time navigation in GNSS-denied, uncharted terrains.

Let $\mathbb{P} \triangleq \{p'; X^{\text{cam}}, Y^{\text{cam}}, Z^{\text{cam}}\}$ denote an orthonormal reference frame centered at the camera's optical center p' , with X^{cam} , Y^{cam} , and Z^{cam} aligned with the camera's axes. The point cloud captured by the D435i at any time instant is defined as the set of points in this frame:

$$\mathcal{P} \triangleq \{(x, y, z) \mid (x, y, z) \in \mathbb{R}^3, (x, y, z) \text{ is a point captured by the D435i in } \mathbb{P}\}. \quad (4.3)$$

Let $\mathcal{O} \subset \mathcal{P}$ represent the set of *positive obstacles*, comprising non-traversable objects (*e.g.*, walls, rocks) that pose a collision risk to the UGV. In contrast, *negative obstacles* refer to regions (*e.g.*, swamps, pits) that may render the UGV immobile due to topography or texture, but are not considered in this research. In addition to positive obstacles, the dense point cloud $\mathcal{P}$ is seen to comprise the traversable surface points as well. The set of *terrain points* $\mathcal{T} \subset \mathcal{P}$, representing the traversable ground surface, is thus:

$$\mathcal{T} \triangleq \mathcal{P} \setminus \mathcal{O}. \quad (4.4)$$

The segmentation problem is formally stated as follows: given the point cloud $\mathcal{P}$, develop an algorithm to partition it into the disjoint sets $\mathcal{T}$ and $\mathcal{O}$, such that $\mathcal{T}$ contains points on the traversable terrain and $\mathcal{O}$ contains points on non-traversable obstacles, enabling accurate environment modeling for tactical motion planning.

To achieve the point cloud segmentation, several approaches have been studied in the literature. A comprehensive review of the terrain detection and segmentation techniques is delineated in [12]. It is observed that, geometric approaches have a proven efficiency over other approaches compared to the learning methods. In particular, the random sample consensus (RANSAC) algorithm [87, 88] is popularly used when the ground is relatively flat and hence used in this research. However, it is also pointed out in [12] that the use of RANSAC

in relatively uneven terrain may not be the most efficient method to extract terrain features. Based on this analysis, in the present research, a novel terrain topography extraction method is proposed as an additional layer to the RANSAC-based terrain segmentation.

4.5.2 Segmentation using the random sample consensus (RANSAC) algorithm

The RANSAC algorithm is a technique of recursive plane fitting to determine a set of inliers or outliers from a dataset. Given the input point cloud, $\mathcal{P}$, the RANSAC algorithm initially randomly selects a minimal subset of points, $\tilde{\mathcal{P}}$, to fit a potential ground plane, $\mathbf{P}$. Typically, three non-collinear points are required to define a plane in $\mathbb{R}^3$. Upon defining such a plane, the algorithm assesses its validity by determining the count of inliers, $|\mathcal{I}|$ s.t. $\mathcal{I} \in \mathcal{P}$, which are essentially, points from the input point cloud that fit the hypothesized model within a predefined tolerance, ϵ and a maximum cone angle with respect to the axis normal to the ground plane, ϵ_ϕ . This tolerance essentially accommodates variations in terrain roughness and measurement noise.

For each iteration, RANSAC refines its model by recalculating the plane based on new random samples and improving the inlier count to ensure the robustness of the detected ground surface. After a predetermined number of iterations, n_{iter} , the plane that exhibits the maximum alignment with the points (*i.e.*, the plane with the highest inlier count, $|\mathcal{I}|_{max}$) is selected as the most representative model of the ground. Upon establishing the ground plane, the segmentation process classifies the points that conform to this model as the ground set, $\mathcal{T}$, effectively segregating them from the remainder of the point cloud, designated as the obstacle set, $\mathcal{O}$. The procedure is summarized in Algorithm 8.

The RANSAC based ground segmentation was tested in real-time using the Intel D435i

camera. The yellow dots in Figure 4.6 represent the obstacles formed by a cardboard box whereas the green dots represent the segmented ground points.



Figure 4.6: Segmented point clouds in the camera frame. The yellow dots correspond to the positive obstacle set whereas the green dots capture the terrain topography

Algorithm 8 RANSAC based point cloud segmentation

Input: Point cloud $\mathcal{P}$, cone angle threshold ϵ_ϕ , distance threshold ϵ

Output: Terrain set $\mathcal{T}$ & obstacle set $\mathcal{O}$

```

1: function RANSACSEGMENT( $\mathcal{P}$ ,  $\epsilon_\phi$ ,  $\epsilon$ )
2:    $\mathcal{T} \leftarrow \emptyset$ ,  $\mathcal{O} \leftarrow \emptyset$ ,  $\mathcal{I}_{\max} \leftarrow \text{null}$ ,  $n_{iter}$  ▷ Initialize variables
3:   while  $k < n_{iter}$  do
4:      $\tilde{\mathcal{S}} \leftarrow$  Random subset of points of  $\mathcal{P}$ 
5:      $\mathbf{P} \leftarrow$  Fit plane to  $\tilde{\mathcal{S}}$ 
6:      $\mathcal{I} \leftarrow$  points in  $\tilde{\mathcal{S}}$  s.t.  $\|\mathbf{P} - \tilde{\mathcal{S}}\| \leq \epsilon$  &  $\cos^{-1}(\langle \hat{n}, Z' \rangle) \leq \epsilon_\phi$ 
7:     if  $|\mathcal{I}| > |\mathcal{I}_{\max}|$  then
8:        $\mathcal{I}_{\max} \leftarrow \mathcal{I}$ 
9:     end if
10:  end while
11:   $\mathcal{T} \leftarrow \mathcal{I}_{\max}$ 
12:   $\mathcal{O} \leftarrow \mathcal{P} \setminus \mathcal{T}$ 
13: end function

```

4.5.3 Segmentation Using Inverse Ray Tracing (IRT)

The inverse ray tracing (IRT) algorithm aims to derive the terrain topography based on the observation perspective of the point cloud $\mathcal{P}$, as defined in Section 4.5.1. A foundational assumption of the algorithm is that points below the terrain level, unless induced by sensor noise as “phantom” points, are impractical for consideration in the computational model, as they typically represent erroneous reflections or occlusions not physically relevant to the traversable surface. To enhance the visibility of terrain features, a virtual plane, denoted as $\mathcal{V}_p$, is established parallel to the camera’s horizontal plane but at a user-defined distance d below the camera’s optical center in the camera reference frame $\mathbb{P} \triangleq \{p'; X^{\text{cam}}, Y^{\text{cam}}, Z^{\text{cam}}\}$. Specifically, $\mathcal{V}_p$ is defined as the set of points satisfying the plane equation in $\mathbb{P}$:

$$\mathcal{V}_p \triangleq \{(x, y, z) \in \mathbb{R}^3 \mid z = -d, (x, y, z) \in \mathbb{P}\}, \quad (4.5)$$

where Z^{cam} is the optical axis (typically pointing forward), X^{cam} and Y^{cam} span the horizontal plane of the camera, and $d > 0$ ensures the plane lies below the camera’s origin p' (at $z = 0$ in $\mathbb{P}$). This configuration ensures that when the point cloud $\mathcal{P}$ is viewed from $\mathcal{V}_p$, only terrain surfaces are visible, while obstacles located above this plane are obscured, adhering to the line-of-sight principle [89]. This effect is analogous to observing a dome with pentagonal features on its surface; from inside, one would perceive only the smooth overarching surface rather than the individual pentagonal structures, as demonstrated in Figure 4.7.

The challenge now is to extract points that lie closest to this virtual plane. Given that the point cloud distribution is typically non-overlapping horizontally at any specific coordinate, a direct norm-based method to identify the lowest points would not suffice. Instead, the algorithm employs a ray-casting technique integrated with a nearest-neighbor search to meticulously select points proximal to $\mathcal{V}_p$. Specifically, points $\hat{\mathcal{P}}$ are sampled on the virtual



Figure 4.7: A dome with discernible hexagonal features on its surface, illustrating the line-of-sight principle used in inverse ray tracing for terrain segmentation.

plane $\mathcal{V}_p$, and rays are cast upwards from each sampled point in $\hat{\mathcal{P}}$ along the Z^{cam} axis. A nearest-neighbor algorithm with search radius ρ , analogous to a spherical bulb, is translated along each ray at incremental steps. If a point from $\mathcal{P}$ is found within this radius, the ray traversal stops, and the point is appended to $\mathcal{T}$. Once this initial collection of candidate terrain points is assembled from each ray's nearest encounter in $\mathcal{T}$, a secondary refinement step is executed. This step involves another round of nearest-neighbor searches, now applied across the entirety of $\mathcal{P}$ from each point in $\mathcal{T}$, to ensure that the most representative terrain points are retained. A schematic is shown in Figure 4.8, and the procedure is summarized in Algorithm 9.

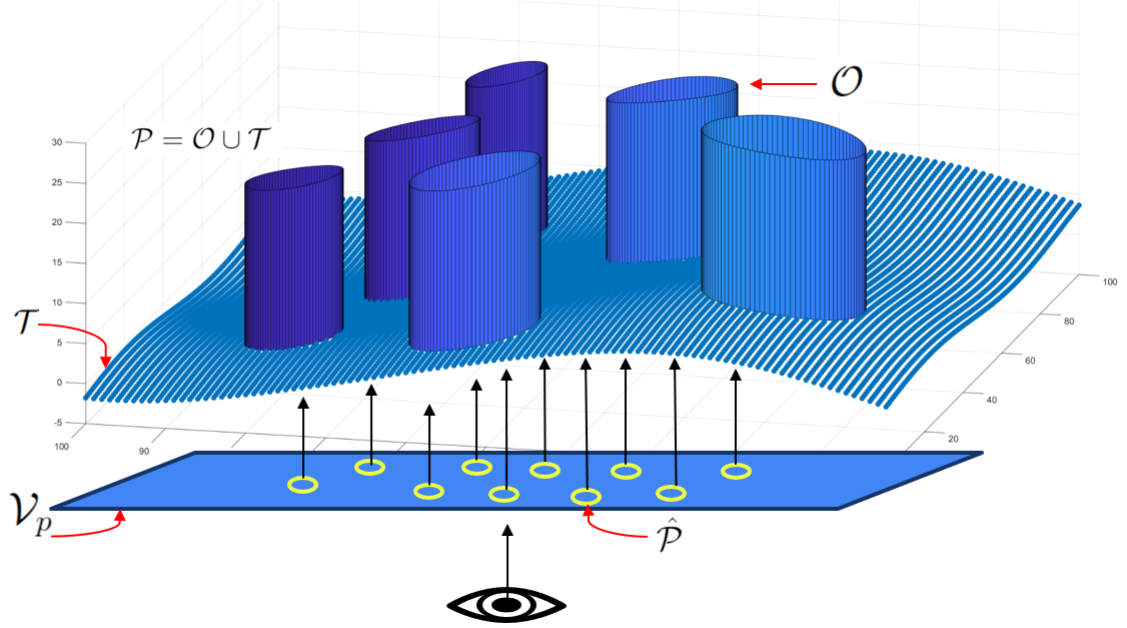


Figure 4.8: Illustration of the inverse ray tracing technique for terrain topography extraction: The diagram shows how virtual rays are cast from a plane $\mathcal{V}_p$ below the terrain to detect terrain points $\mathcal{T}$, while ignoring obstacles $\mathcal{O}$, enabling accurate terrain modeling.

Algorithm 9 Inverse Ray Tracing for Terrain Topography Extraction

Input: Point cloud $\mathcal{P}$, virtual plane distance d , search radius ρ

Output: Terrain set $\mathcal{T}$

```

1: function INVERSERAYTRACE( $\mathcal{P}, d, \rho$ )
2:   Define the virtual plane  $\mathcal{V}_p$  at distance  $d$  below the camera center as per Equation
   (4.5)
3:    $\mathcal{T} \leftarrow \emptyset$  ▷ Initialize terrain set
4:   for each sample point  $\hat{\mathcal{P}} \in \mathcal{V}_p$  do
5:     Cast a vertical ray  $R_s$  upwards from  $\hat{\mathcal{P}}$  along  $Z^{\text{cam}}$ 
6:      $found \leftarrow \text{false}$  ▷ Set flag to stop ray traversal
7:     while not  $found$  do
8:       Perform a spherical nearest-neighbor search within radius  $\rho$ 
9:       if a point  $p \in \mathcal{P}$  is found along  $R_s$  then
10:         $found \leftarrow \text{true}$ 
11:         $\mathcal{T} \leftarrow \mathcal{T} \cup \{p\}$ 
12:      end if
13:      Increment search depth along  $R_s$ 
14:    end while
15:  end for
16:  Perform a final nearest-neighbor search over  $\mathcal{T}$  to refine the set
17:  return  $\mathcal{T}$ 
18: end function

```

4.6 Voxel-Based 3D Mapping

The point clouds generated by the D435i camera, as delineated in Section 4.5, reside within the camera’s frame of reference, denoted as $\mathbb{P}$. In contrast, the planning algorithms generate reference paths or trajectories (detailed in Chapters 2 and 3) within a fixed global reference frame. It is pertinent to recall the gravity-aligned orthonormal reference frame, $\mathbb{I} \triangleq \{O; X, Y, Z\}$, previously elucidated in Section 4.4.1, wherein the UGV’s pose, $\mathcal{X} \in SE(3) \in \mathbb{I}$, was obtained using EKF-based sensor fusion. This section serves as the junction between pose estimation and point cloud segmentation, with the objective of synthesizing these algorithms to construct an environmental model within the global frame, thereby establishing the groundwork for the path planning and trajectory planning algorithms.

To effectively plan paths or trajectories directing the UGV from an initial position $r_0 \in \mathbb{I}$ to a target set $\mathcal{G} \in \mathbb{I}$, the segmented point cloud sets—namely, the terrain set $\mathcal{T} \in \mathbb{P}$ and the obstacle set $\mathcal{O} \in \mathbb{P}$ —must be transformed into the global reference frame $\mathbb{I}$ and registered. These point clouds represent a quantized set of points reflected from surfaces within the camera’s field of view, possessing an exceptionally high resolution on the order of millimeters. The process of *registration* refers to aligning and integrating the segmented sets $\mathcal{T}$ and $\mathcal{O}$ from $\mathbb{P}$ into $\mathbb{I}$, determining the spatial correspondence between points in the two frames and assigning specific regions within the Euclidean space $\mathbb{I} \subset \mathbb{R}^3$ to either obstacles or terrain.

A critical question arises: how should the world in $\mathbb{I}$ be represented computationally to enable efficient registration? In computational frameworks, the environment is typically discretized to manage complexity and memory usage. One option is to represent the world as a collated version of the point clouds, preserving the original high-density point data. However, such a representation, with millimeter-scale resolution, would lead to significant

memory overhead and is impractical for large environments, risking memory leaks and computational inefficiency. Alternatively, voxel-based representations discretize the space into a 3D grid of cubic bins, called voxels, offering a memory-efficient abstraction by aggregating points within each voxel. Other methods, such as potential field representations, model the environment as a continuous scalar field where each point in space is assigned a potential value (*e.g.*, high potential near obstacles, low potential in free space), often used for path planning but less suited for direct point cloud registration due to their focus on gradient-based navigation rather than geometric mapping [90]. Given the need for a balance between accuracy and computational efficiency in this research, a voxel-based approach is adopted. A *voxel* is a discretized cubic bin in 3D space, defined by a fixed resolution (*e.g.*, 10 cm per side), and a *voxel map* is a 3D array of such voxels, where each voxel can store properties like occupancy or terrain height, as shown in Figures 4.11 and 4.9.

Formally, the mapping problem can be articulated as follows: let $\mathfrak{M}^{\mathcal{O}} \in \mathbb{I}$ represent the *occupancy grid* produced by the navigation system, and $\mathfrak{M}^{\mathcal{T}} \in \mathbb{I}$ represent the terrain topography map produced by the navigation system, both structured as voxel maps. These are illustrated in Figures 4.11 and 4.9, respectively. Let $\mathfrak{T}$ represent the actual terrain topography (ground truth) in $\mathbb{I}$, as shown in Figure 4.9, and $\mathfrak{O}$ represent the actual obstacle map (ground truth) surrounding the UGV in $\mathbb{I}$, as shown in Figure 4.10. Given the vehicle pose $\mathcal{X} \in SE(3) \in \mathbb{I}$ and the segmented point cloud sets $\mathcal{T} \in \mathbb{P}$ and $\mathcal{O} \in \mathbb{P}$, the objective of the mapping system is to produce a computationally efficient abstraction of the environment, expressed as $\mathfrak{M}^{\mathcal{O}} \in \mathbb{I}$ and $\mathfrak{M}^{\mathcal{T}} \in \mathbb{I}$, that are as close as possible to the actual obstacle map and terrain map, *i.e.*, $\mathfrak{O}$ and $\mathfrak{T}$, respectively. The problem statement can be encapsulated

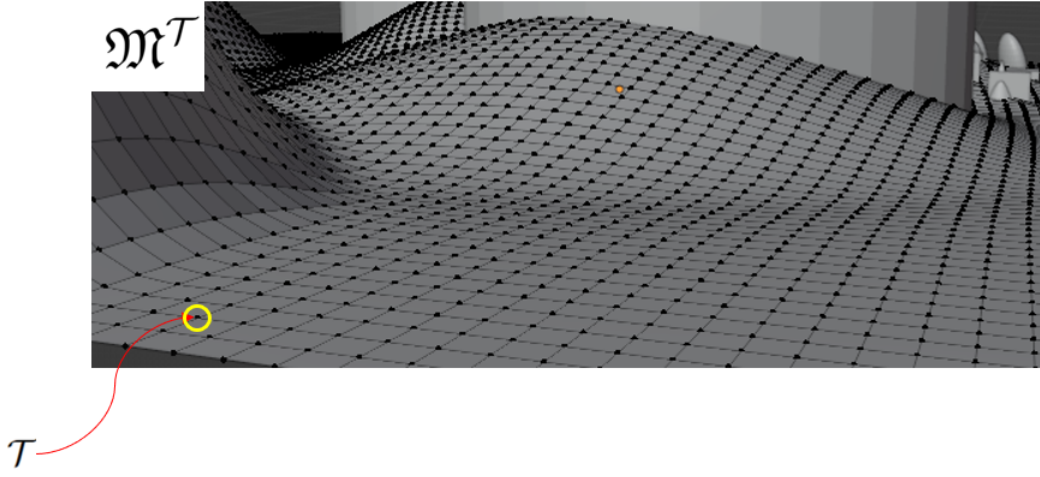


Figure 4.9: 3D representation of terrain topography: The mesh grid illustrates the terrain’s surface, highlighting key features such as elevation changes and curvatures. The point labeled with the yellow circle corresponds to the terrain set $\mathcal{T}$, which is registered in the global frame to form the terrain topography map $\mathfrak{M}^{\mathcal{T}}$.

in the following function signatures:

$$\mathfrak{M}^{\mathcal{O}} = f(\mathcal{X}, \mathcal{O}), \quad (4.6)$$

$$\mathfrak{M}^{\mathcal{T}} = f(\mathcal{X}, \mathcal{T}). \quad (4.7)$$

Having defined the environment representation as a voxel map, the next step is to develop a method for registering the point clouds into this structure. The following subsection introduces a Bayesian inverse sensor model approach, which leverages probabilistic techniques to update the voxel map based on sensor measurements, ensuring robust and accurate mapping for real-time navigation.

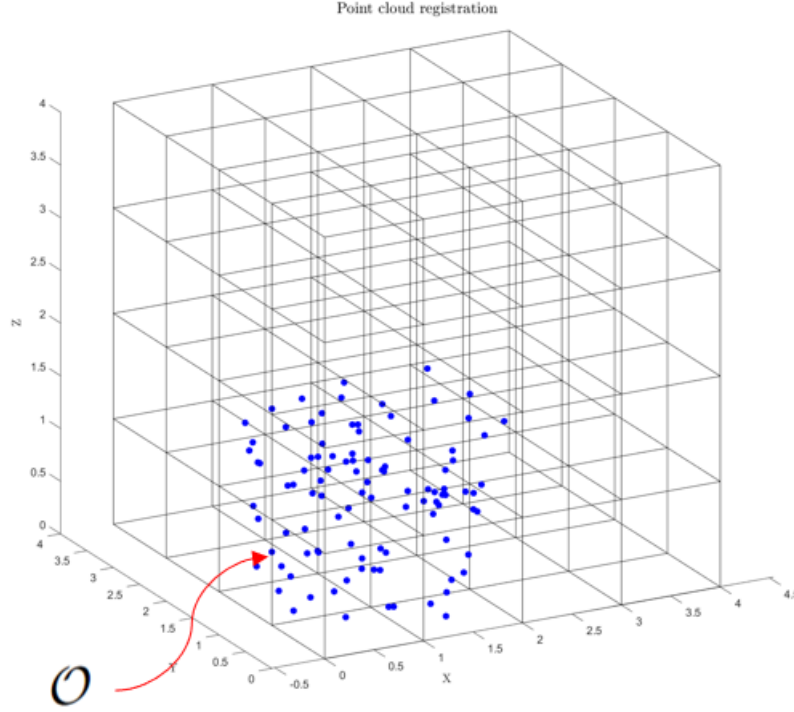


Figure 4.10: Point cloud registration: A 3D visualization demonstrating the process of aligning point cloud data in virtual discrete bins called voxels, with point clouds (blue) representing obstacle locations $\mathcal{O}$ within the mapped environment. The registration process is crucial for accurately modeling the terrain and obstacles in UGV navigation.

4.6.1 Bayesian Inverse Sensor Model for 3D Mapping

The voxel map is implemented as a hashmap, where each key represents integer voxel indices (i, j, k) in a 3D grid, and the value is an unsigned char (0 to 255) representing the voxel's occupancy probability in log-odds form. When a point cloud $\mathcal{P} \in \mathbb{I}$ (transformed from $\mathbb{P}$ using the UGV's pose $\mathcal{X} \in SE(3)$ from Section 4.4.1) is received, each point (x, y, z) is converted to voxel indices (i, j, k) by scaling with the map resolution: $i = \lfloor x \cdot \text{res}_x \rfloor$, $j = \lfloor y \cdot \text{res}_y \rfloor$, $k = \lfloor z \cdot \text{res}_z \rfloor$, where $\text{res}_x = \text{res}_y = \text{res}_z = 5$ voxels/m. The goal is to

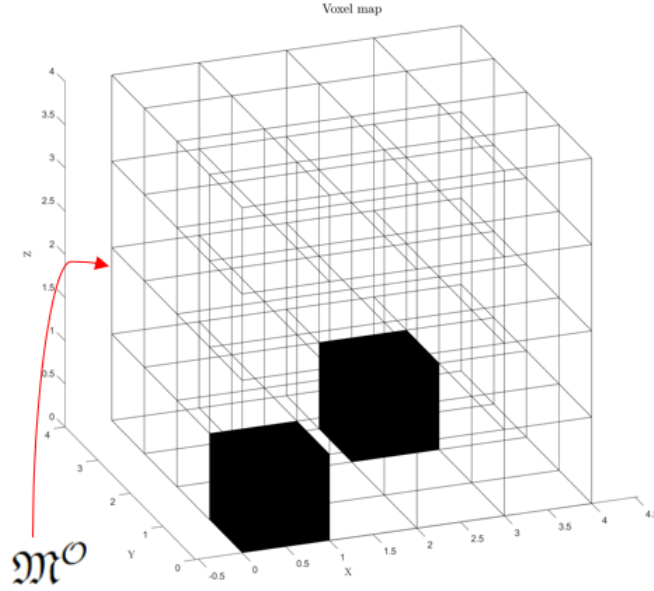


Figure 4.11: Voxel map representation of obstacle data $\mathfrak{M}^{\mathcal{O}}$: The 3D grid illustrates the discretization of space into voxels, with black cubes representing grid elements as occupied after registration of the obstacle set $\mathcal{O}$.

determine the likelihood that a voxel (i, j, k) is occupied, given that a point falls within its corresponding bin, addressing the question: given a sensor measurement, what is the probability that the voxel is occupied?

This probabilistic approach, known as the inverse sensor model, contrasts with the forward sensor model, which estimates the likelihood of a measurement given a known map state, *i.e.*, $p(z_t | m)$, where z_t is the measurement at time t and m is the map state. The forward model is computationally intensive for real-time applications due to the high-dimensional map [15]. Instead, the inverse sensor model estimates $p(m | z_t)$, making it more suitable for iterative updates. Bayes' theorem provides the foundation, expressed for a voxel m_{ijk} as:

$$p(m_{ijk} | z_{1:t}) = \frac{p(z_t | m_{ijk})p(m_{ijk} | z_{1:t-1})}{p(z_t)}, \quad (4.8)$$

where $p(m_{ijk} | z_{1:t})$ is the posterior probability, $p(z_t | m_{ijk})$ is the likelihood, $p(m_{ijk} | z_{1:t-1})$

is the prior, and $p(z_t)$ is the normalizing constant. Assuming a static world and time-independent measurements (Markov assumption), the likelihood simplifies to $p(z_t \mid m_{ijk})$, and $p(z_t)$ is independent of m_{ijk} [91].

The probability is converted to log-odds form to simplify the update process:

$$L(m_{ijk}) = \log \frac{p(m_{ijk})}{1 - p(m_{ijk})}, \quad (4.9)$$

where $p(m_{ijk})$ is the occupancy probability (1 for occupied, 0 for free, 0.5 for unknown). Applying this to Equation (4.8), the log-odds update becomes:

$$L(m_{ijk} \mid z_{1:t}) = L(m_{ijk} \mid z_t) + L(m_{ijk} \mid z_{1:t-1}) - L_0, \quad (4.10)$$

where $L(m_{ijk} \mid z_t)$ is the inverse sensor model contribution, $L(m_{ijk} \mid z_{1:t-1})$ is the prior, and $L_0 = 0$ (initial probability $p = 0.5$). Probabilities are mapped to unsigned char values, with 127 for unknown, 255 for occupied, and 0 for free.

The inverse sensor model $L(m_{ijk} \mid z_t)$ is computed based on whether the voxel is occupied or free. For an occupied voxel (a point in $\mathcal{O}$ or $\mathcal{T}$), the log-odds update uses a weight γ_{occupied} for occupied voxels and a smaller γ_{free} for free voxels, reflecting lower confidence [92]. These weights account for decreasing point density with distance [93]. The update rule is:

$$L_{\text{post}} = L_{\text{prior}} + \gamma \cdot L_{\text{ism}}, \quad (4.11)$$

where L_{ism} is the inverse sensor model contribution (assumed constant), and γ is γ_{occupied} or γ_{free} . The log-odds is clamped between 0 and 255.

To ensure confidence in free space, rays are cast from the UGV's voxel position $(i_{\text{UGV}}, j_{\text{UGV}}, k_{\text{UGV}})$ to each point's voxel position (i, j, k) , marking intermediate voxels as free using γ_{free} in

Equation (4.11). The ray's direction is determined by identifying the primary axis (largest displacement) and computing a unit vector, then tracing the ray voxel-by-voxel [92]. This process runs iteratively at 30 Hz with incoming point clouds, updating the map for visualization or planning (threshold at 180). Algorithm 10 summarizes the process.

Algorithm 10 Bayesian Inverse Sensor Model for 3D Voxel Mapping

Input: Point cloud $\mathcal{P} \in \mathbb{I}$, UGV pose $\mathcal{X} \in SE(3)$, map resolution $\text{res}_x, \text{res}_y, \text{res}_z$, grid dimensions N_x, N_y, N_z , voxel size s_v

Output: Updated voxel map $\mathfrak{M}$ as a hashmap

```

1: function UPDATEVOXELMAP3D( $\mathcal{P}, \mathcal{X}$ )
2:   Compute UGV voxel position  $(i_{\text{UGV}}, j_{\text{UGV}}, k_{\text{UGV}}) \leftarrow (\lfloor \mathcal{X}_x \cdot \text{res}_x \rfloor, \lfloor \mathcal{X}_y \cdot \text{res}_y \rfloor, \lfloor \mathcal{X}_z \cdot \text{res}_z \rfloor)$ 
3:   for each point  $(x, y, z) \in \mathcal{P} \in \mathbb{I}$  do
4:     Compute voxel indices  $(i, j, k) \leftarrow (\lfloor x \cdot \text{res}_x \rfloor, \lfloor y \cdot \text{res}_y \rfloor, \lfloor z \cdot \text{res}_z \rfloor)$ 
5:     Skip if  $(i, j, k)$  outside grid bounds  $[0, N_x) \times [0, N_y) \times [0, N_z)$ 
6:     Compute distance  $d \leftarrow \sqrt{(x - \mathcal{X}_x)^2 + (y - \mathcal{X}_y)^2 + (z - \mathcal{X}_z)^2}$ 
7:     Identify primary axis as  $\arg \max(|i - i_{\text{UGV}}|, |j - j_{\text{UGV}}|, |k - k_{\text{UGV}}|)$ 
8:     Compute unit vector  $(\delta_x, \delta_y, \delta_z)$  along primary axis
9:     Compute ray steps  $N_{\text{steps}} \leftarrow d / \sqrt{\delta_x^2 + \delta_y^2 + \delta_z^2}$ 
10:    for  $n \leftarrow 0$  to  $N_{\text{steps}}$  do ▷ Cast ray for free voxels
11:      Compute ray position  $(r_x, r_y, r_z) \leftarrow (i_{\text{UGV}} + \delta_x \cdot n, j_{\text{UGV}} + \delta_y \cdot n, k_{\text{UGV}} + \delta_z \cdot n)$ 
12:      Compute ray voxel indices  $(i_r, j_r, k_r) \leftarrow (\lfloor r_x / s_v \rfloor, \lfloor r_y / s_v \rfloor, \lfloor r_z / s_v \rfloor)$ 
13:      Skip if  $(i_r, j_r, k_r)$  outside grid bounds
14:      Update voxel  $(i_r, j_r, k_r)$  as free with weight  $\gamma_{\text{free}}$  using Equation (4.11)
15:    end for
16:    Update voxel  $(i, j, k)$  as occupied with weight  $\gamma_{\text{occupied}}$  using Equation (4.11)
17:  end for
18:  Publish updated voxel map for motion planning
19: end function

```

4.6.2 Experimental Tests for 3D Voxel Mapping

To test the 3D voxel mapping approach in real-time, an experiment was conducted in an indoor lab environment at the Advanced COnTrol Systems Lab, Virginia Tech, over an area measuring approximately 20 meters by 20 meters, as depicted in Figure 4.12. The lab features a cluttered setting with desks, boxes, and equipment, alongside open spaces, providing a

diverse testbed for navigation and mapping. The sensor suite, comprising the Intel RealSense D435i and T265 cameras along with the MicroStrain 3DM-GX3-25 IMU and wheel encoders on a Clearpath Husky robot, was deployed to construct the voxel maps $\mathfrak{M}$ using unsegmented point clouds. The map was configured with a voxel resolution of 5 cm (*i.e.*, each voxel represents a 10 cm cube) and dimensions of $100 \times 100 \times 30$ voxels, covering a physical space of $20 \times 20 \times 6$ meters. All communications and data processing were handled using ROS Noetic, with the mapping executed on a laptop equipped with an Intel i7-9750H CPU (2.60 GHz). The mapped data was visualized in RViz using the `sensor_msgs::PointCloud2` message format. The Clearpath Husky was operated at a speed of approximately 0.5 m/s, ensuring real-time mapping performance under dynamic conditions.

The results of the 3D voxel mapping are illustrated in Figure 4.12, where the top images show the lab environment, and the bottom images display the corresponding voxel maps visualized in RViz. The RViz plots use a color gradient to represent height, with blue indicating higher elevations (*e.g.*, obstacles) and green to yellow transitioning to lower elevations (floor). A qualitative correspondence is evident between the mapped data and the physical environment: the open spaces which comprise only floor, and by extension, correspond to lower elevations are accurately represented as predominantly yellow regions while cluttered areas with desks and boxes (*e.g.*, along the lab's perimeter) show elevated regions in green and blue, reflecting the presence of obstacles at varying heights. The map captures the general layout, including the distribution of obstacles and free space, though some finer details (*e.g.*, smaller objects or thin structures) are abstracted due to the 10 cm voxel resolution. This qualitative alignment demonstrates the effectiveness of the Bayesian inverse sensor model in constructing a 3D voxel map suitable for tactical navigation, providing a reliable representation for the path planning algorithms to generate stealthy paths.

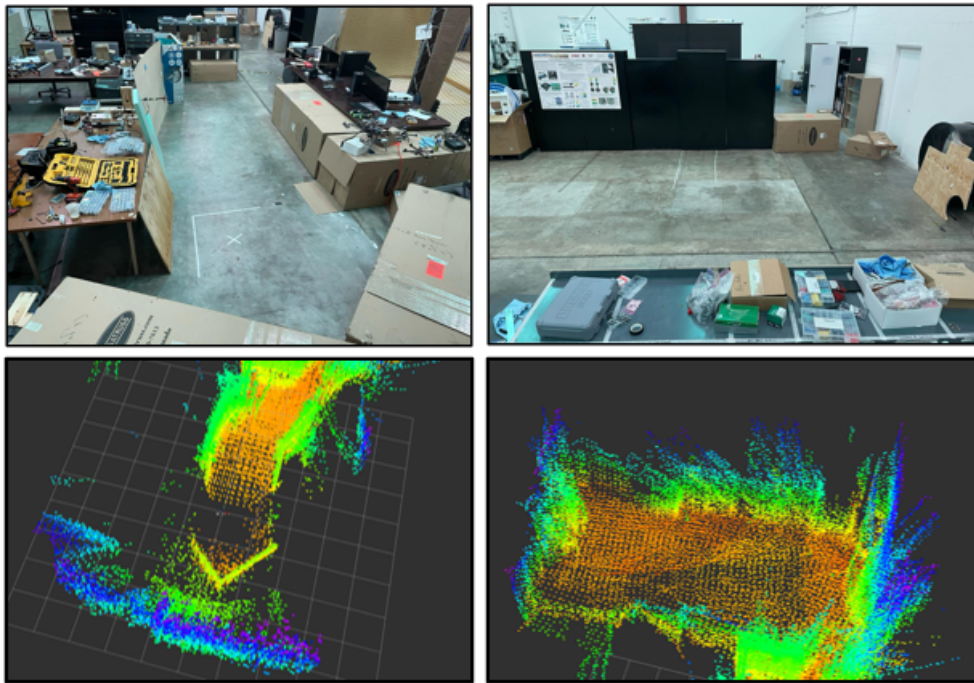


Figure 4.12: Experimental setup and results for 3D voxel mapping in an indoor lab environment. Top: Photographs of the lab ($20\text{ m} \times 20\text{ m}$) showing cluttered areas with desks and boxes (left) and open spaces (right). Bottom: Corresponding voxel maps visualized in RViz, with colors representing height (blue for higher elevations, green to yellow for lower elevations).

4.7 Occupancy grid mapping - experimental results and validation

The path planner delineated in Chapter 2 necessitates a 2D occupancy map. To this end, the segmented obstacle points are passed as an argument to the Bayesian inverse sensor model and then projected into a 2D grid to generate the required Mo. To demonstrate the effectiveness of this algorithm, the mapping was tested in a controlled environment. Unlike the test outlined in Section 4.6.2, a more quantifiable approach was required to validate the occupancy grid produced by the navigation system. To this end, the mapping algorithm was deployed in an indoor setting with tiled floors. The resolution of the voxel was deliberately chosen to match the dimensions of each tile, and the map dimension was adjusted to match the dimensions of the corridor to a reasonable accuracy. The results were then scrutinized by comparing the distance between the two walls in the actual building by counting the number of tiles, *i.e.*, 5 tiles. since each tile corresponds to one voxel, it was observed that the same region in the occupancy map produced by the path planner had 5 voxel, thus verifying the accuracy of the occupancy grid generation algorithm.

4.7.1 Experimental Validation of 2D Occupancy Grid Mapping

The tactical path planning algorithms delineated in Chapter 2, specifically the A^* and LPA^* planners, require a 2D occupancy grid map $\mathfrak{M}^o$ to generate stealthy paths that coast obstacles for covert navigation. To this end, the 3D voxel map constructed using the segmented obstacle point cloud and the Bayesian inverse sensor model (Section 4.6.1) is projected onto the XY plane in the global frame $\mathbb{I}$, aggregating occupancy probabilities along the Z -axis to produce a 2D grid. To validate the accuracy of this projection and

the underlying 3D mapping, a controlled experiment was conducted in an indoor hallway environment with tiled floors, as shown in Figure 4.13. Unlike the qualitative validation in Section 4.6.2, this test adopts a quantifiable approach to assess the fidelity of the occupancy grid by leveraging the ontological similarity between the tiled floor layout and the discretized grid structure.

The experiment was performed in a hallway measuring approximately 10 meters by 3 meters, with a tiled floor where each tile spans 60 cm by 60 cm, as depicted in Figure 4.13. The 3D voxel map was deliberately configured with a resolution of 60 cm per voxel to match the dimensions of each tile. The trajectory was designed to navigate a predetermined number of tiles along the vehicle's roll axis, followed by a 90° turn. The resulting 2D occupancy grid, as shown in Figure 4.13, was scrutinized by comparing the distance between the hallway walls. In the physical environment, the distance spanned 5 tiles (3 meters), systematically quantified using the tiles as real-world indices. Correspondingly, the mapped grid accurately reflected this distance as 5 voxels, confirming the precision of the occupancy grid generation algorithm in capturing spatial relationships for tactical path planning.

4.8 Terrain Topography Mapping

The topography map essentially is a representation of the geometric attribute of the terrain's shape. Indeed, a terrain manifests more often as highly irregular surfaces with undulations, slopes, irregularities *etc.*, as in the case of forests or natural terrains among many. Terrains could also have a well-defined and smooth structure, such as asphalt roads or grasslands with unevenness on a global scale but locally flat. Either way, given that one of the challenges addressed by this research is for the UGV to navigate autonomously in highly uneven terrain, the map that encapsulates the terrain shape stands paramount.



Figure 4.13: Validation of 2D occupancy grid mapping in an indoor hallway environment. Top: Photograph of the hallway with tiled floors (each tile $60\text{ cm} \times 60\text{ cm}$), showing a distance of 5 tiles (3 m) between walls

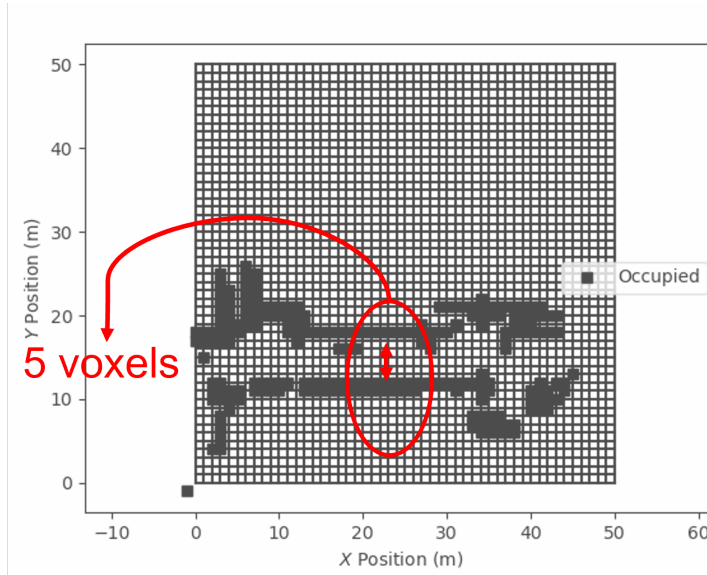


Figure 4.14: Indoor test environment with tiled floor and static obstacles: The image shows a hallway setup used for validating obstacle detection and occupancy grid mapping, with tile dimensions marked to evaluate occupancy map accuracy

The question addressed in this section is essentially, given the segmented ground points $\mathcal{T}$, as discussed in Section 4.5.1, how can a global representation of the terrain topography be developed that could facilitate the planning algorithms?

The terrain topography map $\mathcal{M}^{\mathcal{T}}$ can indeed be represented as a voxel map, and it is reasonable to set a higher resolution to circumvent further processing, such as Delaunay triangulation, which would be required to reconstruct a continuous surface from sparse points [?]. A reasonably high resolution (*e.g.*, 5 cm per voxel) can directly capture the terrain's geometric features, enabling the reuse of the voxel mapping approach outlined in Section 4.6.1. However, due to the computational complexity and memory demands of maintaining a high-resolution voxel map over large areas, it may be preferable to conserve the point clouds directly by downsampling them to a desired cardinality. Furthermore, this predisposition to defining fixed voxel resolutions could lead to cropped regions in the topography under circumstances where the terrain's elevations exceed those predetermined dimensions.

Moreover, as the ultimate objective of the topography modeling algorithm is to represent the terrain map, $\mathfrak{M}^{\mathcal{T}}$, as a collection of spatial points in $\mathbb{R}^3$ that comprehensively span the terrain, a novel algorithm is proposed. This innovative approach eschews any form of artificial discretization and processes the point clouds directly, thereby generating the desired topography model in real-time.

4.8.1 Iterative convex hull-based terrain topography mapping

The following section expounds on the terrain modeling algorithm using iterative convex hulls. As in the previous section, $\mathfrak{T}$ represents the set of spatial coordinates in $\mathbb{R}^3$ that correspond to the actual terrain (ground truth), $\mathcal{T} \in \mathbb{P}$ represents the segmented ground points in the Intel RealSense D435i camera's reference frame and,

$$\mathfrak{M}^{\mathcal{T}} \triangleq \{x, y, z | x, y, z \in \mathbb{R} \subset \mathfrak{T} \in \mathbb{I}\}, \quad (4.12)$$

represents the terrain topography map produced by the navigation system. The set, $\mathcal{T} \in \mathbb{P}$ is initially transformed to the global frame, $\mathbb{I}$. The first set of terrain point set in the global frame, ${}^{\mathbb{I}}\mathcal{T}$ is appended to the map, $\mathfrak{M}^{\mathcal{T}}$. A convex hull,

$$C_h \triangleq \text{conv}({}^{\mathbb{I}}\mathcal{T}) \quad (4.13)$$

is then constructed around the set ${}^{\mathbb{I}}\mathcal{T}$, where $\text{conv}(\cdot)$ represents a convex hull. Once the new ground set, ${}^{\mathbb{I}}\hat{\mathcal{T}}$ is obtained, another convex hull,

$$\hat{C}_h \triangleq \text{conv}({}^{\mathbb{I}}\hat{\mathcal{T}}) \quad (4.14)$$

is constructed. The elements of ${}^{\mathbb{I}}\hat{\mathcal{T}}$ that do not lie in the intersection of the two convex hulls

$$\tilde{\mathcal{T}} = \{x, y, z | x, y, z \in {}^{\mathbb{I}}\hat{\mathcal{T}} \setminus (C_h \cap \hat{C}_h)\}, \quad (4.15)$$

are then appended to $\mathfrak{M}^{\mathcal{T}}$. The process is repeated for every incoming point cloud set that is transformed to the global reference frame, ${}^{\mathbb{I}}\hat{\mathcal{T}}$. The point cloud-based terrain mapping algorithm is summarized in Algorithm 11

Algorithm 11 Convex hull-based terrain topography mapping

```

1: Input: Point clouds from Intel RealSense D435i
2: Output: Terrain topography map  $\mathfrak{M}^{\mathcal{T}}$ 
3: function TERRAINMAPPING( $\mathcal{T}$ )
4:   Initialize  $\mathfrak{M}^{\mathcal{T}} \leftarrow \emptyset$  ▷ Start with an empty map
5:   Transform  $\mathcal{T}$  to global frame  $\mathbb{I}$ , resulting in  ${}^{\mathbb{I}}\mathcal{T}$ 
6:   Append  ${}^{\mathbb{I}}\mathcal{T}$  to  $\mathfrak{M}^{\mathcal{T}}$ 
7:    $C_h \leftarrow \text{CONVEXHULL}({}^{\mathbb{I}}\mathcal{T})$  ▷ Create initial convex hull
8:   while  $\hat{\mathcal{T}} \neq \emptyset$  do
9:     Acquire new segmented ground point cloud,  $\hat{\mathcal{T}}$ 
10:    Transform  $\hat{\mathcal{T}}$  to global frame  $\mathbb{I}$ , resulting in  ${}^{\mathbb{I}}\hat{\mathcal{T}}$ 
11:     $\hat{C}_h \leftarrow \text{conv}({}^{\mathbb{I}}\hat{\mathcal{T}})$ 
12:     $\tilde{\mathcal{T}} \leftarrow {}^{\mathbb{I}}\hat{\mathcal{T}} \setminus (C_h \cap \hat{C}_h)$  ▷ Find points outside the intersection
13:    Append  $\tilde{\mathcal{T}}$  to  $\mathfrak{M}^{\mathcal{T}}$  ▷ Update the terrain map
14:     $C_h \leftarrow \hat{C}_h$  ▷ Update the current convex hull
15:   end while
16:   return  $\mathfrak{M}^{\mathcal{T}}$ 
17: end function

```

4.8.2 Experimental validation of terrain topography mapping

The terrain topography modeling algorithm, as delineated in the previous sections, produces the topography as a set of spatial coordinates in $\mathbb{R}^3$ via an inverse sensor mapping approach and a convex hull-based mapping approach. The actual terrain topography, $\mathfrak{T}$ indeed, is also a set of spatial coordinates in $\mathbb{R}^3$, which span the surface of the terrain. Con-

sequently, the question of validating the terrain topography modeling algorithms devised in this study arises. In pursuit of this objective, an experimental validation was conducted. Indeed, digital elevation models (DEMs) procured via manual techniques like global positioning system (GPS) or passive techniques like stereoscopic satellite data can be used as a ground truth to validate the devised algorithms. The accuracy of these DEMs, however, is contingent upon the sampling density, the methodologies employed to derive height data, and the interpolation techniques utilized. Moreover, the spatial resolution of DEMs often fails to capture the terrain topography accurately at resolutions finer than meters. It is also observed that terrain-adaptive features like slopes and surface curvature can induce significant errors in DEMs [94]. Additionally, accessing these DEMs can necessitate proprietary permissions. Given these challenges in using DEMs as the ground truth, and the overarching goal of developing *in-situ* terrain topography modeling algorithms, it necessitates to devise a more reliable validation study to assess the algorithms' proficiency at capturing terrain details at resolutions as fine as centimeters, thereby facilitating more accurate terrain analysis in uncharted environments.

The *terramechanics test rig* at the Terramechanics, Multibody dynamics and Vehicle Systems Lab, Virginia Tech [95] as shown in Figure 4.16 and 4.15, demonstrates the capability to maneuver agricultural tires under varied conditions, such as toe angles, camber, inflation pressure, and angular velocities. This rig can be equipped with diverse soil types, including clay and sand. Leveraging the fact that the rig provides a controlled environment of the terrain, it allows for the easy and clear acquisition of desired topography like ruts in clay. To challenge the capabilities of the algorithms developed to delineate terrain topography at exceptionally fine resolutions, the task was set to ascertain the rut topography using the two devised algorithms and compare it against the rut profile captured by the Einstar laser scanner, which was considered the ground truth.

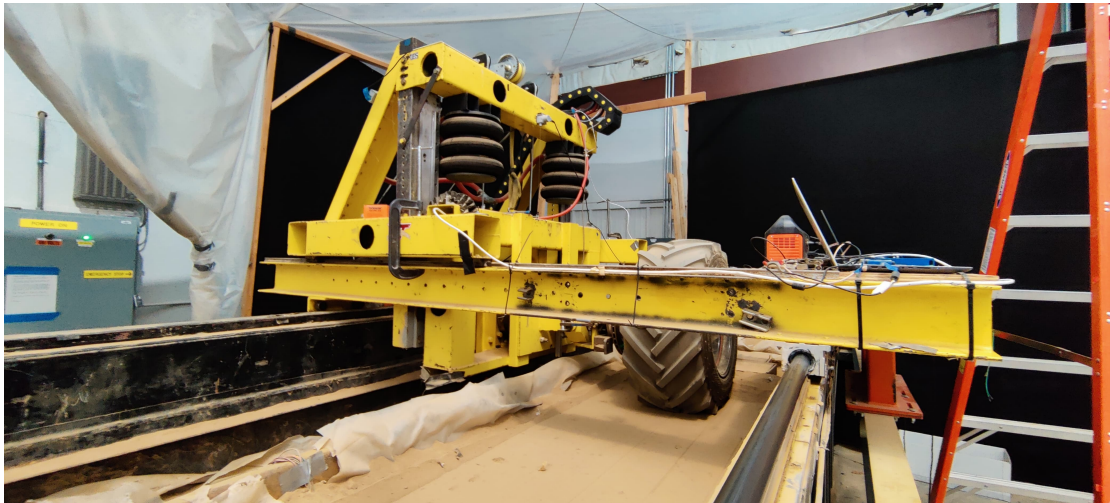


Figure 4.15: Terramechanics rig: Controlled experimental facility with visible terrain features in the order of centimeters. The experimental rig consists of a carriage that allows for longitudinal motion along the rut, which was used for testing the accuracy of the on-board navigation system



Figure 4.16: Terramechanics rig: Controlled experimental facility with visible terrain features in the order of centimeters. The experimental rig consists of a carriage that allows for longitudinal motion along the rut, which was used for testing the accuracy of the on-board navigation system

To this goal, an agricultural tire was initially deployed on clay, leaving a discernible rut in its wake as can be seen in Figure 4.16. Subsequently, the Intel RealSense D435i camera, paired with the Intel RealSense T265 camera, was mounted on the terramechanics test rig. The rig, now without the tire, was then operated to traverse along the rut at a speed of 2 mph. During this operation, the two devised algorithms were employed simultaneously namely, the voxel map-based topography algorithm and the point cloud-based topography algorithm - both utilizing the RANSAC-based ground segmentation method. Following the experiment, the Einstar Laser scanner was utilized to capture the terrain's detailed profile, ensuring high fidelity in the data acquisition process. The data obtained using the Einstar laser scanner is seen in Figure 4.17.

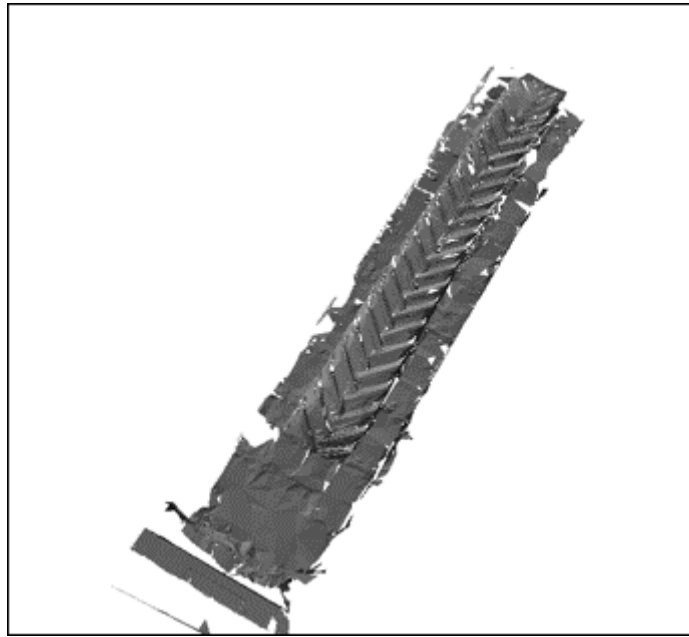


Figure 4.17: 3D model of rut captured using a laser scanner: The *.stl* file representation of the tire track rut provides high-resolution terrain data, used for validating UGV navigation algorithms against precise terrain features

The Einstein laser scanner data was procured in *.stl* file format, from which point cloud data was recovered. This represents the ground truth, $\mathfrak{T}$. The ground truth point

cloud was then compared to the results obtained using both the inverse-sensor model based topography modeling algorithm and the point cloud based topography modeling algorithm. As illustrated in Figure 4.19, the rut profile on the terramechanics rig is effectively captured by both algorithms and aligns well with the data obtained using the laser scanner.

To further quantify the similarity between the three-dimensional point clouds, a normalized Chamfer distance was calculated using the following formula:

$$d_{chamfer} = \frac{1}{|S_1| + |S_2|} \left(\sum_{a \in S_1} \min_{b \in S_2} \|a - b\|_2^2 + \sum_{b \in S_2} \min_{a \in S_1} \|a - b\|_2^2 \right) \quad (4.16)$$

where $|\cdot|$ represents the cardinality of the point cloud sets S_1 and S_2 and both $\{a\}, \{b\} \subset \mathfrak{M}^T \in \mathbb{R}^3$. This distance metric measures the average of the smallest squared Euclidean distances between each point in one set to its nearest point in the other set, effectively capturing the discrepancies in a three-dimensional space.

The Chamfer distance [96] between the topography produced by the laser scanner, $\mathfrak{T}$ and those generated by both the voxel mapping based topography and the convex hull based topography, $\mathfrak{M}^T$ was found to be 0.001m. This low value indicates a high degree of similarity between the measured and modeled topologies, confirming the efficacy of the algorithms in replicating the true terrain features observed by the laser scanner.

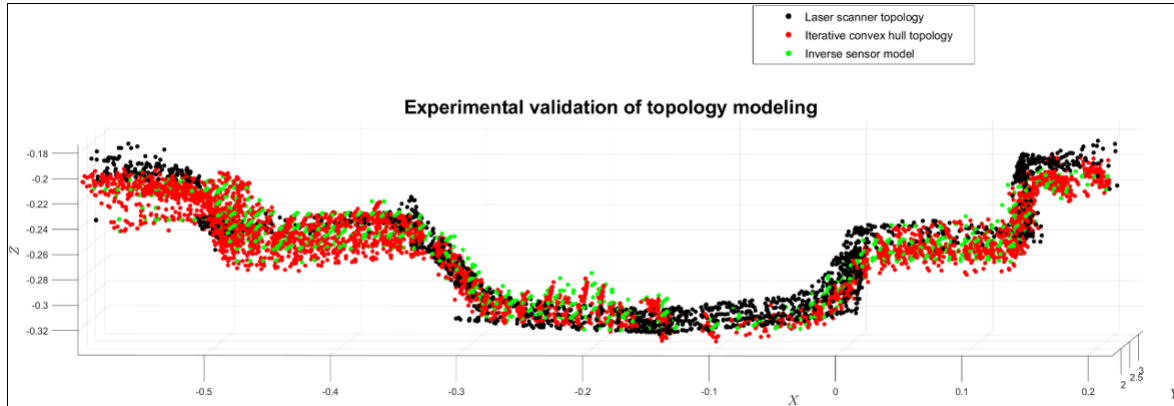


Figure 4.18: Experimental validation of terrain topography modeling: The graph compares terrain topography captured by a laser scanner (black), iterative convex hull topography (red), and inverse sensor model (green). The results demonstrate the accuracy of the proposed algorithms against the ground truth obtained from the laser scanner

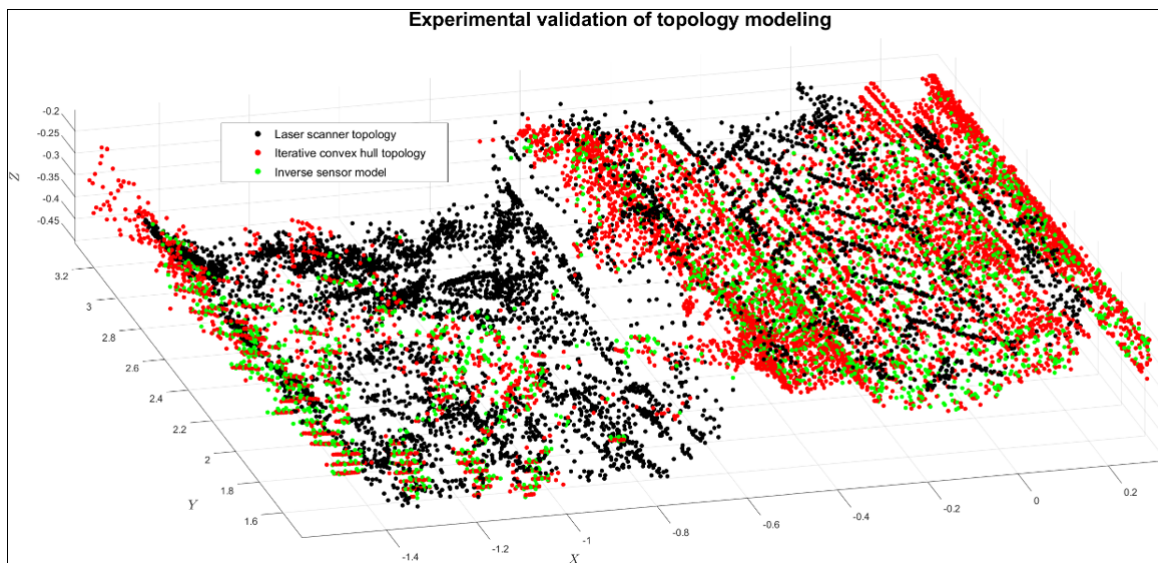


Figure 4.19: Experimental validation of terrain topography modeling: The graph compares terrain topography captured by a laser scanner (black), iterative convex hull topography (red), and inverse sensor model (green). The results demonstrate the accuracy of the proposed algorithms against the ground truth obtained from the laser scanner

Following the validation of the topography mapping algorithm's accuracy in a controlled lab environment at lower resolutions, the navigation system was deployed on the Clearpath Husky robot to capture terrain topography in an outdoor setting. The experiment was con-



Figure 4.20: Field test of terrain topography modeling: The image shows a bumpy terrain whose topography is modeled using the developed navigation system

ducted on an uneven terrain with a muddy texture, where the robot was driven at speeds ranging from 1 to 1.5 m/s. The topography was mapped using the voxel-based approach described in Section 4.6.1, executed on board the Husky. The resulting terrain topography maps, as shown in Figure 4.20 and 4.21, exhibit both qualitative agreement with the physical terrain and reasonable elevation data. The color gradients in the plots, representing height variations, align with the observed undulations and slopes of the terrain, demonstrating the algorithm's ability to capture the geometric attributes of uneven surfaces accurately. This topographic representation supports the planning algorithms in generating terrain-aware trajectories, enhancing the UGV's navigation capabilities in challenging outdoor environments.

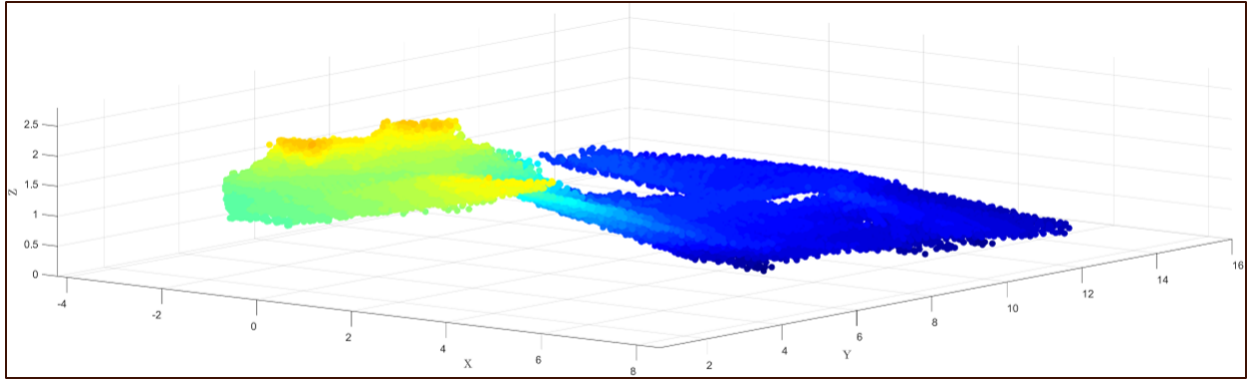


Figure 4.21: Experimental validation of terrain topography modeling: The scatter plot represents the terrain points in the inertial frame, capturing the terrain topography. Green represents higher elevation, whereas blue represents lower elevation

4.9 Conclusions and Future Work

This chapter has addressed the critical challenge of autonomous navigation for an unmanned ground vehicle (UGV) in GNSS-denied environments without *a priori* maps, a scenario that demands robust real-time localization and mapping capabilities. The primary problems identified include the absence of pre-existing environmental models, the need for accurate pose estimation amidst sensor drift, and the requirement to distinguish terrain features from obstacles in uncharted terrains to enable terrain-aware tactical path planning. These challenges were compounded by the computational constraints of real-time operation on resource-limited platforms, necessitating efficient algorithms that balance accuracy with processing demands. The work presented herein developed a comprehensive navigation framework, integrating sensing, segmentation, and mapping techniques to support the tactical motion planning algorithms delineated in Chapters 2 and 3, ultimately facilitating stealthy navigation in hostile environments.

The contributions of this chapter are multifaceted, addressing the identified challenges through novel algorithms and experimental validations. First, a robust sensor fusion strategy was implemented using an Extended Kalman Filter (EKF) to integrate data from the Intel

RealSense T265 camera, the MicroStrain 3DM-GX3-25 IMU, and wheel encoders on the Clearpath Husky, as detailed in Section 4.4.1. This approach significantly mitigated pose drift, achieving accurate 6DoF pose estimation in the global frame $\mathbb{I}$, as demonstrated by the reduced drift in the EKF-fused path compared to the T265’s dead reckoning (Figure 4.3). Second, two complementary point cloud segmentation algorithms were developed to classify ground points $\mathcal{T}$ and obstacles $\mathcal{O}$ from dense point clouds (Section 4.5). The Random Sample Consensus (RANSAC) algorithm effectively identified ground planes in controlled settings, while the novel inverse ray tracing (IRT) method leveraged geometric perspectives to extract terrain points in uneven environments, enhancing robustness (Section 4.5.3). Third, advanced mapping techniques were introduced for both occupancy and terrain topography (Sections 4.6 and 4.8). A voxel-based mapping approach, employing a Bayesian inverse sensor model, was developed to construct the 3D occupancy grid $\mathfrak{M}^{\mathcal{O}}$ and terrain topography map $\mathfrak{M}^{\mathcal{T}}$ in real time, with the latter benefiting from a high-resolution representation to capture terrain features (Section 4.6.1). Additionally, a novel iterative convex hull method was proposed to directly collate point clouds into a sparse representation of $\mathfrak{M}^{\mathcal{T}}$ in $\mathbb{R}^3$, reducing computational overhead while preserving topographic details.

Experimental validations underscored the efficacy of these contributions across diverse settings. Indoor tests in a controlled lab environment at Virginia Tech (Section 4.6.2) demonstrated qualitative alignment between the voxel maps and the physical layout, with RViz visualizations capturing obstacle heights and free spaces effectively (Figure 4.12). A quantifiable validation of the 2D occupancy grid $\mathfrak{M}^{\mathcal{O}}$ in an indoor hallway confirmed precise spatial correspondence, where a real-world distance of 5 tiles (3 meters) was accurately reflected as 5 voxels in the map (Section 4.7.1, Figure 4.13). Outdoor experiments on uneven, muddy terrain further validated the terrain topography mapping algorithms, with both the voxel-based and convex hull methods showing qualitative agreement with the physical terrain and

reasonable elevation data (Figure 4.21). In a controlled terramechanics rig setup, the algorithms were rigorously compared against ground truth data from an Einstar laser scanner, achieving a normalized Chamfer distance of 0.001 m, indicating high fidelity in capturing terrain features at centimeter-scale resolutions (Section ??, Figure 4.19). These validations affirm the algorithms' suitability for tactical navigation, providing reliable environmental models for path planning in GNSS-denied, uncharted settings.

Future work could explore several directions to enhance the scalability and robustness of the proposed navigation framework. First, integrating higher-range sensing technologies, such as the Stereolabs ZED 2i stereo camera, could improve mapping accuracy over larger areas and under diverse lighting conditions, addressing limitations observed in feature-scarce environments. Second, extending the algorithms to handle high-speed navigation (*e.g.*, speeds exceeding 2 m/s) could be investigated, potentially requiring adaptive resolution adjustments to maintain real-time performance. Third, incorporating terrain semantics (*e.g.*, grass, sand) through advanced classification techniques, as surveyed in Section 4.2, could enhance the op-tempo velocity planner by informing traction and slip models, thereby improving dynamic feasibility on varied surfaces. Finally, exploring dynamic map updates to account for moving obstacles or changing terrain conditions could further bolster the framework's applicability in real-world hostile environments. The next chapter delineates the integration of the navigation stack with the motion planning algorithms for complete autonomous navigation.

Chapter 5

Real-time experiments

Contributions

This chapter presents experimental field tests demonstrating the deployment and validation of the full tactical autonomy stack on a UGV platform. The key contributions are:

- Real-world deployment of the autonomy stack on the Clearpath Husky robotic platform.
- Experimental validation of the global tactical path planner in GNSS-denied off-road environments.
- Evaluation of the trajectory planner and its ability to generate feasible time-parameterized commands in field conditions.
- Full-system integration and testing of navigation, planning, and control modules under realistic mission scenarios.

5.1 Introduction

The preceding chapters established the theoretical foundations and simulation-based validation of a comprehensive motion planning framework for tactical navigation of UGVs in GNSS-denied, hostile environments. Chapter 2 developed global path planning algorithms,

specifically the tactical A^* and LPA^* planners, integrating stealth-inducing modifications and dual-mode obstacle interaction to generate waypoints that balance concealment and efficiency. Chapter 3 on the other hand, added to this framework by introducing a cubic polynomial-based trajectory planner, incorporating op-tempo velocity regulation, terrain-aware attitude planning via rotation-minimizing frames (RMFs), and collision avoidance strategies, validated through simulations. Chapter 4 then, addressed the navigation stack, leveraging real-world sensors (Intel RealSense D435i, T265, and IMU) to perform pose estimation, point cloud segmentation, and mapping, producing occupancy grids $\mathfrak{M}^O$ and terrain topography maps $\mathfrak{M}^T$ validated in both controlled and outdoor settings (Sections 3.8.4 and ??).

This chapter bridges the gap between simulation and real-world deployment, applying the insights gained from prior chapters to enable autonomous navigation in uncharted environments. The understanding of algorithmic performance—such as LPA^* 's dynamic adaptability, the double projection RMF method's 20% faster computation, and the navigation stack's mapping accuracy—is leveraged to select efficient algorithms for real-time implementation. These algorithms are integrated with the navigation stack, which generates real-time maps using onboard sensors, to autonomously produce tactical paths and trajectories on the Clearpath Husky platform. The chapter is structured as follows: Section 5.2 describes the Clearpath Husky, detailing its hardware specifications, the onboard computational setup for real-time execution, and the ROS-based communication framework. Section 5.3 outlines the architecture of the integrated autonomy stack. Section 5.4 presents the experimental test scenarios, evaluating the path planner (Section ??), trajectory planner (Section ??), and full-stack deployment (Section ??) in real-time, focusing on performance metrics such as path accuracy, computational latency, and tactical effectiveness in hostile environments. Section 5.5 summarizes the findings and discusses future directions.

5.2 Clearpath Husky Robotic platform

Real-world validation of tactical, terrain-aware motion planning algorithms for unmanned ground vehicles (UGVs) in GNSS-denied, hostile environments demands a robust experimental platform capable of reliable off-road navigation and seamless integration with advanced sensor suites and computational payloads. The landscape of available platforms is diverse, ranging from scaled remote-controlled (RC) cars, such as the Traxxas 1/10th model [97], to compact robotic platforms like the TurtleBot or AgileX Scout Mini, and full-scale vehicles like the HMMWV or Polaris MRZR employed in military research [98]. Early platforms, such as Carnegie Mellon’s Terregator, demonstrated the feasibility of autonomous off-road navigation, while DARPA’s Grand Challenge and subsequent programs like the Crusher and RACER fleets highlighted the need for rugged, adaptable testbeds. However, full-scale vehicles incur high costs and maintenance challenges, whereas smaller RC cars, despite affordability, often lack the payload capacity or durability for complex outdoor missions. The present research aims to develop scalable algorithms for tactical navigation that balance stealth, terrain adaptability, and computational efficiency, with potential applications in military logistics and reconnaissance, benefiting entities like the U.S. Army Research Laboratory (ARL) and Ground Vehicle Systems Center (GVSC).

ARL’s Scalable, Adaptive, and Resilient Autonomy (SARA) program, focused on advancing off-road UGV autonomy, selected the Clearpath Husky for its terrain classification and navigation experiments due to its ruggedness, payload capacity, and open-source compatibility. Consequently, this research adopts the Clearpath Husky as the test platform, leveraging its proven reliability in military research and compatibility with the proposed autonomy stack to validate tactical navigation algorithms in uncharted, hostile environments

5.2.1 Vehicle description

The Clearpath Husky is a four-wheeled, skid-steer unmanned ground vehicle. As depicted in the Figure 5.1, the Husky measures 990 mm in length, 670 mm in width, and 390 mm in height, with a track of 555 mm and a wheelbase of 512 mm. The vehicle weighs 50 kg and offers a maximum payload capacity of 75 kg on relatively flat terrain, reduced to 20 kg under all-terrain conditions, such as climbing a 30-degree grade or turning in high-friction scenarios. This payload capacity supports the integration of the sensor suite used in this study—comprising an Intel RealSense D435i depth camera, an Intel RealSense T265 tracking camera, a 3DM-CX5-10 inertial measurement unit (IMU), and wheel encoders—as well as the onboard computational platform, an Intel i7-9750H CPU (2.60 GHz) laptop running ROS Noetic on Ubuntu 20.04 LTS, as detailed in Chapter 4. Furthermore, the Husky is equipped with a computer on-board running ROS Noetic on a linux operating system.

The Husky achieves a maximum speed of 1.0 m/s, with a ground clearance of 130 mm, enabling it to traverse obstacles up to 80 mm in height and climb grades up to 45 degrees, while maintaining stability on traversal grades up to 30 degrees. The vehicle’s wheel encoders, offering a resolution of 78,000 ticks per meter, provide precise odometry data critical for navigation in GNSS-denied environments. The Husky’s chassis, constructed from durable aluminum, is designed to operate within an ambient temperature range of 10 to 30 °C, ensuring reliability in the outdoor, hostile settings targeted by this research. These specifications, combined with the Husky’s modular payload bay, enable seamless integration of the proposed autonomy stack, facilitating real-time execution of tactical path planning, trajectory generation, and navigation algorithms in uncharted terrains



Figure 5.1: Clearpath Husky UGV with key dimensions: 990 mm (length), 670 mm (width), 390 mm (height), 555 mm (track), and 512 mm (wheelbase). The platform is shown equipped with sensors used in this study, including Intel RealSense cameras and an IMU, mounted on the payload bay.

5.2.2 Distributed ROS Communication Architecture

The experimental setup for the Clearpath Husky leverages a distributed communication architecture built on the Robot Operating System (ROS) 1 Noetic ecosystem, facilitating seamless integration of the autonomy stack across two computational nodes. The Husky’s onboard computer, pre-installed by Clearpath Robotics, serves as the ROS master, managing the core system nodes and interfacing directly with the vehicle’s internal sensors, including the 3DM-CX5-10 IMU and wheel encoders (78,000 ticks/m resolution), which publish odometry and orientation data to topics such as `/odometry/filtered` and `/imu/data` at near 100 Hz. The second node, a laptop equipped with an Intel i7-9750H CPU (2.60 GHz) running Ubuntu 20.04 LTS and ROS Noetic, operates as a ROS client. This laptop processes the navigation stack and motion planning stack, handling data from the Intel RealSense D435i and T265 cameras, which are connected via USB at near 30 Hz, as detailed in Chapter 4.

The two machines communicate over a direct Ethernet connection, ensuring low-latency and reliable data exchange without reliance on external networks, a critical feature for operation in GNSS-denied, hostile environments. To mitigate timing discrepancies in this distributed setup, the clocks of both computers are synchronized at startup using the Network Time Protocol (NTP), ensuring consistent timestamps for sensor fusion and navigation tasks. The ROS master-client architecture is configured by setting the `ROS_MASTER_URI` on the laptop to point to the Husky’s onboard computer. Key data flows include the navigation stack on the laptop publishing occupancy grids $\mathfrak{M}^o$, waypoint sequences from the tactical planners, and velocity commands to the Husky’s `/cmd_vel` topic, which the Husky’s controller subscribes to for motor actuation. This setup ensures real-time execution of the autonomy stack while maintaining isolation and security in contested environments.

5.3 Autonomy stack architecture

This section delineates the architecture of the integrated autonomy stack deployed on the Clearpath Husky, focusing on the engineering implementation and operational integration of the system's modules within the Robot Operating System (ROS) 1 Noetic framework. The stack leverages the ROS publish-subscribe framework to facilitate modular data flow across multiple nodes, ensuring scalability and fault isolation, as established in the distributed communication setup (Section 5.2). Figure 5.2 provides a visual representation of the ROS architecture, illustrating the interactions between nodes responsible for processing sensor data, generating maps, planning paths, computing trajectories, and controlling the UGV.

The data flow initiates with the perception modules, where the Intel RealSense T265 camera provides visual-inertial odometry data, and the D435i camera captures depth information as point clouds. The T265 data, combined with measurements from the 3DM-CX5-10 IMU and wheel encoders, is processed through an Extended Kalman Filter (EKF) node to produce fused odometry estimates, ensuring accurate pose estimation in the global frame despite the GNSS-denied setting. Concurrently, the D435i's point clouds are processed by a segmentation node, which separates obstacle and terrain points using the RANSAC and inverse ray tracing methods delineated in Chapter 4, as visualized in the segmented point cloud in Figure 5.2. These points are then transformed using the fused pose and registered into a 3D voxel map by the mapping node, producing probabilistic representations of obstacles and terrain, as shown in the mapped environment in Figure 5.2.

The path planning module, implemented on the ROS client, utilizes the LPA*-based planner to generate a stealth-aware waypoint sequence based on the 2D occupancy grid derived from the 3D voxel map, as detailed in Chapter 2. The planner prioritizes concealment by coasting obstacles, balancing stealth with efficiency, as demonstrated by the planned path

navigating the maze in Figure 5.2. The waypoint sequence is updated in each cycle to connect the UGV's current position to the goal, ensuring adaptability to dynamic map updates. The trajectory planning module, also on the ROS client, employs cubic spline interpolation to produce a time-parameterized trajectory, integrating rotation-minimizing frames (RMFs) for terrain-aware orientation, as visualized in the trajectory with RMFs in Figure 5.2. To mitigate communication latencies observed during real-time tests, the trajectory planner subscribes to the vehicle's pose, using the current orientation as the initial frame for RMF computation, ensuring smooth transitions.

The control module, residing on the ROS master, receives the trajectory planner's velocity commands and executes motor actuation on the Husky, adapting to unmodeled uncertainties. The processes operate in parallel across ROS nodes, leveraging the pub-sub framework's asynchronous communication for real-time responsiveness, as each module independently processes and exchanges data. Engineering challenges included managing latency between map updates and trajectory replanning, addressed by optimizing node synchronization and prioritizing pose updates, ensuring the system's operational reliability in the indoor lab maze environment.

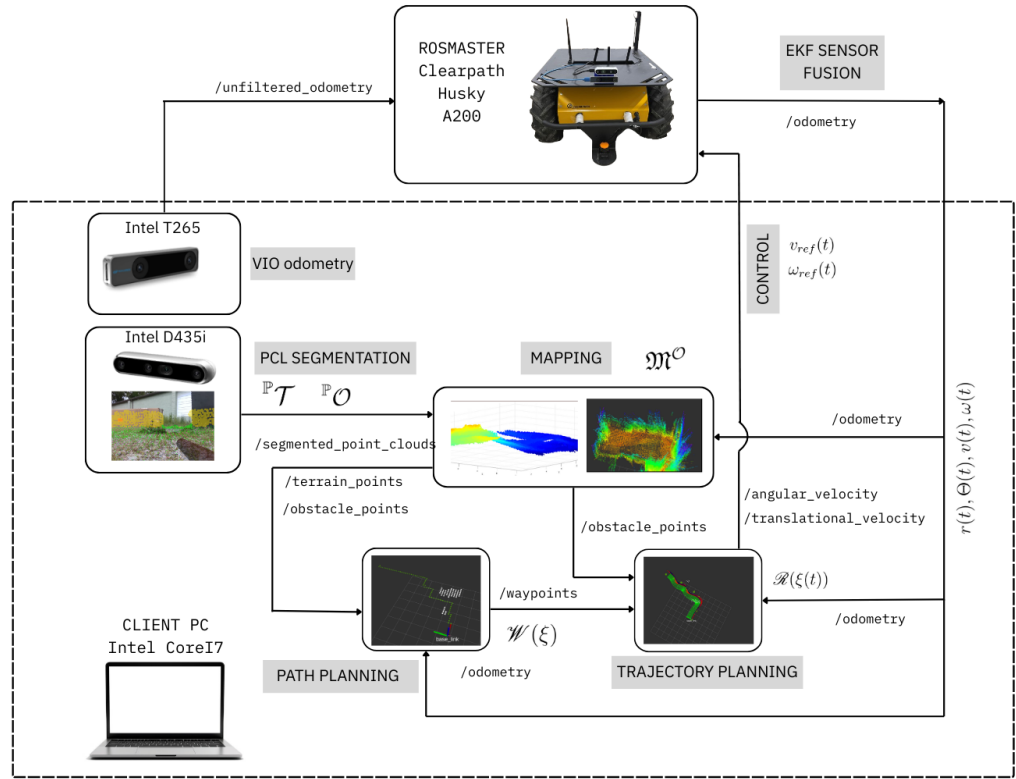


Figure 5.2: ROS-based autonomy stack for tactical navigation on the Clearpath Husky A200. The diagram illustrates the data flow from sensors (Intel RealSense T265 and D435i) through EKF sensor fusion, point cloud segmentation, mapping, path planning, and trajectory planning modules, culminating in control commands

5.4 Experimental Test Scenario

5.4.1 Test Scenario Description

The experimental test scenario validates the tactical navigation framework in an indoor environment, reflecting the mission requirements outlined in Chapter 2 for a simulated GNSS-denied setting. In simulation, the UGV was tasked with navigating from a start to a goal while prioritizing stealth and handling obstacles on uneven terrain. The indoor scenario,

depicted in Figure 5.3, consists of a lab space at the Advanced Control Systems Lab, Virginia Tech, transformed into a maze using cardboard boxes as obstacles. The lab features concrete flooring, workstations, and scattered equipment, with the maze constructed to create narrow, curved pathways. The objective is to autonomously drive the Clearpath Husky from a start point to coast a curved path along the maze, reaching a goal set, while generating a real-time map of the environment.

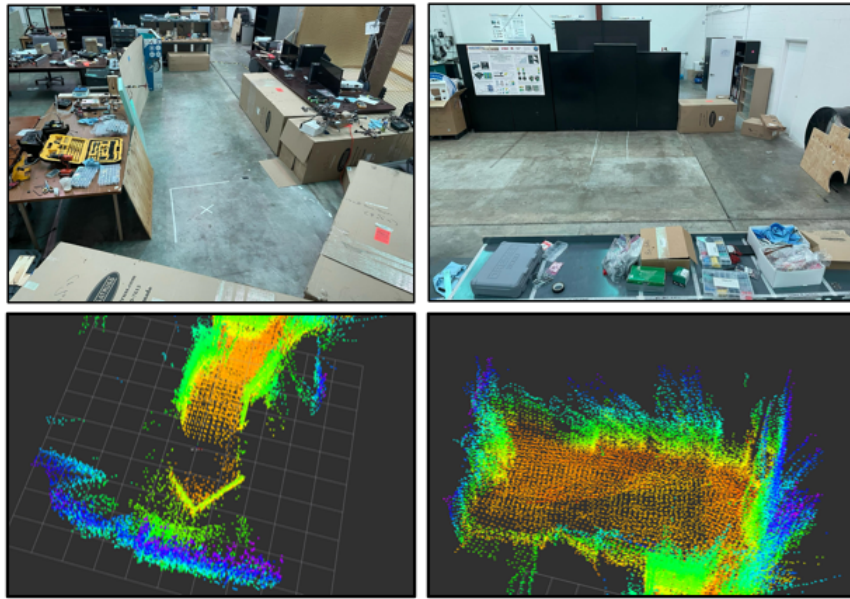


Figure 5.3: Indoor test scenario in a lab environment. The top images show the maze constructed with cardboard boxes. The bottom left image displays the RViz visualization of the voxel map.

The cardboard boxes are captured as obstacles in the occupancy grid $\mathfrak{M}^{\mathcal{O}}$, while the concrete floor’s variations are represented in the topography map $\mathfrak{M}^T$, as described in Chapter 4. The path planner, utilizing LPA*, generates a sequence of waypoints $\mathscr{W}$ to navigate the maze’s curves, followed by the trajectory planner computing a time-parameterized trajectory $\mathscr{R}(t)$ for smooth traversal. This setup evaluates the autonomy stack’s performance in a controlled yet complex indoor environment, as detailed in the following subsection.

5.4.2 Module Evaluation

Navigation Module:

The navigation module, leveraging the Intel RealSense D435i and T265 cameras, processes sensor data to enable real-time localization and mapping. The D435i produces point clouds at 30 Hz, as confirmed using `rostopic hz`, but real-world data exhibited significant noise, particularly in cluttered indoor settings. Probabilistic filtering or voxel grid (leaf) filtering, while effective for noise reduction, slowed the pipeline considerably, impacting real-time performance. Instead, a simpler downsampling approach was implemented, maintaining the 30 Hz rate while reducing computational load, though at the cost of some detail loss. The RANSAC-based segmentation, used to distinguish obstacles and terrain, occasionally failed due to data latency in the TF transforms, especially under high computational loads. The T265, providing odometry at 30 Hz, faced challenges in poor lighting conditions, abruptly stopping data output and disrupting pose estimation, which affected mapping consistency. In contrast, the Husky's onboard odometry, published at 50 Hz via `/odometry/filtered`, provided faster updates than the T265, and the fused odometry (via EKF) performed reliably at slow speeds (0.5 m/s), ensuring stable localization during tests.

Path Planner Evaluation:

The path planning module, employing the LPA* algorithm, was rigorously evaluated for its ability to generate a stealth-aware waypoint sequence $\mathcal{W}$ within the indoor lab maze environment, as depicted in Figure 5.4. Implemented in C++ and integrated into ROS Noetic, the planner dynamically updates the start point to the UGV's current pose, ensuring real-time path adaptation. Standalone static tests demonstrated an average execution time of 2000 to 9000 microseconds, indicating high computational efficiency for individual planning cycles. However, in the ROS environment, the waypoint publication rate, measured via `rostopic hz`, averaged 25 Hz, constrained by the dependency on map updates, as the plan-

ner waits for new occupancy grid data before replanning, introducing latency in dynamic scenarios. Parameter tuning for stealth modulation was tested, with the tactical setting producing paths that coasted closer to obstacles (*e.g.*, cardboard boxes) for concealment, while a less cautious setting prioritized efficiency, as observed in RViz visualizations. A critical challenge emerged from noisy point cloud maps, where lighting-induced artifacts introduced false obstacles, causing the planner to fail in finding a feasible path—a limitation necessitating enhanced map robustness. When maps were clean, the planner consistently generated optimal routes, whether reckless or tactical, validated through teleoperation tests mimicking navigation apps, with RViz confirming real-time path updates. Projecting waypoints onto the terrain surface enabled 3D path generation, but maintaining a high-resolution topography map $\mathfrak{M}^T$ in the near vicinity was necessary to capture surface variations, highlighting the need for efficient terrain representation strategies in future work.

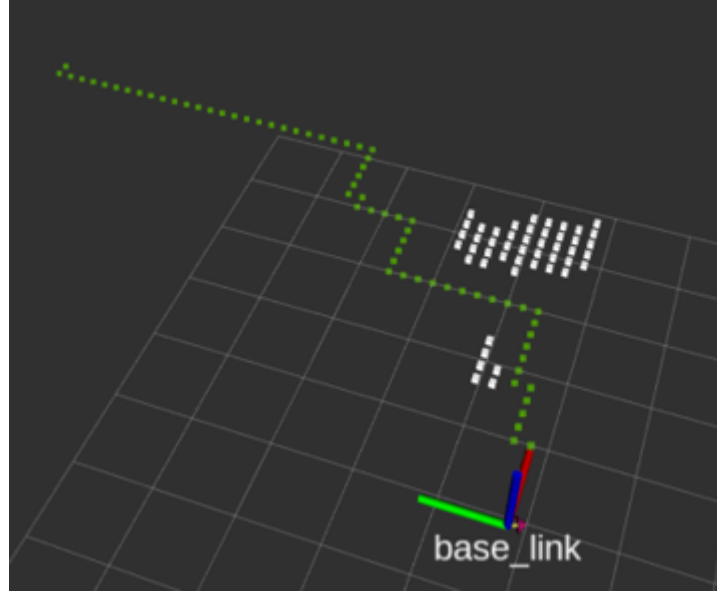


Figure 5.4: Indoor field test results for path planning. Navigation system capturing the lab maze, with boxes in the occupancy grid $\mathfrak{M}^o$ (white) and planned path (green dashed line) from start navigating the maze while avoiding obstacles.

Trajectory Planner Evaluation:

The trajectory planner, implemented in C++ and integrated into ROS Noetic, generates a time-parameterized trajectory $\mathcal{R}(t)$ using cubic spline interpolation, as detailed in Chapter 3. A significant challenge arose when directly using waypoints from the path planner: by the time waypoints were received and velocities computed, the UGV had moved, rendering the “current pose” outdated and resulting in erratic trajectories. To address this, the planner was modified to filter waypoints, selecting only those beyond a set threshold from the current pose (obtained via `/odometry/filtered`) and ensuring at least three monotonically increasing points for cubic spline computation using the GSL library’s Hermite interpolation. Attitude planning followed, where the initial frame of reference was derived by converting the odometry orientation into a rotation matrix, then applying parallel transport for RMF computation via the double projection method, which proved computationally efficient, executing in under 1500 microseconds in static tests. Velocities were calculated using Rodrigues’ method and SLERP, with SLERP being slower. Another observation was that, at low velocities (*e.g.*, 0.1 m/s), the angular velocity ω often fell below 0.02 rad/s - which is expected, however practical requirements to enable angular motion requires a certain baseline angular velocity for the lower level control to implement the wheel torques. Manual tuning of gains scaled ω , but the approach was velocity-dependent, requiring careful adjustment at higher speeds (*e.g.*, 0.5 m/s) where ω values were naturally larger. The trajectory publication rate, limited by the navigation system, averaged 25 Hz via `rostopic hz`, yet RViz visualizations (Figure 5.5) confirmed smooth, terrain-aware navigation when waypoints were available.

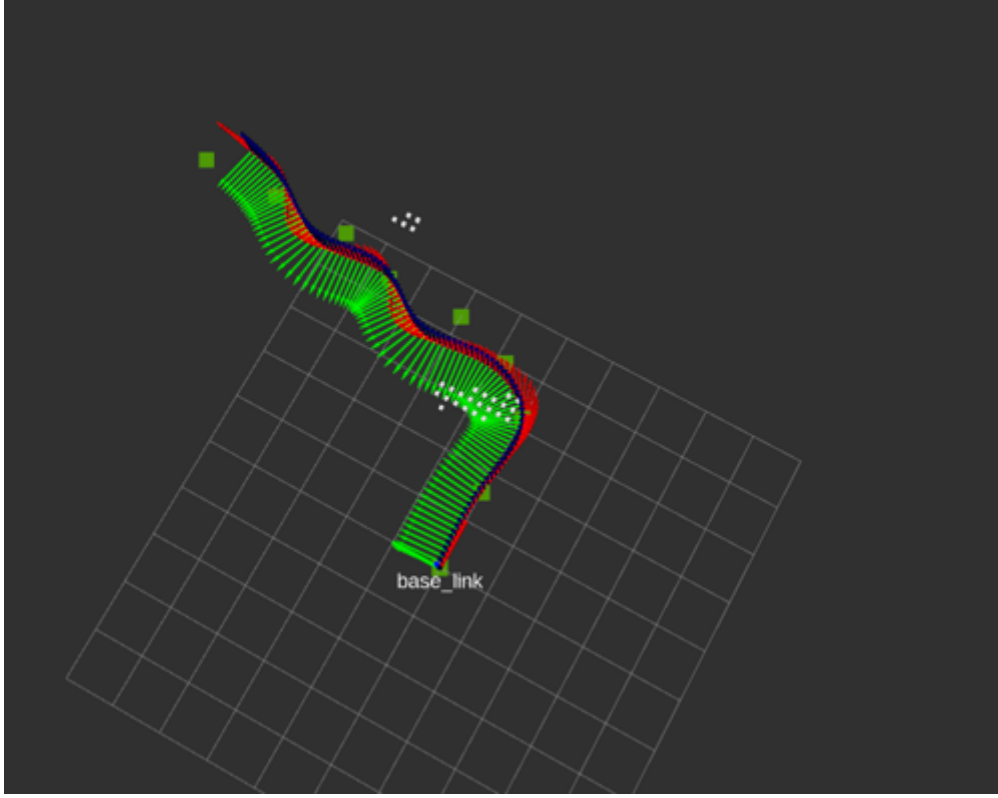


Figure 5.5: Trajectory planning results in the indoor lab maze. The trajectory (blue curve) is overlaid on the point cloud map, with $SO(3)$ frames (red, green, blue axes) visualizing the RMFs along the path, ensuring smooth, terrain-aware navigation.

5.5 Conclusions

This chapter successfully demonstrated the real-world deployment of the tactical navigation framework on the Clearpath Husky platform within an indoor lab maze environment at the Advanced Control Systems Lab, Virginia Tech. The navigation stack, leveraging the Intel RealSense D435i and T265 cameras, effectively captured the environment despite challenges, producing occupancy grids $\mathfrak{M}^{\mathcal{O}}$ for obstacles (cardboard boxes) and topography maps $\mathfrak{M}^T$ for the concrete floor, as validated through RViz visualizations (Section 5.4). The D435i's 30 Hz point cloud data required downsampling to mitigate noise-induced slowdowns,

while the T265’s odometry faced disruptions in poor lighting, underscoring the need for robust sensor performance in dynamic settings. The LPA*-based path planner generated a stealth-aware waypoint sequence $\mathcal{W}$, navigating the maze’s curved pathways while avoiding obstacles, though its 25 Hz planning rate was constrained by map update latency, and noise-induced false obstacles occasionally caused path failures. The trajectory planner produced a smooth, time-parameterized trajectory $\mathcal{R}(t)$ using cubic spline interpolation, with rotation-minimizing frames (RMFs) ensuring terrain-aware orientation, as evidenced by the $SO(3)$ frame visualizations in RViz. However, low angular velocities at slow speeds (*e.g.*, 0.1 m/s) required manual gain tuning to meet the Husky’s threshold, highlighting latency issues across threads.

Preliminary tests of the full autonomy stack, integrating navigation, path planning, and trajectory planning, confirmed autonomous motion along the planned trajectory in the lab maze, validating the framework’s capability in a controlled, GNSS-denied setting. RViz observations during teleoperation tests demonstrated the system’s ability to dynamically update paths and trajectories, adapting to the UGV’s pose in real-time, though challenges such as map update delays and reference tracking inefficiencies were evident. The high-resolution topography map $\mathcal{M}^T$ posed memory inefficiencies on flat terrain, suggesting a future adaptive storage approach (*e.g.*, planes for flat regions, points for uneven areas). Future work will focus on enhancing the reference tracking module to improve tracking accuracy and robustness, addressing latency and synchronization issues across nodes, and optimizing terrain map storage for efficiency. Expanding test scenarios to include diverse indoor and outdoor environments will further validate the framework’s adaptability, supporting its application in military logistics and reconnaissance tasks in hostile, uncharted terrains.

Chapter 6

Conclusions and Future Work

6.1 Problems Addressed

This dissertation tackled the overarching challenge of autonomous tactical navigation for unmanned ground vehicles (UGVs) in GNSS-denied, hostile environments without *a priori* maps, a scenario critical for military logistics and reconnaissance. The primary problems addressed include the absence of pre-existing environmental models, necessitating real-time localization and mapping (Chapter 4), and the need for stealth-aware path planning that balances concealment with efficiency in uncharted terrains (Chapter 2). Additionally, the work addressed the generation of smooth, terrain-aware trajectories with tactical velocity regulation and collision avoidance, ensuring safe navigation over uneven surfaces (Chapter 3). Computational constraints on resource-limited platforms were a recurring challenge, requiring efficient algorithms to enable real-time operation while maintaining accuracy and robustness (Chapter 5). These challenges were compounded by the need to distinguish terrain features from obstacles and adapt to changing conditions, all while ensuring the UGV's survivability in contested environments.

6.2 Contributions and Developments

The research developed a comprehensive tactical navigation framework, integrating novel algorithms and an experimental setup on the Clearpath Husky platform. Chapter 4 introduced a navigation stack with robust sensor fusion via an Extended Kalman Filter (EKF), combining data from the Intel RealSense D435i, T265, and 3DM-CX5-10 IMU, achieving accurate 6DoF pose estimation (Section 4.4.1). It also contributed point cloud segmentation algorithms (RANSAC and inverse ray tracing) and voxel-based mapping techniques to produce occupancy grids $\mathfrak{M}^{\mathcal{O}}$ and topography maps $\mathfrak{M}^{\mathcal{T}}$ (Sections 4.6, 4.8). Chapter 2 developed tactical A^* and LPA^* planners with stealth-inducing modifications, incorporating a dual-mode obstacle interaction paradigm to distinguish shelters from hazards (Section 2.6). Chapter 3 proposed a cubic spline-based trajectory planner with op-tempo velocity regulation, terrain-aware attitude planning via rotation-minimizing frames (RMFs), and collision avoidance strategies, including a novel double projection method for $SO(3)$ planning (Section 3.4.2). Chapter 5 deployed this framework in an indoor lab maze, validating the navigation, path planning, and trajectory planning modules through real-world tests.

6.3 Key Learnings

The research yielded significant insights across its components. The navigation stack's EKF fusion reduced pose drift, while the inverse ray tracing method enhanced terrain segmentation in uneven environments, achieving a normalized Chamfer distance of 0.001 m against ground truth. The LPA^* planner and A^* planner were evaluated in situations with incremental map discovery to generate waypoints, producing tunable tactical paths, and revealed an emergent benefit of higher path entropy (1.5 bits), disrupting adversarial predic-

tion. The trajectory planner’s double projection method for RMFs reduced computational time by 20% compared to Wang’s approach in [64], ensuring smooth orientation with near-zero torsion (Section 3.8.1). Rodrigues interpolation for angular velocities offered the best balance of accuracy and efficiency, supporting real-time deployment (Section 3.8.3). Field tests confirmed the framework’s capability to navigate a lab maze, with the path planner avoiding obstacles and the trajectory planner generating feasible trajectories, though preliminary full-stack tests highlighted the need for improved reference tracking (Section 5.5).

6.4 Future Recommendations

Future work should focus on enhancing the framework’s scalability and robustness. First, integrating higher-range sensors like the Stereolabs ZED 2i could improve mapping accuracy in larger, feature-scarce environments (Chapter 4). Second, extending the algorithms for high-speed navigation by implementing adaptive resolution adjustments could maintain real-time performance (Chapter 4). Third, optimizing the reference tracking module to improve tracking accuracy and robustness is critical, enabling seamless full-stack integration for complex scenarios (Chapter 5). Fourth, incorporating terrain semantics (*e.g.*, grass, sand) through advanced classification could enhance the *op-tempo* velocity planner by informing traction models (Chapter 3). Finally, exploring dynamic map updates for moving obstacles and entropy-driven path planning to further confound adversarial tracking could bolster tactical effectiveness in real-world hostile environments (Chapter 2), ensuring the framework’s applicability across diverse military applications.

Bibliography

- [1] J. A. Marshall, R. B. Anderson, W.-Y. Chien, E. N. Johnson, and A. L’Afflitto, “A guidance system for tactical autonomous unmanned aerial vehicles,” *Journal of Intelligent & Robotic Systems*, vol. 103, pp. 1–36, 2021.
- [2] J. A. Marshall, P. Binder, and A. L’Afflitto, *Autonomous Multi-rotor Unmanned Aerial Vehicles for Tactical Coverage*. Cham: Springer Nature Switzerland, 2024, pp. 31–76.
- [3] T. W. Heggie and T. M. Heggie, “Search and rescue trends and the emergency medical service workload in utah’s national parks,” *Wilderness & environmental medicine*, vol. 19, no. 3, pp. 164–171, 2008.
- [4] J. Harper, “Special ops forces fuel demand for ultralight vehicles,” *National Defense*, vol. 100, no. 740, pp. 30–33, 2015.
- [5] M. Ciesa, S. Grigolato, and R. Cavalli, “Retrospective study on search and rescue operations in two prealps areas of italy,” *Journal of Outdoor Recreation and Tourism*, 2014.
- [6] H. Hsiao, J. Chang, and P. Simeonov, “Preventing emergency vehicle crashes: status and challenges of human factors issues,” *Human factors*, vol. 60, no. 7, pp. 1048–1072, 2018.
- [7] R. R. Murphy, J. Peschel, C. Arnett, and D. Martin, “Projected needs for robot-assisted chemical, biological, radiological, or nuclear (CBRN) incidents,” in *International Symposium on Safety, Security, and Rescue Robotics*. IEEE, 2012, pp. 1–4.

- [8] S. Keenan and J. C. Riesberg, “Prolonged field care: Beyond the “Golden Hour”,” *Wilderness & Environmental Medicine*, vol. 28, no. 2, Supplement, pp. S135–S139, 2017, tactical Combat Casualty Care: Transitioning Battlefield Lessons Learned to Other Austere Environments.
- [9] A. Ghobadpour, G. Monsalve, A. Cardenas, and H. Mousazadeh, “Off-road electric vehicles and autonomous robots in agricultural sector: Trends, challenges, and opportunities,” *Vehicles*, vol. 4, no. 3, pp. 843–864, 2022.
- [10] J. Ni, J. Hu, and C. Xiang, “A review for design and dynamics control of unmanned ground vehicle,” *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 235, no. 4, pp. 1084–1100, 2021.
- [11] M. Stone, J. G. Blich, and A. Rudolph, “Future technology vision for unmanned ground vehicles (ugvs),” in *Unmanned Ground Vehicle Technology VI*, vol. 5422. SPIE, 2004, pp. 101–111.
- [12] M. M. Kabir, J. R. Jim, and Z. Istenes, “Terrain detection and segmentation for autonomous vehicle navigation: A state-of-the-art systematic review,” *Information Fusion*, vol. 113, p. 102644, 2025.
- [13] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [14] J. A. Marshall, W. Sun, and A. L’Afflitto, “A survey of guidance, navigation, and control systems for autonomous multi-rotor small unmanned aerial systems,” *Annual Reviews in control*, vol. 52, pp. 390–427, 2021.
- [15] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

- [16] A. V. Goldberg and T. Radzik, “A heuristic improvement of the bellman-ford algorithm,” *Applied Mathematics Letters*, vol. 6, no. 3, pp. 3–6, 1993.
- [17] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
- [18] C. Lamini, S. Benhlima, and A. Elbekri, “Genetic algorithm based approach for autonomous mobile robot path planning,” *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.
- [19] L. Blasi, E. D’Amato, M. Mattei, and I. Notaro, “Path planning and real-time collision avoidance based on the essential visibility graph,” *Applied Sciences*, vol. 10, no. 16, p. 5613, 2020.
- [20] F. Aurenhammer, “Power diagrams: properties, algorithms and applications,” *SIAM Journal on Computing*, vol. 16, no. 1, pp. 78–96, 1987.
- [21] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.
- [22] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, “Path planning with modified a star algorithm for a mobile robot,” *Procedia engineering*, vol. 96, pp. 59–69, 2014.
- [23] J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *International Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.
- [24] J. Hossain, A.-Z. Faridee, N. Roy, A. Basak, and D. E. Asher, “Covernav: Cover following navigation planning in unstructured outdoor environment with deep reinforcement

- learning,” in *International Conference on Autonomic Computing and Self-Organizing Systems*. IEEE, 2023, pp. 127–132.
- [25] D. L. Page, A. F. Koschan, M. A. Abidi, and J. L. Overholt, “Ridge-valley path planning for 3D terrains,” in *International Conference on Robotics and Automation*. IEEE, 2006, pp. 119–124.
- [26] E. Birgersson, A. Howard, and G. S. Sukhatme, “Towards stealthy behaviors,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 2. IEEE, 2003, pp. 1703–1708.
- [27] L. Santos, N. Ferraz, F. N. dos Santos, J. Mendes, R. Morais, P. Costa, and R. Reis, “Path planning aware of soil compaction for steep slope vineyards,” in *International Conference on Autonomous Robot Systems and Competitions*. IEEE, 2018, pp. 250–255.
- [28] J. Jiang, Z. Han, J. Li, Y. Wang, J. Wang, and S. Xu, “Global path planning of UGVs in large-scale off-road environment based on improved A-star algorithm and quadratic programming,” in *Intelligent Vehicles Symposium*. IEEE, 2023, pp. 1–7.
- [29] C. Shen, S. Yu, B. I. Epureanu, and T. Ersal, “An efficient global trajectory planner for highly dynamical nonholonomic autonomous vehicles on 3D terrains,” *IEEE Transactions on Robotics*, 2023.
- [30] H. Wang, H. Zhang, K. Wang, C. Zhang, C. Yin, and X. Kang, “Off-road path planning based on improved ant colony algorithm,” *Wireless Personal Communications*, vol. 102, no. 2, pp. 1705–1721, 2018.
- [31] M. Sivaprakasam, S. Triest, W. Wang, P. Yin, and S. Scherer, “Improving off-road

- planning techniques with learned costs from physical interactions,” in *International Conference on Robotics and Automation*. IEEE, 2021, pp. 4844–4850.
- [32] S. Koenig, M. Likhachev, and D. Furcy, “Lifelong planning a*,” *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.
- [33] Blender online community, *Blender: A 3D modelling and rendering package*, Blender foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: <http://www.blender.org>
- [34] A. Stentz *et al.*, “The focussed D* algorithm for real-time replanning,” in *International Joint Conference on Artificial Intelligence*, vol. 95, 1995, pp. 1652–1659.
- [35] S. Koenig and M. Likhachev, “D* lite,” in *Eighteenth national conference on Artificial intelligence*, 2002, pp. 476–483.
- [36] T. Overbye and S. Saripalli, “Fast local planning and mapping in unknown off-road terrain,” in *International Conference on Robotics and Automation*. IEEE, 2020, pp. 5912–5918.
- [37] J. Dallas, Y. Weng, and T. Ersal, “Combined trajectory planning and tracking for autonomous vehicles on deformable terrains,” in *Dynamic Systems and Control Conference*, vol. 84270. American Society of Mechanical Engineers, 2020, p. V001T15A001.
- [38] K. Lynch, *Modern Robotics*. Cambridge University Press, 2017.
- [39] E. J. Haug, *Computer-Aided Kinematics and Dynamics of Mechanical Systems, Volume II: Modern Methods*, 1994.
- [40] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, “Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments,” *Journal of Field Robotics*, vol. 34, no. 5, pp. 940–984, 2017.

- [41] D. Kalaria, S. Sharma, S. Bhagat, H. Xue, and J. M. Dolan, “Wroom: An autonomous driving approach for off-road navigation,” *arXiv preprint arXiv:2404.08855*, 2024.
- [42] J. Gibson, B. Vlahov, D. Fan, P. Spieler, D. Pastor, A.-a. Agha-mohammadi, and E. A. Theodorou, “A multi-step dynamics modeling framework for autonomous driving in multiple environments,” in *International Conference on Robotics and Automation*. IEEE, 2023, pp. 7959–7965.
- [43] J. Liang, P. Gao, X. Xiao, A. J. Sathyamoorthy, M. Elnoor, M. C. Lin, and D. Manocha, “Mtg: Mapless trajectory generator with traversability coverage for outdoor navigation,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 2396–2402.
- [44] T. Löw, T. Bandyopadhyay, J. Williams, and P. V. Borges, “Prompt: Probabilistic motion primitives based trajectory planning.” in *Robotics: Science and Systems*, vol. 17, 2021.
- [45] B. Li and Y. Zhang, “Fast trajectory planning for off-road autonomous driving with a spatiotemporal tunnel and numerical optimal control approach,” in *International Conference on Advanced Robotics and Mechatronics*. IEEE, 2019, pp. 924–929.
- [46] Y. Wang, S. Li, W. Cheng, X. Cui, and B. Su, “Toward efficient trajectory planning based on deterministic sampling and optimization,” in *Chinese Automation Congress*. IEEE, 2020, pp. 1318–1323.
- [47] Y. Qi, B. He, R. Wang, L. Wang, and Y. Xu, “Hierarchical motion planning for autonomous vehicles in unstructured dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 496–503, 2022.
- [48] T. Lei, T. Sellers, C. Luo, D. W. Carruth, and Z. Bi, “Graph-based robot optimal path

- planning with bio-inspired algorithms,” *Biomimetic Intelligence and Robotics*, vol. 3, no. 3, p. 100119, 2023.
- [49] N. T. T. Le, E. Bray, K. M. B. Lee, and G. Best, “Adaptive trajectory library planner for fast outdoor robots,” in *Australasian Conference on Robotics and Automation*, 2023.
- [50] A. Clark and Z. Li, “Resilient trajectory planning in adversarial environments,” in *Conference on Decision and Control*. IEEE, 2019, pp. 3839–3846.
- [51] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, “Optimal trajectory generation for dynamic street scenarios in a frenet frame,” in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 987–993.
- [52] D. D. Fan, K. Otsu, Y. Kubo, A. Dixit, J. Burdick, and A.-A. Agha-Mohammadi, “Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation,” *arXiv preprint arXiv:2103.02828*, 2021.
- [53] T. Yang, G. Xiong, Y. Zhang, L. Yang, B. Tang, M. Wu, and J. Gong, “A practical planning framework and its implementation for autonomous navigation in off-road environment,” in *Intelligent Vehicles Symposium*. IEEE, 2019, pp. 1571–1576.
- [54] F. Tian, R. Zhou, Z. Li, L. Li, Y. Gao, D. Cao, and L. Chen, “Trajectory planning for autonomous mining trucks considering terrain constraints,” *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 4, pp. 772–786, 2021.
- [55] S. Hong and G. Park, “Trajectory optimization and robust tracking control for off-road autonomous vehicle,” *IEEE Access*, vol. 12, pp. 82 205–82 219, 2024.
- [56] N. Goulet and B. Ayalew, “Energy-optimal ground vehicle trajectory planning on deformable terrains,” *IFAC-PapersOnLine*, vol. 55, no. 27, pp. 196–201, 2022.

- [57] X. Gong, G. Tao, R. Qiu, Z. Zang, S. Zhang, and J. Gong, “Terrain-aware spatiotemporal trajectory planning for ground vehicles in off-road environment,” in *International Conference on Electrical Engineering, Control and Robotics*. IEEE, 2024, pp. 200–207.
- [58] M. Toscano-Moreno, A. Mandow, M. A. Martínez, and A. García-Cerezo, “DEM-AIA: Asymmetric inclination-aware trajectory planner for off-road vehicles with digital elevation models,” *Engineering Applications of Artificial Intelligence*, vol. 121, p. 105976, 2023.
- [59] J. Song and I. Sharf, “Rollover-free trajectory planning for reconfigurable vehicles under terrain uncertainty,” in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 4088–4094.
- [60] X. Cai, M. Everett, L. Sharma, P. R. Osteen, and J. P. How, “Probabilistic traversability model for risk-aware motion planning in off-road environments,” in *International Conference on Intelligent Robots and Systems*. IEEE, 2023, pp. 11 297–11 304.
- [61] H. Tian, B. Li, H. Huang, and L. Han, “Driving risk-averse motion planning in off-road environment,” *Expert Systems with Applications*, vol. 216, p. 119426, 2023.
- [62] L. Schumaker, *Spline Functions: Computational Methods*. Philadelphia, PA: SIAM, 2015.
- [63] F. Miller, A. Vandome, and J. McBrewster, *Frénet-Serret Formulas: Vector Calculus, Curve, Derivative, Euclidean Space, Kinematics, Darboux Frame, Differential Geometry of Curves, Affine Geometry of Curves*. New York, NY: Alphascript Publishing, 2010.
- [64] W. Wang, B. Jüttler, D. Zheng, and Y. Liu, “Computation of rotation minimizing frames,” *ACM Transactions on Graphics (TOG)*, vol. 27, no. 1, pp. 1–18, 2008.

- [65] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.
- [66] A. K. Singh, K. M. Krishna, and S. Saripalli, “Planning non-holonomic stable trajectories on uneven terrain through non-linear time scaling,” *Autonomous Robots*, vol. 40, pp. 1419–1440, 2016.
- [67] S. Yu, C. Shen, and T. Ersal, “Nonlinear model predictive planning and control for high-speed autonomous vehicles on 3D terrains,” *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 412–417, 2021.
- [68] F. Sabry, *K Nearest Neighbor Algorithm: Fundamentals and Applications*. One Billion Knowledgeable, 2023, vol. 28.
- [69] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [70] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto, “A high-performance software package for semidefinite programs: Sdpa 7,” *Tokyo, Japan*, 2010.
- [71] H. White, J. Mukherjee, A. L’Afflitto, and C. Sandu, “Terrain sensing techniques and algorithms: A review,” in *International Society for Terrain-Vehicle Systems Conference*, Montreal, Canada, Sep. 2021.
- [72] E. M. DuPont, R. G. Roberts, M. F. Selekwa, C. A. Moore, and E. G. Collins, “Online terrain classification for mobile robots,” in *ASME International Mechanical Engineering Congress and Exposition*, vol. 42169, 2005, pp. 1643–1648.

- [73] S. Khaleghian and S. Taheri, "Terrain classification using intelligent tire," *Journal of Terramechanics*, vol. 71, pp. 15–24, 2017.
- [74] P. Filitchkin and K. Byl, "Feature-based terrain classification for littledog," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1387–1392.
- [75] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena, and M. Hutter, "Where should I walk? Predicting terrain properties from images via self-supervised learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1509–1516, 2019.
- [76] J. Libby and A. J. Stentz, "Using sound to classify vehicle-terrain interactions in outdoor environments," in *International Conference on Robotics and Automation*. IEEE, 2012, pp. 3559–3566.
- [77] G. Reina, R. Galati, and A. Milella, "All-terrain estimation for mobile robots in precision agriculture," in *International Conference on Industrial Technology*. IEEE, 2018, pp. 63–68.
- [78] T. Braun, H. Bitsch, and K. Berns, "Visual terrain traversability estimation using a combined slope/elevation model," in *Advances in Artificial Intelligence: 31st Annual German Conference on AI*. Kaiserslautern, Germany: Springer, 2008, pp. 177–184.
- [79] F. Schilling, X. Chen, J. Folkesson, and P. Jensfelt, "Geometric and visual terrain classification for autonomous mobile navigation," in *International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 2678–2684.
- [80] F. Yang, G. Lin, and W. Zhang, "Terrain classification for terrain parameter estimation based on a dynamic testing system," *Sensor Review*, vol. 35, no. 4, pp. 329–339, 2015.

- [81] B. Sebastian and P. Ben-Tzvi, “Support vector machine based real-time terrain estimation for tracked robots,” *Mechatronics*, vol. 62, p. 102260, 2019.
- [82] C. A. Brooks and K. Iagnemma, “Self-supervised terrain classification for planetary surface exploration rovers,” *Journal of Field Robotics*, vol. 29, no. 3, pp. 445–468, 2012.
- [83] R. O. Chavez-Garcia and O. Aycard, “Multiple sensor fusion and classification for moving object detection and tracking,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 525–534, 2015.
- [84] Intel Corporation, “Intel® realsense™ tracking camera T265 and depth camera D435: Tracking and depth,” Intel Corporation, Tech. Rep., 2019, white Paper, Revision 001. [Online]. Available: https://www.intelrealsense.com/wp-content/uploads/2019/11/Intel_RealSense_Tracking_and_Depth_Whitepaper_rev001.pdf
- [85] P. Schmidt, J. Scaife, M. Harville, S. Liman, and A. Ahmed, “Intel® realsense™ tracking camera T265 and intel® RealSense™ depth camera D435-tracking and depth,” *Real Sense*, 2019.
- [86] T. Moore and D. Stouch, “A generalized extended Kalman filter implementation for the robot operating system,” in *Intelligent Autonomous Systems*. Springer, 2016, pp. 335–348.
- [87] S. M. Isa, S. A. Shukor, N. Rahim, I. Maarof, Z. Yahya, A. Zakaria, A. Abdullah, and R. Wong, “Point cloud data segmentation using ransac and localization,” in *Materials Science and Engineering*, vol. 705, no. 1. IOP Publishing, 2019, p. 012004.
- [88] P. Narksri, E. Takeuchi, Y. Ninomiya, Y. Morales, N. Akai, and N. Kawaguchi, “A slope-robust cascaded ground segmentation in 3D point cloud for autonomous vehicles,” in

- International Conference on intelligent transportation systems.* IEEE, 2018, pp. 497–504.
- [89] N. T. Sangary and N. K. Nikolova, “Line-of-sight approximation to the equivalence principle,” *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 7, pp. 1890–1897, 2004.
- [90] O. Khatib, “The potential field approach and operational space formulation in robot control,” in *Adaptive and Learning Systems: Theory and Applications*. Springer, 1986, pp. 367–377.
- [91] S. Thrun, “Probabilistic robotics,” *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
- [92] W. Y. Chien, “Stereo-camera occupancy grid mapping,” Master’s thesis, Penn State University, 2020.
- [93] T.-N. Nguyen, B. Michaelis, A. Al-Hamadi, M. Tornow, and M.-M. Meinecke, “Stereo-camera-based urban environment perception using occupancy grid and object tracking,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 154–165, 2011.
- [94] D. Pandya, V. K. Rana, and T. M. V. Suryanarayana, “Inter-comparison and assessment of digital elevation models for hydrological applications in the Upper Mahi River Basin,” *Applied Geomatics*, pp. 1–24, 2024.
- [95] S. D. Naranjo, C. Sandu, S. Taheri, and S. Taheri, “Experimental testing of an off-road instrumented tire on soft soil,” *Journal of Terramechanics*, vol. 56, pp. 119–137, 2014.
- [96] H. Fan, H. Su, and L. J. Guibas, “A point set generation network for 3D object recon-

- struction from a single image,” in *Conference on computer vision and pattern recognition*, 2017, pp. 605–613.
- [97] C. Liu, C. Zhou, W. Cao, F. Li, and P. Jia, “A novel design and implementation of autonomous robotic car based on ros in indoor scenario,” *Robotics*, vol. 9, no. 1, p. 19, 2020.
- [98] L. Champeny-Bares, S. Coppersmith, and K. Dowling, “The terregator mobile robot,” Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Technical Report, 1991.