DEVELOPMENT OF A SOFTWARE TOOL TO IMPLEMENT A VISION-BASED NAVIGATION METHOD IN LOW-TEXTURED ENVIRONMENTS

Andriz Vegas Lopez

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

DEVELOPMENT OF A SOFTWARE TOOL TO IMPLEMENT A VISION-BASED NAVIGATION METHOD IN LOW-TEXTURED ENVIRONMENTS

Andriz Vegas Lopez

Trabajo de grado para optar por el título de Ingeniero Electrónico

Director
Eval Bladimir Bacca Cortes, Ph.D.

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

AGRADECIMIENTOS

Agradezco a mi familia por su entendimiento, paciencia y apoyo constante a través de todas las dificultades, posibilitando la dedicación sobre este trabajo. Agradezco también a Eneas, quien me acompañó fielmente durante las vigilias en el desarrollo del mismo.

Mi profundo agradecimiento al Dr.Bacca-Cortes, Profesor de la Escuela de Ingeniería Eléctrica y Electrónica (EIEE) de la Universidad del Valle, por su valiosa dirección, orientación y enseñanzas a través de todas las etapas de este trabajo. Agradezco igualmente a los profesores de la EIEE de la Universidad del Valle por sus enseñanzas y por enfatizar constantemente en la importancia de la dedicación en este trabajo.

Contents

List of Tables **2**

List of Annexes **2**

Glossary **4**

Abstract **5**

1 Introduction **6**
1.1 Problem Statement . . . 7
1.2 Goals . . . 7
1.3 Proposed Solution . . . 8
1.4 Document Structure . . . 8

2 Related works and Background **9**
2.1 Related works . . . 9
2.2 Background . . . 12
2.2.1 Computer Vision Processing . . . 13
2.2.2 Field Robotics . . . 16
2.2.3 State Estimation and Sensor Fusion . . . 20
2.2.4 Landscape contour detection method . . . 22
2.3 Final Remarks . . . 23

3 Application Development: A Software Tool for Implementing Vision-Based Navigation in Low-Texture Environments **25**
3.1 Introduction and Implementation of the Software Tool for Vision-Based Navigation in Low-Texture Environments . . . 25
3.1.1 Functional and Non-Functional Requirements . . . 25
3.1.2 User Stories (see Annex A) . . . 27
3.1.3 Conceptual Diagram . . . 28
3.1.4 Class Diagram . . . 29
3.2 Implementation . . . 30
3.2.1 Configuration Module . . . 30
3.2.2 Execution Module . . . 36
3.2.3 RTAB-Map as an Independent Evaluation Baseline . . . 45
3.2.4 Report Module . . . 47
3.3 Conclusions . . . 47

4 Results **49**
4.1 Integration Testing . . . 49
4.2 Quantitative Testing . . . 53
4.2.1 Data Acquisition . . . 53
4.2.2 Scale Factor Empirical Validation . . . 60
4.3 Field Testing . . . 62
4.4 Conclusions . . . 70

5 Conclusion and Future Work **72**
5.1 Future Work . . . 73

6 Annexes **77**
6.1 Annex A: User Stories . . . 77
6.2 Annex B: Literature review backward-forward analysis . . . 78

List of Tables

2.1 Comparison of Background Studies . . . 10
3.1 Skyline GUI Configuration Parameters and Default Values . . . 32
4.1 Summary of Field Testing Results . . . 71

List of Annexes

Annex A. User Stories . . . 77
Annex B. Literature review backward-forward analysis . . . 78

List of Figures

1.1 Natural skyline as reference for GNSS-denied navigation . . . 6
1.2 Landscapes variability . . . 7
1.3 System architecture for vision-based localization . . . 8
2.1 Distribution of selected literature . . . 9
2.2 Thresholding for sky segmentation . . . 13
2.3 Comparison of four color spaces . . . 14
2.4 CLAHE enhancement comparison . . . 14
2.5 Morphological operations on binary mask . . . 15
2.6 SIFT feature detection . . . 15
2.7 ORB feature detection . . . 16
2.8 Monocular depth predictions . . . 17
2.9 Commercial sensors for robotic perception . . . 17
2.10 Indoor and outdoor SLAM pipelines . . . 18
2.11 RTAB-Map visual SLAM reconstruction . . . 18
2.12 ICP alignment process . . . 20
2.13 Pose-graph representation . . . 21
2.14 EKF recursive cycle . . . 21
2.15 Edge detection for landscape contours . . . 23
2.16 Skyline contour extraction . . . 23
3.1 GUI interface overview . . . 25
3.2 Conceptual diagram of navigation tool . . . 28
3.3 UML component diagram . . . 29
3.4 System architecture module interaction . . . 30
3.5 Configuration Module diagram . . . 30
3.6 Configuration Module interface correspondence . . . 31
3.7 Dice coefficient for multi-plane skylines . . . 34
3.8 Low texture skyline evaluation . . . 35
3.9 Cloudiness gradient evaluation . . . 35
3.10 Cloud shade variation evaluation . . . 35
3.11 Skyline segmentation evaluation tools . . . 36
3.12 Execution module workflow . . . 36
3.13 ICP registration in pixel space . . . 37
3.14 ICP pipeline execution and logging . . . 38
3.15 Pixel-to-metric projection (t=1-2) . . . 39
3.16 Sequential alignment (intermediate) . . . 39
3.17 Alignment sequence final timesteps . . . 40
3.18 EKF-based sensor fusion evolution . . . 41
3.19 Localization process interface . . . 43
3.20 Localization pipeline visualization . . . 43
3.21 Trajectory generation pipeline . . . 44
3.22 Pose-graph vs GPS ground-truth . . . 45
3.23 RTAB-Map offline execution . . . 46
3.24 Report Module workflow . . . 47
4.1 Configuration Stage testing . . . 50

4.2 Execution Stage testing . . . 52
4.3 Reporting Stage testing . . . 53
4.4 Dataset acquisition at Universidad del Valle . . . 54
4.5 GNSS and skyline trajectory comparison . . . 55
4.6 High-resolution trajectory view . . . 56
4.7 ICP vs GNSS heading comparison . . . 57
4.8 ICP-GNSS heading discrepancy analysis . . . 57
4.9 Trial 13 evaluation . . . 58
4.10 Trial 48 evaluation . . . 58
4.11 Trial 111 evaluation . . . 59
4.12 ICP iterations per transform . . . 60
4.13 ICP fitness scores . . . 61
4.14 Scale factor evolution . . . 61
4.15 FT0 panoramic images . . . 62
4.16 FT0 field test setup . . . 62
4.17 RTAB-Map results . . . 63
4.18 FT0 pose-graph scale comparison . . . 64
4.19 Odometry drift vs optimized result . . . 64
4.20 RANSAC inlier count . . . 65
4.21 FT0 offline localization . . . 65
4.22 FT0 error analysis . . . 66
4.23 FT0 RMSE analysis . . . 67
4.24 Skyline localization visualization . . . 67
4.25 FT1 panoramic images . . . 68
4.26 FT1 trajectory comparison . . . 68
4.27 FT1 3D grid and correspondence . . . 69
4.28 FT1 error analysis . . . 70
4.29 FT1 RMSE analysis . . . 71
4.30 FT1 pose-graph scale factors . . . 72
4.31 FT1 RANSAC inliers . . . 73
4.32 FT1 offline localization . . . 73
4.33 Skyline localization final trajectory . . . 74
6.1 Project repository structure . . . 77
6.2 User stories by epics . . . 77
6.3 Literature backward-forward analysis . . . 78

Glossary

CLAHE: Contrast Limited Adaptive Histogram Equalization. Image enhancement technique that improves local contrast while limiting noise amplification.

DEM: Digital Elevation Model. A 3D representation of terrain surface created from elevation data.

EKF: Extended Kalman Filter. A nonlinear extension of the Kalman filter used for state estimation in robotics.

ENU: East-North-Up. A local tangent plane coordinate system used in navigation.

GNSS: Global Navigation Satellite System. Satellite-based positioning systems including GPS.

GPS: Global Positioning System.

GUI: Graphical User Interface. A visual interface for user interaction with software systems.

HSV: Hue-Saturation-Value. A color space model that describes colors in terms of their shade, saturation, and brightness.

ICP: Iterative Closest Point. An algorithm for the alignment of two point clouds by minimizing the distance between corresponding points.

LiDAR: Light Detection and Ranging. A remote sensing method using laser light to measure distances.

MiDaS: Monocular Depth Estimation model developed for depth prediction from single images.

ORB: Oriented FAST and Rotated BRIEF. A fast feature detector and descriptor used in computer vision.

RANSAC: Random Sample Consensus. An iterative method for estimating parameters of a mathematical model from data containing outliers.

RGB: Red-Green-Blue. The standard additive color model used in digital imaging.

RMSE: Root Mean Square Error. A measure of the differences between predicted and observed values.

ROS: Robot Operating System. A flexible framework for writing robot software.

RTAB-Map: Real-Time Appearance-Based Mapping. A RGB-D, stereo, and lidar SLAM approach based on an incremental appearance-based loop closure detector.

RViz: Robot Visualization. A 3D visualization tool for ROS.

SE(3): Special Euclidean group in 3D. The group of rigid body transformations (rotation and translation) in 3D space.

SIFT: Scale-Invariant Feature Transform. A feature detection algorithm in computer vision to detect and describe local features in images.

SLAM: Simultaneous Localization and Mapping. The computational problem of constructing a map of an unknown environment while simultaneously keeping track of an agent's location.

UML: Unified Modeling Language. A standardized modeling language for software engineering.

WGS84: World Geodetic System 1984. The standard coordinate reference system used by GPS.

RESUMEN

Se desarrolló una herramienta de software para implementar un método de navegación basado en visión en entornos exteriores de baja textura donde los sistemas GNSS pueden ser poco confiables. El sistema utiliza el perfil del horizonte natural (skyline) como característica visual principal para la localización. La arquitectura integra tres módulos: configuración, ejecución y generación de reportes. El módulo de ejecución implementa detección de skyline mediante segmentación semántica, alineación de nubes de puntos mediante el algoritmo ICP (Iterative Closest Point), y fusión con datos GNSS a través de un Filtro de Kalman Extendido (EKF). Como método de comparación y mediante uso de estimación de profundidad monocular con MiDaS, se empleó RTAB-Map para obtener una trayectoria optimizada como grafo de poses. Se realizaron pruebas de campo en la Universidad del Valle, donde en la prueba de campo FT0 (trayectoria de 25 m, aproximadamente 2800 poses), se obtuvo un RMSE de posición de 0.369 m y RMSE de orientación de 12.41°. En FT1 (trayectoria de 40 m, aproximadamente 7700 poses y condiciones de iluminación más desafiantes), se obtuvo un RMSE de posición de 1.259 m y RMSE de orientación de 30.47°. El software fue desarrollado siguiendo metodología Scrum con arquitectura modular basada en ROS, interfaz gráfica para experimentación interactiva y visualización. Los resultados validan la viabilidad de la localización basada en skyline como complemento a sistemas tradicionales de navegación en escenarios de campo específicos.

Palabras clave: Navegación basada en visión, Detección de skyline, Localización sin GNSS, ICP, Filtro de Kalman Extendido, SLAM.

ABSTRACT

A software tool was developed to implement a vision-based navigation method in low-texture outdoor environments where GNSS systems may be unreliable. The system uses the natural horizon profile (skyline) as the primary visual feature for localization. The architecture integrates three modules: Configuration, Execution, and Report Generation. The Execution Module implements skyline detection through semantic segmentation, point cloud alignment via the Iterative Closest Point (ICP) algorithm, and fusion with GNSS data through an Extended Kalman Filter (EKF). As a comparison method and using monocular depth estimation with MiDaS, RTAB-Map was used to obtain an optimized trajectory as a pose graph. Field tests were conducted at Universidad del Valle. In FT0 (25 m trajectory, approximately 2,800 poses), a position RMSE of 0.369 m and an orientation RMSE of 12.41° were obtained. In FT1 (40 m trajectory, approximately 7,700 poses and more challenging illumination conditions), a position RMSE of 1.259 m and an orientation RMSE of 30.47° were obtained. The software was developed following Scrum methodology with a modular ROS-based architecture, a graphical interface for interactive experimentation and visualization. The results validate the feasibility of skyline-based localization as a complement to traditional navigation systems in specific field scenarios.

Keywords: Vision-based navigation, Skyline detection, GNSS-denied localization, ICP, Extended Kalman Filter, SLAM.

1 Introduction



Figure 1.1: Using the natural skyline as a reference for navigation in different GNSS-denied environments

The range of sensors that currently enable robots to estimate their location includes Global Navigation Satellite Systems (GNSS)/GPS, and active sensors such as radars and Lidars [1], among others. Although these active sensors are popular today, their nature is unfavorable for tasks involving intelligence, recognition, and surveillance because they are detectable. Moreover, they can generate interference with other sensors [2]. Additionally, GNSS signals exhibit inadequate behavior or are non-existent in complex environments as the ones in Figure 1.1, making computer vision an emerging alternative [3] that offers the possibility of performing location estimations autonomously using only cameras as sensors. This method is particularly relevant in low-texture environments such as deserts, open skies, or extraplanetary environments where feature extraction is challenging due to the uniformity of the environment [4].

In the aforementioned environments, algorithms based on visual features such as SIFT, ORB, and Shi-Tomas are not entirely reliable. Other algorithms like Stereo-PLSLAM, SuperLine, and Structure-SLAM work in low-texture environments; however, the environment must be structured (e.g., buildings, walls, etc.), which involves recognizing lines and planes in the environment. In low-texture and unstructured environments, these methods can still fail [5]. This is why, in unstructured and low-texture environments where there is a skyline (horizon profile), which is the dividing line between the sky and the ground, estimating the location is possible despite the difficulty imposed by the low texture of the environment [4].

Environments such as glacial terrains, mountainous regions, and maritime landscapes are particularly difficult due to their lack of reliable visual features, frequent occlusions, and extreme illumination variability, as shown in Figure 1.2. These conditions often render traditional localization techniques ineffective or unusable. Conventional robotic navigation systems are either infeasible or unreliable in low-textured environments like those encountered in polar or mountainous regions [6]. As a result, research has increasingly turned toward vision-based techniques that leverage natural visual cues—such as skyline or horizon boundaries—to achieve orientation and localization when other methods fail [7], [8]. This is achieved by exploiting the contrast between sky and terrain to estimate orientation even in the absence of ground features [9], and it is especially useful in glacial and mountainous terrains where terrain silhouettes can serve as natural landmarks. Several algorithms have been developed to extract these features, using dynamic programming over edge maps to trace skyline contours efficiently or through supervised learning approaches that utilize dense classifier score images (DCSI) to detect horizons even when traditional edge cues are insufficient.

Classification-based techniques have further improved robustness by filtering non-horizon features such as clouds or mountain ridges. These methods classify edge pixels using support vector machines (SVM) [10] or convolutional neural networks (CNN) [11], which enhance skyline continuity and reduce false positives [8]. Hybrid models that combines edge and classification-based methods have demonstrated superior performance in occluded or weather-degraded scenes [9].

Vision-augmented odometry systems, particularly those tailored for glacial environments, have demonstrated effective localization through monocular visual odometry and terrain-based visual cues [6]. These systems use sparse visual features, where preprocessing methods such as contrast-limited adaptive histogram equalization (CLAHE) have been shown to enhance feature detection and tracking under low-light and low-contrast conditions, both in glacial terrain and in vision-based navigation more broadly [12].

Visual odometry systems tailored for glacial environments have demonstrated effective localization using monocular cameras and natural visual cues such as horizon profiles [6]. These systems rely on sparse visual features where preprocessing methods such as contrast-limited adaptive histogram equalization



Figure 1.2: Landscapes variability

(CLAHE) have been shown to enhance feature detection and tracking under low-contrast conditions in glacial terrain [6] and under low-light conditions in vision-based navigation more broadly [12].

The objective of this work is to develop a software application that uses computer vision methods to determine location in these low-texture environments through the use of the horizon profile.

1.1 Problem Statement

Through global navigation satellite systems (GNSS), it is possible to estimate the location of a receiver or device on the ground using GPS. This type of application is widely used in modern life, such as in military, and maritime contexts [1], among others. However, when GNSS signals are unavailable or not sufficiently precise, location estimation can be strategically advantageous by analyzing data provided by various sensors, which implies a vision-based solution through the use of cameras [1]. Currently, various solutions are employed for the problem of autonomous location estimation (without using GNSS) such as inertial navigation, odometry, or visual odometry [4].

However, these solutions are part of "dead-reckoning" techniques and exhibit cumulative error proportional to the distance traveled. Other approaches, such as terrain beacons, allow estimating the position of the robotic agent in relation to them, but this involves modifying the navigation environment, which is not feasible in many situations due to the terrain's geography.

Moreover, using GNSS as another approach to determine position is not always a reliable alternative for several reasons: one of them is the delay of the GPS signal and its inoperability in extraplanetary applications. Another reason is that the update rate of GNSS signals is very low compared to the high update frequency required for autonomous vehicle applications [13]. Hence, the need for a long-distance localization method that uses passive perception methods (e.g., monocular or stereo vision). For an image taken outdoors, it is possible to define the dividing line between the sky and the ground (horizon profile) that does not necessarily correspond to the actual horizon line. On the other hand, a low-texture environment is characterized by the absence of distinctive elements, which presents challenges for location estimation [14]. This type of environment can be analyzed by extracting the horizon profile and using current algorithms that work in low texture such as SIFT, ORB, Shi-Tomas, Stereo-PLSLAM, SuperLine, and Structure-SLAM with the goal of performing location estimation. Based on the above, the following engineering question is proposed: **How can a software tool be developed to perform location estimation in low-texture environments using computer vision?**

1.2 Goals

The overall goal is to develop a software application that implements a computer vision-based method for determining the localization of autonomous vehicles in low-texture environments, and the following specific objectives enabled the achievement of this general objective:

1. Definition of the functional and non-functional requirements of the application by conducting a literature review of algorithms for vision-based navigation in low-texture environments.

2. Design of the software application that implements a computer vision-based localization method in low-texture environments.

3. Implementation of the software application using a computer vision-based localization method in low-texture environments.

4. Conduct integration and functional tests to verify the application's functionality.

1.3 Proposed Solution

The proposed solution for developing a software application that enables vision-based navigation in low-texture environments is structured into two main phases: Application Design, Implementation, and Results (see Sections 3 and 4). Figure 1.3 presents the overall architecture of the proposed vision-based navigation system for low-texture environments. The process begins with input sources, either from a pre-recorded dataset or rosbags, which are then managed by an Input & Output (I&O) module responsible for initial preprocessing. This includes noise reduction. Once preprocessed, the data is forwarded to the Vision-Based Navigation Module, where skyline detection is performed as a primary visual cue for navigation. The output of this module feeds into a SLAM-based Localization System, enabling simultaneous estimation of both position and orientation. The final processed information is then managed and synchronized (green segment) for visualization, allowing real-time display through a graphical user interface (GUI). Color-coded indicators represent different preprocessing calls: red for artifact removal and green for real-time data synchronization.

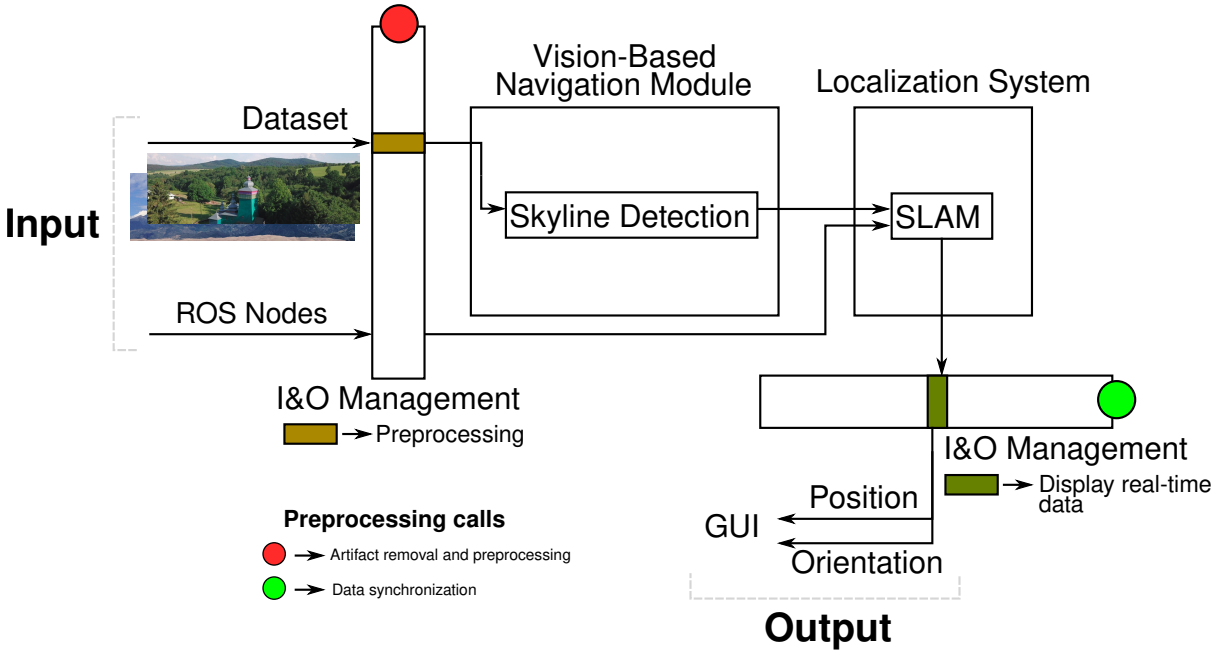


Figure 1.3: System architecture for vision-based localization: input data from datasets or ROS nodes undergoes preprocessing before skyline detection and SLAM-based localization. The output, which is position and orientation is synchronized and displayed in real time via a GUI

1.4 Document Structure

The development is carried out through several sections, which include the problem statement and the need for an alternative localization system; the related works and background, which reviews previous works, technologies, and algorithms used in the selected works. Then, the design, development, and implementation of the application are described, and the results are presented for integration, quantitative evaluations, and field testing.

2 Related works and Background

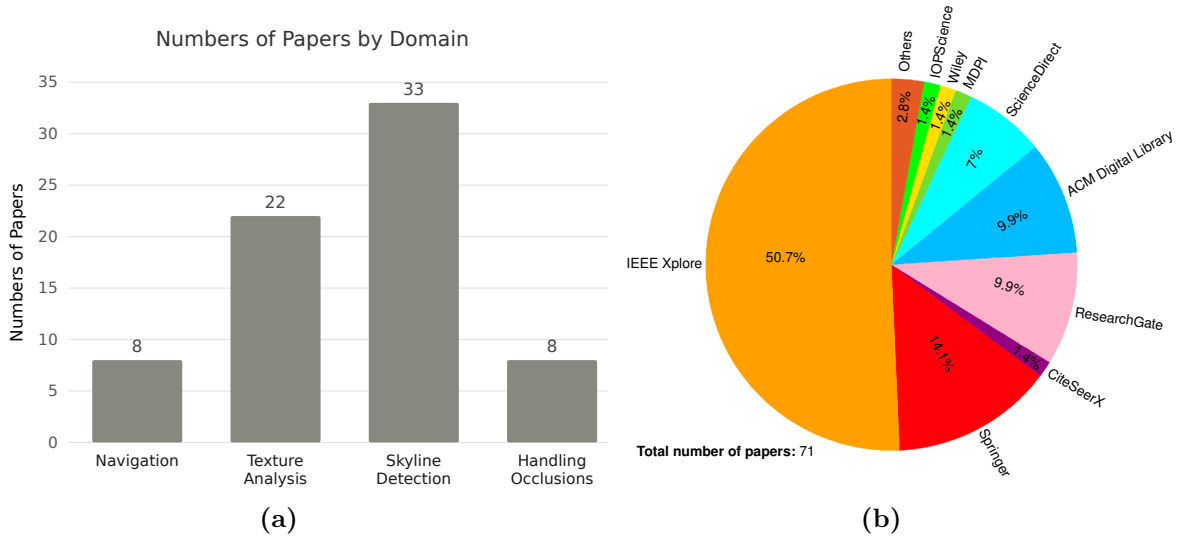


Figure 2.1: Distribution of selected literature: (a) focus areas, (b) source databases

It was necessary to conduct a comprehensive literature review to investigate how related works in the field of vision-based navigation and localization algorithms achieve their objectives and to understand the current state of the art. For this purpose, a total of 71 relevant articles were selected for their focus on navigation based on the visual perception of robotic agents in low-texture environments, where the detection of distinctive features becomes more challenging. The majority of these articles were obtained from the IEEE Xplore database, which accounted for 36 publications (50.7% of the total selected papers), as shown in Figure 2.1.

After reviewing these studies, four main areas of focus emerged: navigation, skyline detection, texture analysis, and handling occlusions. The following citation mapping tables illustrate the application of search techniques as outlined in Appendix B. The following figure presents a detailed distribution of the selected articles by database, as well as the identified four main areas of focus.

2.1 Related works

Relevant works emerged where the navigation of the robotic agent is developed through visual perception, whether in stereo or monocular configuration and its subsequent processing. A comparison of the conditions, systems, and singularities of each work is synthesized in Table 2.1. This table was structured considering the lighting conditions in the consulted works and which metrics were used to evaluate the results.
Although not all works used a robotic agent for testing, the table's structuring took this into account and analyzed the constituent elements of the works not only at the hardware and software levels but also the algorithms that allowed correlating a scheme of obtained images with a determined position. Additionally, given that the nature of the work involves the use of computer vision, the table structure noted which type of feature extraction algorithm was used and whether each system tolerated occlusions.

A series of criteria were established to compare the evaluated works as background. Key criteria such as type of lighting (day, night, or variable lighting conditions) have a direct impact on the performance of visual sensors. The operating environment is classified into outdoor and indoor settings, given the implication of the appearance of the horizon profile. Another key criterion considered is whether the environment is structured or not, which influences the complexity of the navigation task. The robotic platforms indicate the mobility and context of the systems' application. Feature extraction is perhaps one of the most important evaluation criteria, with various algorithms (e.g., SIFT, LSD, among others) used to identify and track landmarks or features in the environment in the context of vision-based navigation.

Table 2.1: Comparison of Background Studies

Ref	Functioning					Methods		Dataset	Occlusion	Metric	Error
	Lighting	**Environment**	**Terrain**	**Sensors**	**Platform**	**Feature Ext.**	**Algorithm**				
[1]	Day	Outdoor	Unstructured	Mono Cam.	PER agents	Sem. Seg.	DeepLab v3+	DEM	Yes	RMSE	0.18°
[4]	Day	Outdoor	Unstructured	Stereo Cam.	Magellium's Rover	SIFT	Skyline Proj.	DEM	No	Param. Errors	4.0°
[3]	Day	Outdoor	Unstructured	Mono Cam.	No	CNN	MS-DeepLab	DEM	No	Dist. Euc.	12.02°
[15]	Night	Outdoor	Structured	TIR Cam.	QAV400 Quad-Rotor	REMODE	VINS-MONO	Motion-Cap. Arena	No	XYZ Drift	3.17°
[2]	Day	Indoor & Out.	Unstructured	Mono Cam.	Exabot & P3-AT	STAR & BRIEF	Map-and-Replay	Odometry	No	RMS Dist. Euc.	0.08°
[14]	Day	Indoor	Structured	RGB-D Cam.	No	LSD	VO System	TUM RGB-D	No	ATE	3.61°
[16]	*Low-Light*	*Outdoor*	*Unstructured*	*Hybrid Cam.*	*Virtual Lunar Rover*	*Edge-Recog. Alg. & SIFT*	*Path-Planner*	*No*	*Yes*	*Error Det.*	1.43°
[9]	Day (variable)	Outdoor	Unstructured	Mono Cam.	No	No	SDA	No	Yes	Process. Time	3.2 ms
[5]	Day	Indoor	Structured	RGB-D Cam.	No	LSD and RANSAC	Factor Graph Opt. (Bund. Adj.)	TUM RGB-D	Yes	ATE	0.74°
[17]	Variable	Outdoor	Structured	Mono Cam.	No	Edge Extractor	Azimuth Matching	SRTM, ASTER and Flickr	No	Mean Absolute Error	1.04°

Continued on next page

Table 2.1 – Continued from previous page

Ref	Functioning					Methods		Dataset	Occlusion	Metric	Error
	Lighting	**Environment**	**Terrain**	**Sensors**	**Platform**	**Feature Ext.**	**Algorithm**				
[18]	Variable	Outdoor	Unstructured	Mono Cam.	Rover robot agent	Canny Edge and Ramer-D-P.	Horizon Line Extraction	No	Yes	Rank-order Stats.	$< 2.5\%$ of image pixels
[7]	Variable	Outdoor	Unstructured	Mono Cam.	No	Sobel, Canny, Marr-Hildreth	Dynamic programming	No	Yes	Process. Time	0.21s per frame
[8]	Low-Light	Outdoor	Both	Mono Cam.	Rover robot agent	HOG, Sobel, Canny and Hough	Dynamic Programming, CNN, SVM	No	Yes	Abs. Vertical Pixel Error	SVM-Error: 1.01 pixels, CNN-Error: 0.75 pixels
[6]	Low-Light	Outdoor	Unstructured	Mono Cam.	SnoMote Mk2	SIFT, SURF, CLAHE	Weighted Heuristics	No	Yes (Robust)	Loc. Accuracy Error	4°

Reference datasets serve as comparison points to validate the accuracy of the systems, and algorithms such as DeepLab v3+, Skyline projection, and VO System are evaluated against these datasets. Metrics like root mean square error (RMSE), parameterized errors, absolute trajectory error (ATE), and e uclidean distance (Euc. Dist.) provide quantitative measures of system performance. Finally, the ability to tolerate occlusions is considered, as it has a definitive impact on the functioning of vision-based algorithms. These evaluation criteria were functional for comparing and analyzing the performance of different visual localization systems.

In [1], the datasets captured by robotic agents on Mars and Earth are in lighting conditions corresponding to daytime. A semantic prediction was performed by integrating semantic segmentation with DeepLab v3+, and from there, the skyline was detected. The location estimation was compared with a digital elevation model (DEM) that acted as ground truth, as it provided a 360°-FOV projection for each point in the model. Using Magellium's rover for a project in the field of space robotics, an alternative was developed in [4] to address the problem of loss of localization in environments where GPS is unavailable. The sequence of this work includes rendering a skyline from a digital elevation model as ground truth and then starting the construction of the algorithm by extracting the obtained skyline from an image. Next, rectification, skyline detection, and position estimation of the rover are applied. Additionally, in [7], a system consisting of two parts was proposed. In the input interface, feature extraction and plane detection were performed using the Line Segment Detector (LSD).

The output of this interface is the detected planes, which are then used for pose estimation through a graph-based factor. For testing this work, the TUM RGB-D dataset was used, with images having a resolution of 640 x 480. On the other hand, the method proposed in [2] based its navigation on visual perception using a commercial camera. It employed a hybrid approach for navigation called teach-and-replay, which combined two algorithms: path-following and feature-based. The work demonstrated that this composite method improves localization accuracy, simplifies the map learning process, and reduces the size of the map. Experiments conducted in this work were carried out in both indoor and outdoor environments.

Works like [3], where the GPS signal is denied, are interesting in how, from the moment the ground truth was constructed based on a digital elevation model (DEM), a structure for calculating Euclidean distance was defined with a region of interest on the obtained images by contrasting with the detected skyline. It is worth noting that in this work, a proprietary dataset was generated to avoid occlusion; this was ensured by taking images at a distance from the mountains and not nearby, as the latter increases the probability of occlusion by capturing objects like trees.

In the previously cited works, the lighting conditions correspond to daytime, making the work conducted in [8] particularly interesting. This study addresses the lighting problem by using a thermal monocular camera installed on a quadcopter in the context of a safe autonomous landing. The scenario for this work involved predefined routes for the quadcopter at an altitude between 2 and 3 meters. Autonomously, it had to evaluate a safe landing site using pixel intensity as a depth measurement for feature detection. In [9], the development was carried out in a virtual environment representing the moon, making the issue of low texture especially noteworthy in this work.

The vision-based navigation, both monocular and stereo, was achieved through their combination. This allowed for the generation of a 3D reconstruction of the scene, enabling route lanning estimation. Wheel-terrain models were considered for this work. In summary, the background analysis found that through varied environments and changing lighting conditions, important properties such as semantic segmentation, feature extraction, and horizon profile detection algorithms were evaluated using digital elevation models (DEM) as ground truth for precise localization without GPS. In contexts of variable lighting, thermal cameras enabled safe autonomous landing and optimized skyline detection with reduced computational load.

2.2 Background

Autonomous navigation in unstructured, low-texture environments—such as polar regions, deserts, maritime settings, and extraplanetary surfaces—poses significant challenges for traditional localization techniques. These environments often lack distinctive visual features, rendering conventional methods like GNSS-based localization and feature-based visual odometry (e.g., SIFT, ORB) less effective. Moreover, active sensors like LiDAR and radar, while useful in structured settings, can be unreliable or counter-

productive in such scenarios due to occlusion, interference, or stealth requirements [19].

Recent advancements in computer vision have highlighted the potential of natural environmental cues, particularly skylines or horizon profiles, for orientation and localization in challenging settings. These methods leverage the contrast between the sky and terrain to estimate orientation, even in the absence of ground features [8]. For instance, in environments like glacial terrains or planetary landscapes, the horizon profile serves as a reliable reference when other techniques fail [20]. Vision-augmented odometry systems, particularly those designed for glacial or planetary exploration, integrate monocular SLAM and terrain-based visual cues to achieve accurate localization in environments devoid of reliable GNSS signals. These systems utilize sparse landmarks and visual slope estimates to reconstruct terrain, enabling safe and robust navigation[21]. In light of these developments, the following foundational elements were employed in a highly interconnected manner.

2.2.1 Computer Vision Processing

Classical computer vision techniques—such as SIFT, SURF, ORB, edge detection, and thresholding—have long been foundational in enabling robust perception capabilities within robotic systems. These methods play a critical role in feature detection, obstacle avoidance, and environment mapping, particularly in scenarios where real-time performance and computational efficiency are needed. This section explores how these algorithms are employed and evaluated in robotic applications, with an emphasis on their operational advantages, limitations, and contributions to autonomous navigation.

Thresholding:



(a) **(b)**

Figure 2.2: Application of Otsu's thresholding for sky segmentation. The original RGB image (left) is processed to generate a binary mask (right), where white pixels represent the detected sky region, which allows the system to exclude the sky in later stages of image analysis.

As a foundational technique in computer vision that segments images into foreground and background regions, masking effect procued by thresholding enables robotic systems to isolate objects or features for further processing. In navigation-based applications, such as agricultural harvesting, autonomous vehicles, and exploration robots, thresholding is crucial for robustly identifying obstacles, terrain boundaries, and objects of interest despite environmental variability. For instance, in [22], adaptive thresholding is achieved by converting RGB images to the HSV color space, focusing on hue and saturation thresholds that are dynamically adjusted using reinforcement learning. This adaptive approach allows the robot to respond effectively to lighting changes and occlusions. This enables detection of key features (like fruits or terrain contours) that inform the robot's motion planning and navigation decisions. In this work, the application of this technique is reflected in the use of thresholding to identify the pixels that belongs to the sky, and thus, to exclude them in subsequent stages of processing, as shown in Figure 2.2.

Colors Space:
In vision-based navigation and robotic perception systems, color segmentation serves as a fundamental preprocessing step for object recognition and scene understanding. [23] highlights the critical role of selecting the appropriate color space to ensure robust performance in these applications. By evaluating RGB, HSV, HSL, and YUV color spaces, they demonstrate that HSV and HSL outperform RGB and YUV in terms of classification speed and accuracy. Their findings underscore that the choice of color space directly influences the reliability of object detection and segmentation, especially in dynamic and

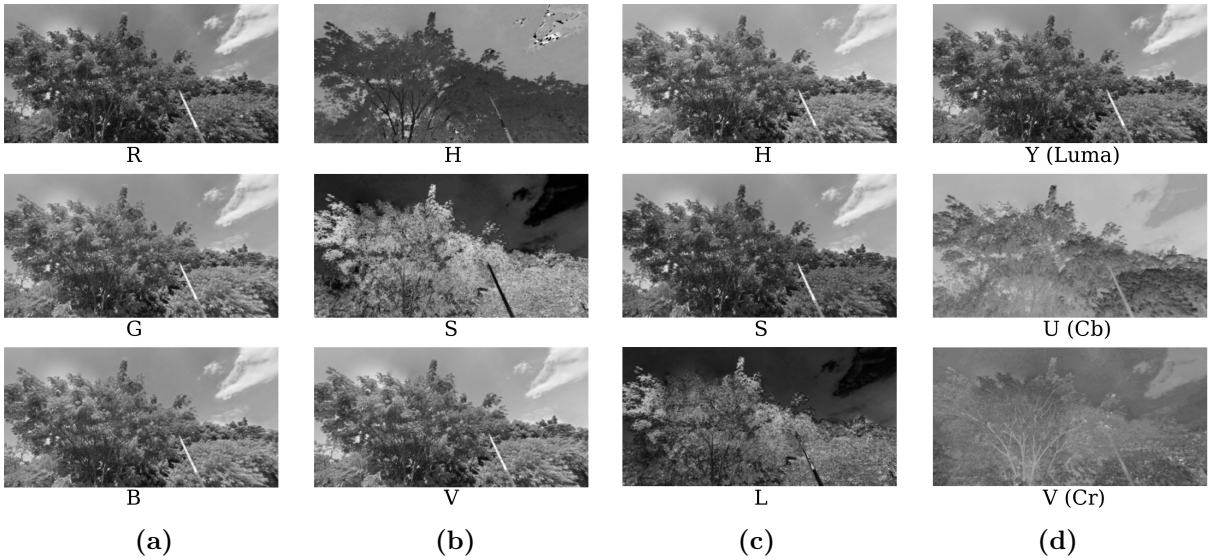


Figure 2.3: Comparison of image representations across four color spaces: (a): RGB, (b): HSV, (c): HSL, and (c): YUV. Each column corresponds to a different color space, and each row displays one of its individual channels. This visualization highlights how different color spaces separate information, such as brightness and chromaticity, to allow more effective segmentation in different lighting conditions. For this work, the enhanced distinction in HSV and HSL channels supports a more robust object detection in vision-based navigation tasks.

unstructured environments. These insights are relevant for this work, as accurate color-based segmentation enhances navigation capabilities and informs decision-making processes in low-textured or variable illumination scenarios, as shown in Figure 2.3. The research further validates the necessity of considering color space characteristics when designing vision-based navigation algorithms for autonomous systems operating in diverse real-world settings.

Contrast Limited Adaptive Histogram Equalization (CLAHE):



Figure 2.4: Comparison between the original image (left) and the enhanced result using Contrast Limited Adaptive Histogram Equalization (CLAHE) (right). CLAHE improves local contrast by adaptively equalizing small regions while limiting noise amplification. This enhancement reveals finer details in shaded areas and supports better feature extraction under challenging lighting conditions—crucial for navigation systems in unstructured outdoor environments.

Contrast Limited Adaptive Histogram Equalization (CLAHE) is an image enhancement technique that improves local contrast by adaptively equalizing the histogram of small regions within an image (see Figure 2.4. Unlike standard histogram equalization, CLAHE includes a contrast-limiting step that prevents the over-amplification of noise, making it especially valuable in low-light and high-dynamic-range scenarios. This localized approach to contrast enhancement has been widely adopted across various computer vision domains, including medical imaging, remote sensing, and, more recently, autonomous

vehicle perception. For instance, [24] demonstrated the effectiveness of CLAHE in improving the performance of YOLO-based models for road marking recognition at night, highlighting its practical value in vision-based navigation applications.

Morphological Operations:



(a) **(b)** **(c)** **(d)**

Figure 2.5: Application of morphological operations to a binary sky segmentation mask. From left to right: (a): closing, (b): opening, (c): erosion, and (d): dilation. These operations refine the segmented region by removing small artifacts (erosion), filling gaps (dilation), eliminating thin noise (opening), and closing small holes (closing), which enhances the structural consistency of the segmented foreground.

Morphological operations, as the ones illustrated in the Figure 2.5, are fundamental techniques in computer vision, essential for processing and analyzing the shapes and structures of objects within images. As highlighted by [25], these operations—including erosion, dilation, opening, and closing. They are particularly effective for noise removal, boundary smoothing, and feature extraction in both binary and grayscale images. Using a structuring element, which is a small matrix that interacts with the image, these operations takes into account each pixel's neighborhood to manipulate object boundaries and refine image details. For instance, erosion removes small, isolated noise and sharpens object boundaries by shrinking foreground regions, while dilation expands object boundaries and fills small holes or gaps. Opening and closing operations, which are combinations of erosion and dilation, further improve image quality by eliminating thin bridges or small gaps, respectively, without compromising the integrity of larger structures.

Scale-Invariant Feature Transform (SIFT):

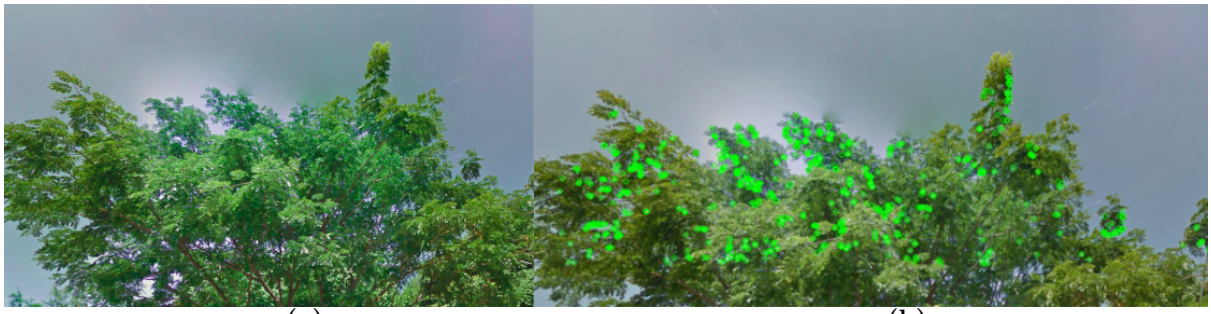


(a) **(b)**

Figure 2.6: SIFT feature detection and orientation assignment. (a) image shows the original scene, while (b) overlays the detected keypoints in the upper canopy using the SIFT algorithm. Circles represent the location and scale of each keypoint, and lines within the circles indicate the dominant gradient orientation. This orientation-based description enables rotation invariance and highlights the algorithm's capacity to identify stable, discriminative features in natural, texture-rich environments.

This algorithm, as detailed by [26], provides a foundational approach to scale and rotation invariant feature detection and description. SIFT locates keypoints through a Difference of Gaussians (DoG) applied in scale space, which allows precise identification of stable image features across multiple resolutions. Each keypoint receives a dominant orientation based on local gradient direction as shown in Figure 2.6, which ensures full rotation invariance. The descriptor captures the spatial distribution of gradients around the keypoint that offers high distinctiveness and robustness. Experimental results confirm the effectiveness of SIFT in object recognition, image matching, and scene reconstruction. Its resilience to changes in scale, rotation, occlusion, and illumination has established it as a standard in computer vision, particularly in robotics and vision-based navigation.

Oriented FAST and Rotated BRIEF (ORB):



(a) **(b)**

Figure 2.7: ORB feature detection and orientation-aware keypoint extraction. While (a) shows the original input, (b) displays the detected ORB keypoints overlaid in green. ORB builds upon the FAST detector and augments it with orientation estimation and an efficient binary descriptor (rBRIEF). The resulting features offer robustness to rotation and illumination variations.

This algorithm, proposed by [27], addresses the need for efficient and robust feature detection and description in computer vision applications. ORB builds upon the FAST keypoint detector by introducing a fast and accurate orientation component, which enhances the rotation invariance of detected features. Furthermore, ORB incorporates a modified version of the BRIEF descriptor, called rBRIEF, which learns a set of uncorrelated binary tests to improve matching performance while keeps high computational efficiency. As shown in Figure 2.7, ORB achieves matching performance comparable to SIFT, yet is significantly faster, making it ideal for real-time applications in robotics and mobile devices. The algorithm's rotation invariance, noise resistance, and low computational burden underscore its relevance in vision-based navigation and mapping, where robust and efficient feature matching is essential.

Monocular Depth Estimation:
The work [28] address the central limitation of monocular depth estimation: the lack of sufficiently large and diverse datasets that cover the wide range of environments encountered in real imagery. Existing datasets rely on sensors with restricted operating conditions, incomplete reconstructions, or limited depth accuracy, which introduces strong biases and prevents broad generalization across scenes. The authors identify that no single dataset offers enough diversity to train a model that maintains robust performance outside its native domain, which leads them to explore a strategy that unifies heterogeneous sources of depth information. Their work seeks a practical solution that permits a model to operate reliably across diverse environments without requiring specialized tuning for each dataset.

Previous work, such as the method illustrated in Fig. 2.8, proposes a depth-estimation framework that can be trained on many different datasets, even when they use incompatible depth units or camera settings. As a result, the model generalizes well to new, unseen datasets (zero-shot transfer) and better approximates real-world performance than models trained on a single dataset.

This work uses MiDaS due to its ability to operate across domains that differ from typical single curated datasets. The model introduced by [28] suits this requirement through training on heterogeneous data sources and through performance that remains stable on unseen scenarios. The chosen MiDaS variant includes a ResNeXt-101 encoder pretrained on weakly supervised and ImageNet data, scale and shift-invariant depth objectives, and a Pareto-optimal dataset, which mixes a strategy that unifies diverse sources such as ReDWeb, MegaDepth, WSVD, DIML Indoor and 3D Movies. This allows for a consistent depth prediction across varied outdoor environments and support this thesis by improving the reliability of skyline-based localization and ICP-based mapping without requiring environment-specific re-train.

2.2.2 Field Robotics

Whether navigating rugged terrains, exploring planetary surfaces, or conducting missions in remote or hazardous areas, field robots rely on robust perception, mobility, and control capabilities to operate independently. This section delves into the foundational principles, key methodologies, and ongoing

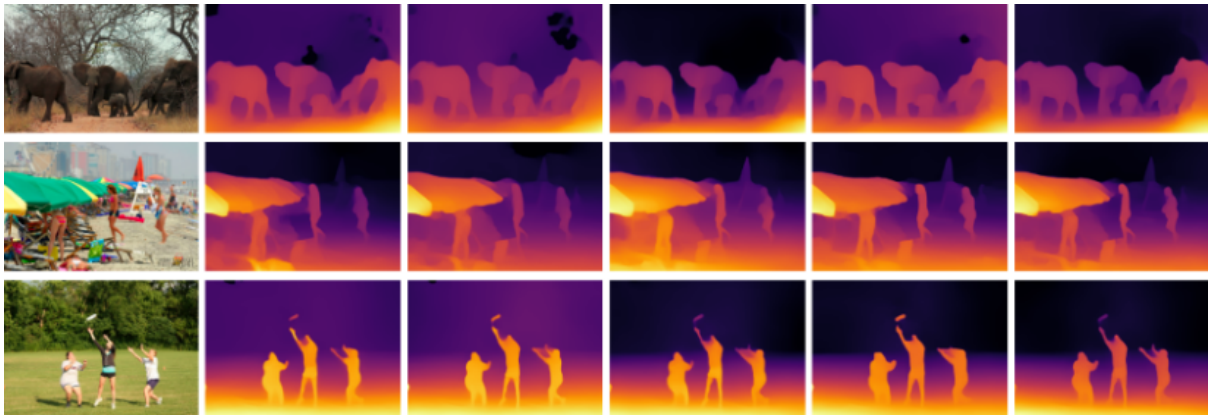


Figure 2.8: Qualitative depth predictions. Input RGB images (left) and the corresponding monocular depth estimates produced by the method proposed in Towards Robust Monocular Depth Estimation. The examples show consistent scene structure and strong zero-shot generalization across varied outdoor settings. (Source: [28])

challenges in field robotics, which illustrates how these systems bridge the gap between laboratory research and practical applications across diverse settings.

Sensing and Perception:



Figure 2.9: Representative commercial sensors used in robotic perception for outdoor environments. From left to right and top to bottom: 2D and 3D LiDARs (e.g., Velodyne, Hokuyo), omnidirectional and multi-camera systems (e.g., Occam, NCTech), and RGB-D or panoramic imaging sensors. These devices support critical perception tasks such as semantic segmentation, object detection, and terrain mapping in unstructured natural environments.

Sensing and perception are crucial for robots in outdoor natural environments. As summarized in [29], forests pose challenges such as changing weather, dense vegetation, and moving obstacles, all under real-time constraints. Using sensors like RGB-D cameras, LiDAR, and multispectral systems, together with perception methods for scene understanding, robots can detect obstacles, interpret terrain, and navigate safely. These capabilities strongly influence a field robot's autonomy and reliability. Fig. 2.9 illustrates some of these representative sensing and perception tools.

Localization and Mapping:
Localization and Mapping remains a major challenge in field robotics, particularly in unstructured or GPS-denied environments where external positioning is unreliable, as illustrated in Figure 2.10. Prior work such as [32] demonstrates how stereo vision and Rao–Blackwellized particle filtering achieve real-time localization without expensive LiDAR sensors, while [33] extends SLAM to multi-robot settings through decentralized Fast-SLAM techniques for data association, alignment, and map-merging. Together, these studies highlight a clear evolution from single-robot SLAM toward collaborative and dynamic environments. Within this progression, RTAB-Map stands out as a practical, real-time visual SLAM framework that integrates loop-closure detection and pose-graph optimization, making it well suited for the visual-based localization tasks addressed in this work.

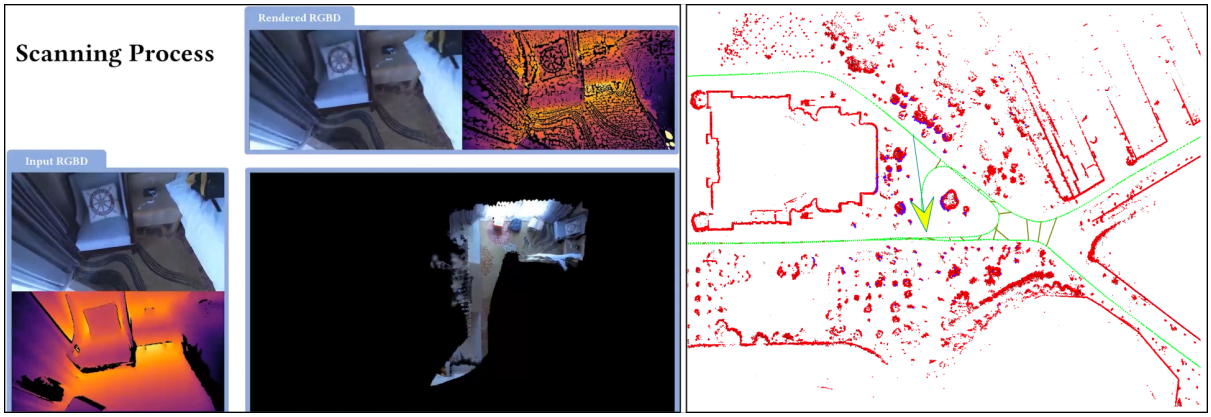


Figure 2.10: (a): Illustration of an indoor SLAM pipeline using RGB-D input for 3D real-time reconstruction and localization taken from [30]. The system processes raw RGB and depth data to generate a dense map of the environment, which demonstrates efficient onboard sensing in structured but GPS-denied settings. In (b), the work [31] shows outdoor SLAM results from a decentralized, multi-robot system operating in an unstructured environment. The approach integrates data from multiple agents using a modified Fast-SLAM framework, supporting map-merging and navigation across dynamically changing terrain.

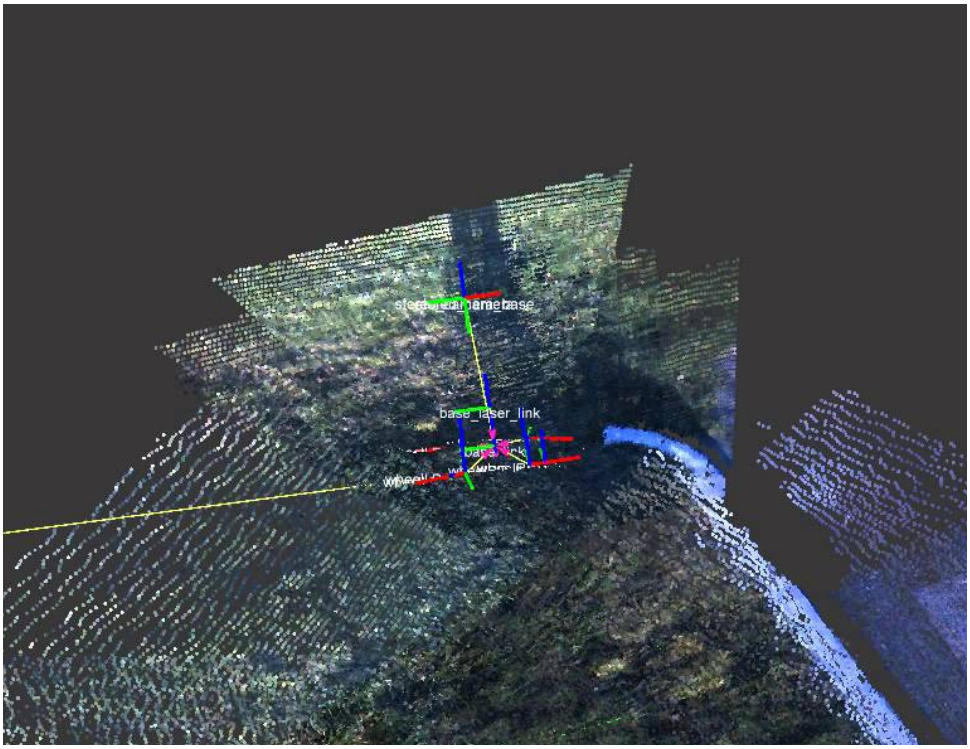


Figure 2.11: RTAB-Map visual SLAM reconstruction shows the estimated robot pose and the surrounding dense point-cloud map. The camera, base, and laser reference frames appear as colored axes, while the accumulated visual data produces a textured 3D representation of the environment. This output reflects the ability of RTAB-Map to maintain a consistent pose-graph and generate a spatially coherent map from monocular inputs in outdoor field conditions

In this work, skyline-based ICP odometry is used as a lightweight, high-frequency motion estimator that tracks small pose increments directly from visual contours. However, ICP alone accumulates drift rapidly, especially in outdoor environments with appearance changes, weak texture, or partial skyline

occlusions. For this reason, it is not integrated into RTAB-Map as a front-end; instead, it is run as a separate module and its pose estimates are directly compared against GNSS ground truth to quantify translational and angular errors. In RTAB-Map, the inlier ratio of the skyline alignment is used as a quality indicator, and errors are analyzed when this ratio drops below a threshold for robust localization. Independently, in this work RTAB-Map is executed to obtain a globally consistent trajectory and map, and its trajectory is also evaluated with respect to GNSS as a second baseline.

Visual SLAM can map and localize a robot in GPS-denied environments using only camera data. In this work, we use RTAB-Map, which maintains a pose graph, detects loop closures, and refines the trajectory through graph optimization. As shown in Figure 2.11, it builds a dense point cloud and keeps the robot frames aligned with the estimated path. This makes RTAB-Map a strong geometric baseline for evaluating our ICP-based odometry.

Iterative Closing Point (ICP) Algorithm:
The Iterative Closest Point (ICP) algorithm is a technique for rigid point-cloud registration, originally formulated to align two geometric datasets by iteratively refining a transformation until convergence. As described in [34], the core idea is to repeatedly establish correspondences between points in the source cloud and their nearest neighbors in the target cloud, then compute the rigid-body transformation (rotation and translation) that minimizes the Euclidean alignment error. Each iteration consists of two steps: computing nearest-neighbor matches, and solving for the optimal transformation commonly using methods such as quaternions. The algorithm then updates the estimated pose and repeats the process until the change in error falls below a threshold or the maximum iterations are reached. Because this iterative refinement directly minimizes the squared point-to-point distance, ICP can yield very accurate alignments when its assumptions are satisfied.

At each iteration, ICP estimates the rigid motion that best aligns one skyline frame to the next, refining the solution through a rotation and translation that jointly minimize the correspondence error. This rotation is the key element that reorients the source frame toward the target, while the translation corrects its relative displacement. Formally, such motions are expressed as homogeneous matrices in the Lie group $SE(3)$, which unifies rotation and translation in a single operator and provides a consistent algebraic framework for composition, inversion, and incremental updates. The planar instance of this representation, relevant to skyline odometry, is given in Eq. 1.

$$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

From the homogeneous transformation, the motion quantities relevant to ICP arise directly from its rotation and translation blocks. The upper left submatrix encodes the in-plane rotation, which aligns one skyline frame to the next, while the upper right column provides the horizontal displacement. Because skyline odometry is inherently planar, the full $SE(3)$ transform collapses to an $SE(2)$ action: all z-related terms reduce to identity elements, and only the rotation $R(\theta)$ and the translation vector $\mathbf{t} = [t_x\ t_y]^\top$ are relevant for the scope of this work. These two components form the minimal parametrization of the ICP update, and their extraction is simply the decomposition shown in Eq. 2.

$$R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \tag{2}$$

Once we keep only the planar motion, the rotation matrix R corresponds to a single rotation angle θ, which gives the change in heading between two skyline frames. We compute this angle as $\theta = \text{atan2}(r_{21}, r_{11})$; it tells us how much the camera has to turn so that the two silhouettes line up. In our ICP odometry, this yaw increment sets the direction of the unit heading vector from one step to the next, while the translation $\mathbf{t} = [t_x\ t_y]^\top$ gives the 2D position shift that happens at the same time. Together, θ and $\mathbf{t}$ define the small motion estimated at each ICP iteration, which we then chain to build not just an estimated orientation plot and trajectory, but also to compute the corresponding errors.

In summary, the decomposition of the homogeneous transformation into its planar rotation and translation components offers a compact and geometrically meaningful description of the incremental motion

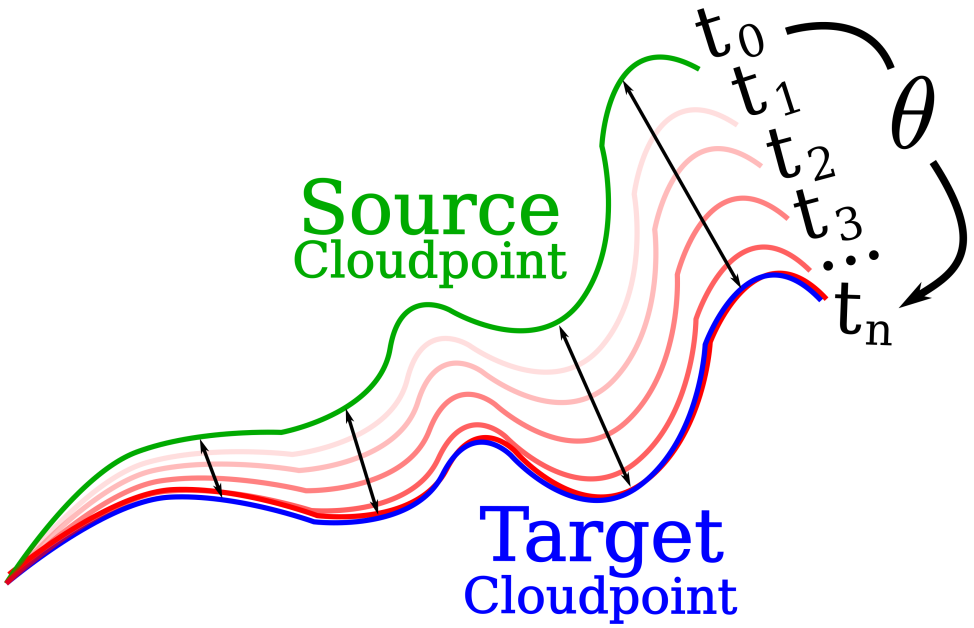


Figure 2.12: Iterative Closest Point (ICP) alignment process: the source cloud (green) is incrementally registered to the target cloud (blue) through a sequence of rigid transformations $t_0, t_1, ..., t_n$ and rotation θ, to progressively reduce the distance between corresponding points

recovered by ICP. Each iteration provides a refined estimate of the angle change θ and the associated displacement t which together reorient and reposition the source skyline so that it progressively approaches the target configuration. These updates matches the mechanics of the algorithm: through repeated correspondence estimation and rigid-motion optimization, the source point set is gradually deformed until it aligns with the target. This convergence process is characterized in Fig. 2.12. In this work, this accumulation enables both the sequential heading chain and the reconstruction of the global skyline trajectory.

Pose-graph representation with RTAB-Map:
In our work, RTAB-Map builds and continuously optimizes the same type of pose-graph illustrated in Fig. 2.13. Each node $X_i = [x_i, y_i, \theta_i]$ represents the camera pose at a given time, and each directed edge c_{ij} encodes a relative motion constraint between two poses, typically obtained from visual odometry or loop-closure detection. As new images are acquired, RTAB-Map adds a new node and an odometry edge to the graph, and when it detects that the robot has revisited a previously seen location, it introduces additional loop-closure edges. Periodically, the system runs a graph-optimization step that adjusts all node poses so that they best satisfy all these constraints, which reduces drift and produces a globally consistent estimate of the trajectory.

In the context of skyline odometry, assigning a location to a node is both natural and theoretically justified: every image defines a unique camera viewpoint on the environment. The ICP alignment between consecutive skyline silhouettes then becomes a relative pose estimate, exactly matching the role of an edge in the pose-graph. Each such edge contributes a rigid transformation (θ, t_x, t_y) that captures how the viewpoint must change for the current skyline to align with the next. When chained together, these edges generate an incremental trajectory; when embedded into a pose-graph, they form the structure required for later global optimization, drift correction, or loop-closure integration. This design mirrors established pipeline architectures in SLAM, where image-to-image or scan-to-scan registration steps feed directly into the graph structure that reconstructs the global motion.

2.2.3 State Estimation and Sensor Fusion

Autonomous robots must estimate their position and orientation in a global frame, a task known as localization, which is difficult to solve with a single sensor. Visual odometry methods, such as the Skyline ICP used in this work, provide smooth, continuous trajectories but their error grows over time (drift), while

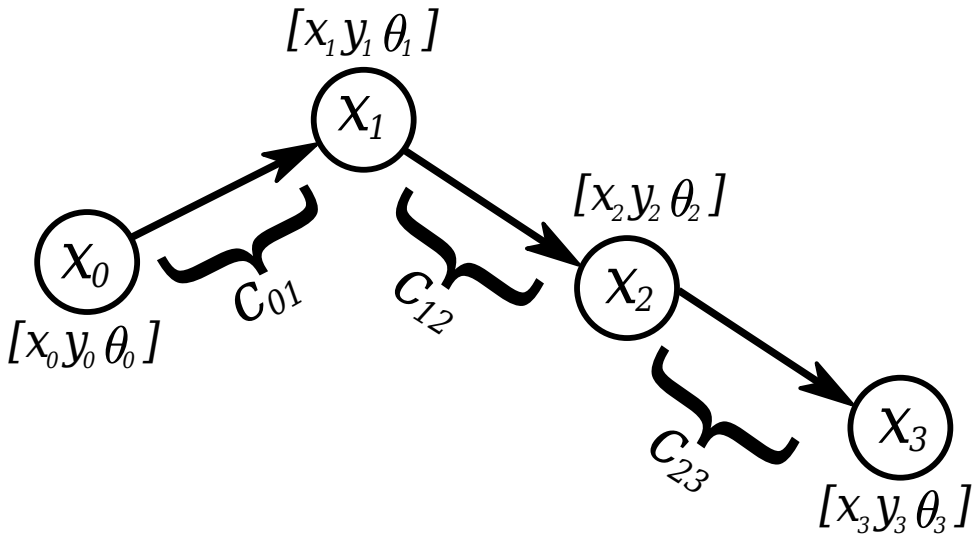


Figure 2.13: Pose-graph representation of the estimated trajectory. Each node X_i denotes the robot pose $[x_i, y_i, \theta_i]$, while directed edges c_{ij} encode motion constraints between consecutive poses. Together, these nodes and edges define the graph-based trajectory

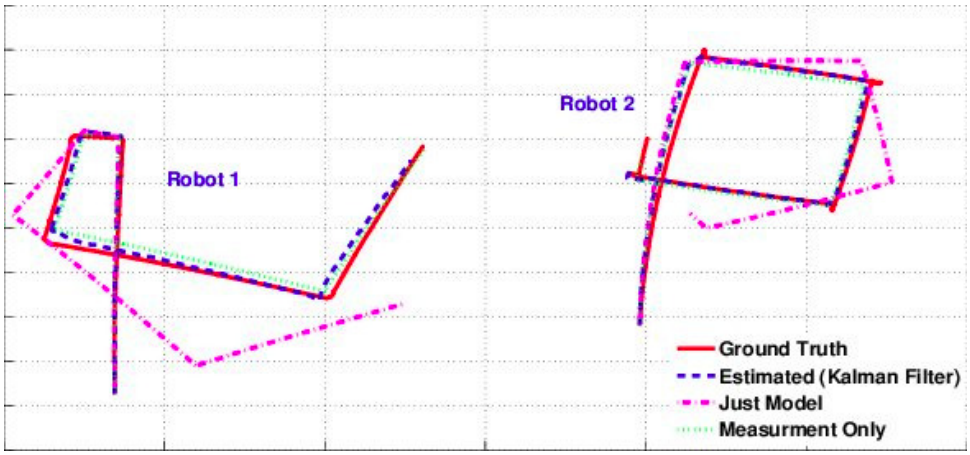


Figure 2.14: Geometric interpretation of the Extended Kalman Filter (EKF) recursive cycle. The figure illustrates the uncertainty propagation during the *Prediction Step* (green ellipse), where the state covariance $P_{k|k-1}$ expands due to process noise. The *Correction Step* (red ellipse) fuses the measurement z_k to produce the final state estimate $\hat{x}_{k|k}$, resulting in a reduced uncertainty covariance $P_{k|k}$. Adapted from [35]

absolute systems like GNSS offer bounded error yet tend to be noisy, affected by multipath, and updated at low rates or the contraints of the environment. To compensate for these limitations, multiple sensors are combined using probabilistic fusion, most commonly within the Kalman Filter family. A Kalman Filter is a recursive estimator that fuses noisy measurements to minimize the mean squared error of the state, under the assumption of Gaussian noise; however, mobile robot motion is non-linear because it depends on trigonometric functions of the heading. The Extended Kalman Filter (EKF) addresses this by linearizing the motion and measurement models around the current estimate, which makes it suitable for robot localization.

As illustrated in Fig. 2.14, the performance of the Extended Kalman Filter (EKF) as a sensor fusion engine is governed by the recursive interaction between the system process model and the measurement model. This interaction is mathematically defined by two distinct steps: the Time Update (Prediction) and the Measurement Update (Correction). The behavior of the filter during these steps is strictly determined by the stochastic modeling of the process noise covariance matrix ($\mathbf{Q}$) and the measurement noise covariance matrix ($\mathbf{R}$).

In the prediction step, the filter projects the current state estimate forward in time using the visual odometry derived from the Skyline-ICP algorithm. The a priori state estimate $\hat{\mathbf{x}}_{k|k-1}$ and the a priori error covariance $\mathbf{P}_{k|k-1}$ are computed as follows:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{3}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{4}$$

Where $\mathbf{F}_k$ is the Jacobian of the motion model f with respect to the state.

Equation (4) incorporates the **Process Noise Covariance** $\mathbf{Q}_k$. This matrix models the uncertainty accumulation inherent in the visual odometry process, which corresponds to ICP registration errors in the context of this work. Accordingly, $\mathbf{Q}_k$ is modeled as a diagonal matrix:

$$\mathbf{Q}_k = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{5}$$

When an external measurement signal is received, the filter corrects the predicted state. The influence of this correction is determined by the Kalman Gain ($\mathbf{K}_k$), which balances the uncertainty of the prediction against the uncertainty of the measurement:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{6}$$

Here, $\mathbf{R}_k$ represents the **Measurement Noise Covariance**. It models the expected error of the absolute positioning sensor (e.g., GNSS):

$$\mathbf{R}_k = \begin{bmatrix} \sigma_{meas_x}^2 & 0 \\ 0 & \sigma_{meas_y}^2 \end{bmatrix} \tag{7}$$

In this proposed implementation, σ_{meas} is set to **0.6 m**. The state update is then performed:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})) \tag{8}$$

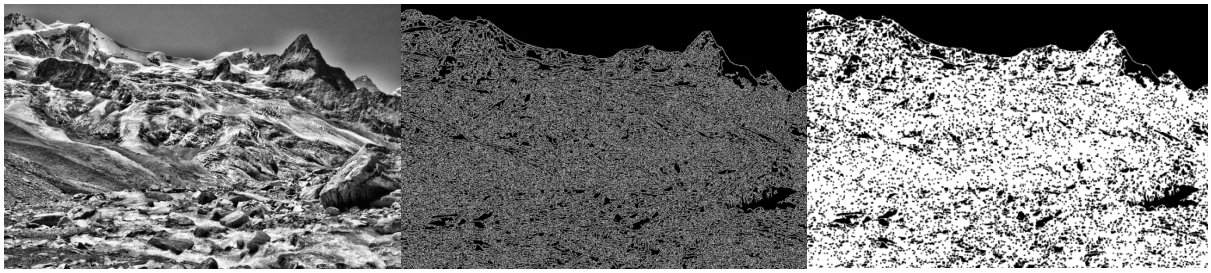
In this work, the tuning strategy employs a configuration where the modeled process noise is lower than the measurement noise ($\sigma_{process} < \sigma_{measurement}$).

Specifically in this work, setting $\sigma_{xy} = 0.4$ and $\sigma_{meas} = 0.6$, makes the system prioritize the local smoothness and structural consistency of the visual odometry over the absolute sensor data. This specific ratio results in a lower Kalman Gain $\mathbf{K}_k$, which causes the filter to treat instantaneous measurement fluctuations as noise. In summary: the fused trajectory retains the high-fidelity maneuvers captured by the Skyline-ICP algorithm while only gradually correcting towards the absolute coordinates to limit long-term drift. This acts as a low-pass filter on the absolute position data.

2.2.4 Landscape contour detection method

The Landscape Contour Detection rests over foundational techniques such as edge detection and contour representation, which are based on the task of identifying and delineating the boundaries of natural features within a landscape. After applied preprocessing steps, these techniques enable systems to interpret, localize, navigate, and interact with complex terrains. Below is an overview of such foundational principles.

Edge Detection:
In this work, edge detection plays one of the preprocessing steps for skyline delineation. The developed software tool integrates Canny and Sobel operators directly into the Graphical User Interface (GUI). Users can select and apply either method depending on the dataset and terrain characteristics. The

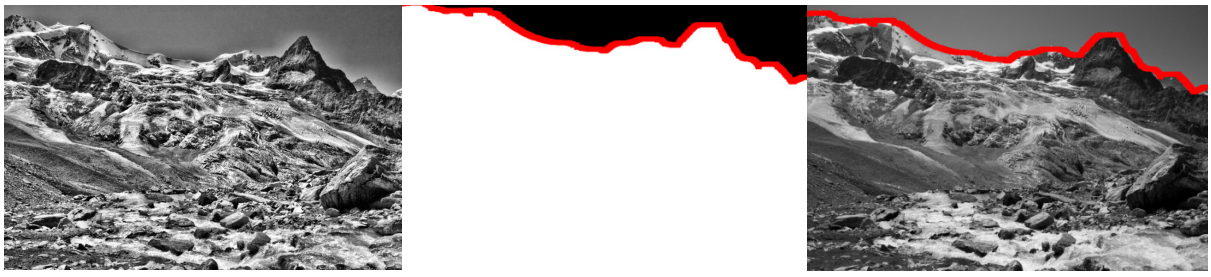


(a) **(b)** **(c)**

Figure 2.15: Edge detection applied to landscape contour analysis. The sequence shows (a) Preprocessed image, (b) Canny edge output, and (c) Sobel edge output.

Canny detector offers robustness in the identification of continuous and fine-grained edges while noise is reduced, which is helpful in low-textured environments where the skyline may not appear directly. The Sobel operator emphasizes gradient magnitude and direction, which provides complementary results. The implementation follows the system architecture and computes both methods and applies the one chosen in the GUI dropdown. The inclusion of these methods in the software design aligns with findings in the literature, where edge detection stands as a fundamental tool for contour delineation in complex terrains. For example, [36] employed the Canny method together with terrain analysis from Digital Elevation Models (DEMs) to refine terrace delineation, which achieved high accuracy in mountain topographies.

Contour Representation: In the literature, contour representation enables amn interpretation of natural features in complex terrains. For example, [37] proposed a vision-based geo-localization framework that matched skyline contours from panoramic images to DEM-derived references, which provided reliable positioning in mountainous environments. Contour representation in this work forms a critical step after edge detection, as it transforms raw edge responses into structured skyline candidates. The software tool applies morphological operations (closing, opening, dilation, erosion) and connected component analysis to refine binary masks and remove irrelevant blobs. The system isolates potential skyline contours through candidate filtering and reconstructs them as continuous red-only lines over the original image. This preserves the structural consistency of the skyline, rejects most of the noise and discontinuities, and prepares the contours for evaluation with algorithms such as dynamic programming. These steps are embedded directly into the GUI, and the software provides an interactive and practical platform for skyline extraction in low-texture navigation scenarios such as the sky, as shown in Fig. 2.16.



(a) **(b)** **(c)**

Figure 2.16: Skyline contour extraction and overlay for geo-localization. The sequence shows (a) preprocessed image, (b) binary skyline mask, and (c) contour detection using Multi-stage Graph Construction. This approach supports accurate positioning in GNSS-denied mountainous environments by matching visual skylines to DEM-derived references

2.3 Final Remarks

The background analysis reveals that vision-based navigation in low-texture environments requires systems to overcome the limitations of GNSS and traditional feature-based methods by exploiting scene-level

visual cues and adaptive perception strategies. These environments often lack strong visual features, making classic approaches like SIFT or ORB unreliable. To address this, recent systems integrate a range of computer vision techniques, such as thresholding for sky segmentation, adaptive histogram equalization for contrast enhancement, and color space transformations to improve robustness under varying illumination.

Edge detection and morphological operations contribute to terrain structure extraction, while descriptors like SIFT and ORB provide resilient keypoint matching when features are available. These low-level processes feed into higher-level methods such as skyline-based localization and DEM-based contour matching, which offer robust alternatives in GNSS-denied contexts. When fused with inertial or pseudo-LiDAR data, and organized through SLAM, these visual cues support consistent localization and mapping. Thus, the proposed software tool is designed to integrate these components: thresholding, contrast enhancement, color space selection, skyline detection, and multi-sensor fusion into a unified vision-based localization pipeline optimized for low-texture environments. This integration reflects the core properties highlighted in the background: robustness to environmental variability, low dependency on high-contrast features, and adaptability.

Finally, the software development process adopted the Scrum framework to organize the project into iterative sprints. Each sprint focused on the incremental construction and validation of a specific module (Configuration, Execution, or Report). Further details on the development workflow are reflected in the modular structure presented in chapter 3.1.

3 Application Development: A Software Tool for Implementing Vision-Based Navigation in Low-Texture Environments

3.1 Introduction and Implementation of the Software Tool for Vision-Based Navigation in Low-Texture Environments

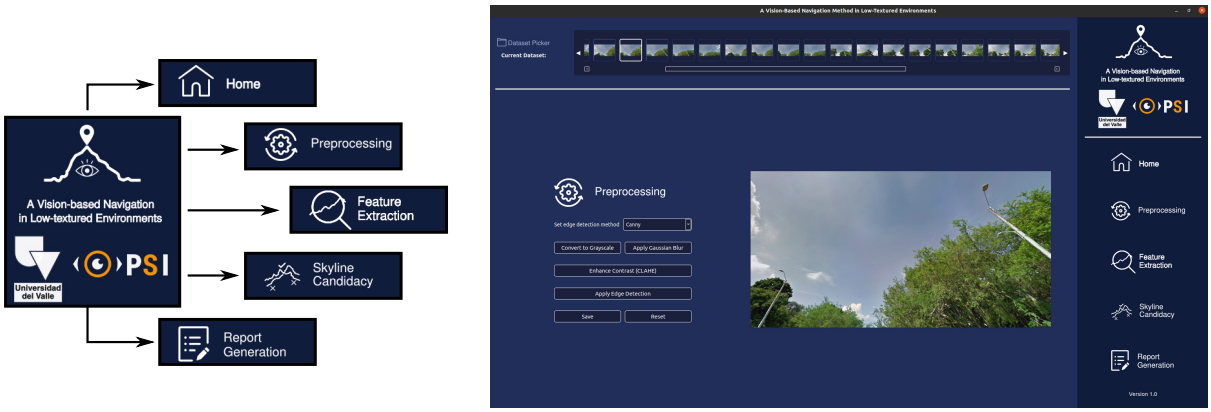


Figure 3.1: The interface allowed users to select datasets and access the main system modules

The software tool was developed to address the limitations of conventional localization in GNSS-denied, low-texture environments. This subsection introduces the implementation strategy, which integrates computer vision techniques, such as preprocessing, edge detection, contour segmentation, and skyline extraction, into a unified framework. The tool was conceived with modularity in mind, as shown in Fig. 3.1, which includes different acquisition sources, including pre-recorded imagery and rosbags. Its design emphasizes both usability, through a Graphical User Interface (GUI) that allows intuitive interaction with preprocessing and feature extraction parameters, features extracted, candidate skyline validation, and classification. In this way, the implementation bridges the gap between theoretical algorithms discussed in the background section and a practical, user-oriented application capable of support the localization tasks in unstructured obtained images.

The implementation of the software tool required a definition of both functional and non-functional requirements. These requirements establish what the system must accomplish, as well as the technical constraints under which it must operate. The development process was designed with a tradeoff between goals and practical performance, which serves as the basis for the subsequent implementation.

3.1.1 Functional and Non-Functional Requirements

The software tool effectively supports vision-based navigation in low-texture environments as long as it was necessary to establish a clear set of functional and non-functional requirements. The functional requirements define the specific features and behaviors the system must provide, such as dataset handling, preprocessing operations, skyline detection, and reporting capabilities. The non-functional requirements, on the other hand, describe the quality attributes and technical constraints that guide implementation, which includes performance, usability, operating environment, and library dependencies.

Functional Requirements

- **1.0**: It should be possible to choose the type of dataset (based on textures) where the software will be tested. (*Category: GUI*)
- **1.1**: It should be possible to adjust graphical parameters such as brightness, contrast, and exposure directly from the interface. (*Category: GUI*)
- **1.2**: It should allow switching between different processing views, such as raw images, preprocessed versions, and detected skylines. (*Category: GUI*)
- **1.3**: When detected, the candidate lines should be highlighted as an overlay on the original image. (*Category: GUI*)

- **1.4**: There should be a visual indicator when the skyline detection is in progress. (*Category: GUI*)
- **1.5**: If something goes wrong (e.g., incorrect file format, missing metadata), the system should show clear error messages. (*Category: GUI*)
- **1.6**: It should provide options to export processed images and results with skyline overlays. (*Category: GUI*)
- **2.0**: The system must support multiple image formats (JPEG, PNG, TIFF). (*Category: Data acquisition*)
- **2.1**: If metadata files (CSV with image locations) are uploaded, the system should automatically associate them with the images. (*Category: Data acquisition*)
- **2.2**: If an uploaded file has missing or corrupted data, the user should be notified before processing starts, and the system must allow user to upload a file again. (*Category: Data acquisition*)
- **3.0**: The system must automatically resize all images to a standard resolution (720x480) to ensure uniform processing. (*Category: Preprocessing*)
- **3.1**: For noise reduction, the system would apply Gaussian Blur, Median Filtering, and Bilateral Filtering to remove unwanted high-frequency noise. (*Category: Preprocessing*)
- **3.2**: The system needs to convert images to extract the blue channel and convert to grayscale using adaptive thresholding, so the contrast between the sky and the terrain is enhanced. (*Category: Preprocessing*)
- **3.3**: The system should apply Contrast-Limited Adaptive Histogram Equalization (CLAHE) to enhance weak edges and to deal with low-light conditions, foggy environments, and varying lighting conditions. (*Category: Preprocessing*)
- **3.4**: The system should apply morphological operations (opening, closing, dilation, erosion) to remove small noise and three-occlusion in mountains, identify and mask occluded areas to avoid false skyline detections and use interpolation + heuristic-based gap filling. (*Category: Preprocessing*)
- **3.5**: The system must offer the option to choose which edge detection algorithms to apply (Sobel or Canny). (*Category: Preprocessing*)
- **3.6**: The system must rank detected edges based on length of the skyline segment, higher contrast against the background, smoothness, continuity, height position in the image, and alignment with expected skyline geometry. (*Category: Preprocessing*)
- **3.7**: The system should automatically define a ROI in the upper portion of the image, where skylines are most likely located. (*Category: Preprocessing*)
- **4.0**: The system should apply contour-based segmentation to isolate potential skyline structures before full processing. (*Category: Feature Extraction*)
- **4.1**: If an image contains high-contrast regions, the system should focus on extracting skyline features from those areas first. (*Category: Feature Extraction*)
- **4.2**: The system should use OpenSIFT/ORB due to its robustness for keypoint-based feature extraction. (*Category: Feature Extraction*)
- **4.3**: The system must evaluate with a rank system the detected features and filter non-skyline features ignoring small, disconnected segments. (*Category: Feature Extraction*)
- **4.4**: The system should use Machine Learning classifiers, such as Support Vector Machines (SVM) to refine skyline candidate selection. (*Category: Feature Extraction*)
- **5.0**: The system should validate reconstructed skylines against edge continuity analysis (ensuring reconstructed sections align smoothly with detected edges). (*Category: Skyline candidacy*)
- **5.1**: The system should reject a skyline candidate if: it has excessive noise or discontinuities, its contrast is too weak to be a reliable skyline, or it is classified as a false skyline due to occlusions. (*Category: Skyline candidacy*)

- **6.0**: The system must allow users to save and generate a report (Yaml or JSON/XML) with the current session state, including preprocessed images, processing steps completed, extracted skyline candidates, rankings, skyline extraction accuracy metrics, and user-defined configurations (selected skyline extraction methods, filtering parameters). (*Category: Report Generation*)
- **6.1**: The system must auto-save session progress every 5 minutes. (*Category: Report Generation*)
- **6.2**: When closing a session, the system should confirm whether the user wants to save changes or discard modifications. (*Category: Report Generation*)

Non-Functional Requirements

- **1.0**: Ubuntu 20.04, 200GB available for storage, RAM: 7.6 GiB, Processor: Intel® Core™ i5-9300H CPU@2.40GHz×8, Graphics: GeForce® GTX 1650. (*Category: OS*)
- **2.0**: The system would use the GUI Frameworks PyQt5 and Tkinter libraries to build a consistent GUI. (*Category: GUI*)
- **2.1**: For image display and manipulation, handle different images, image processing, segmentation, analysis, and SIFT: OpenCV (cv2). (*Categories: GUI, Data acquisition, Preprocessing, Feature Extraction, Skyline Candidacy*)
- **2.2**: To plot overlays for detected skylines and visualization of ROI: Matplotlib. Version: 3.7.5. (*Categories: GUI, Preprocessing*)
- **2.3**: The image processing in the GUI of the system and image format handling: PIL (Pillow). Version: 10.4.0. (*Categories: GUI, Data acquisition*)
- **2.4**: In order to record the system flow and show the messages log: logging. Version: 3.8.10 (Builtin Python version). (*Category: GUI*)
- **3.0**: Data management: Pandas. Version: 2.0.3. (*Category: Data acquisition*)
- **4.0**: Since the system would need to perform fast image matrix operations: NumPy. Version: 1.24.4. (*Category: Preprocessing*)
- **4.1**: For the application of CLAHE, adaptive thresholding, morphological filtering, and region analysis: Scikit-Image. Version: 0.21.0. (*Categories: Preprocessing, Feature Extraction*)
- **5.0**: Since the system needs to perform interpolation for missing skyline sections, Scipy is needed. Version: 1.10.1. (*Category: Feature Extraction*)
- **5.1**: Scikit-Learn to apply the Support Vector Machine (SVM) for skyline classification. Version: 1.3.2. (*Categories: Feature Extraction, Skyline Candidacy*)
- **6.0**: For YAML/JSON report format, the libraries PyYAML and JSON are necessary. Version: (PyYAML) 5.3.1, JSON (2.0.9). (*Category: Report Generation*)

3.1.2 User Stories (see Annex A)

The definition of user stories provided a practical bridge between the identified requirements and the actual use of the software tool, as shown for each user story in Section 6.1. Each story captures the perspective of users and developers, with their roles, goals, and benefits in a format that reflects real-world interactions with the system. By framing the functionalities in terms of user expectations, such as dataset selection, preprocessing choices, feature extraction, and report generation, these stories help ensure that the design remains user-centered while also clarifying acceptance criteria for testing.

3.1.3 Conceptual Diagram

The conceptual diagram, as shown in Fig. 3.2, outlined the main structure of the software tool and the flow of information across its modules. It showed how the system acquired data from images, applied preprocessing to enhance visual cues, and extracted skyline features for analysis. It also included validation and classification steps to ensure detection in low-texture environments, such as in the sky without clouds. The final outputs were presented through the graphical user interface, which provided visualization, configuration options, and report generation.

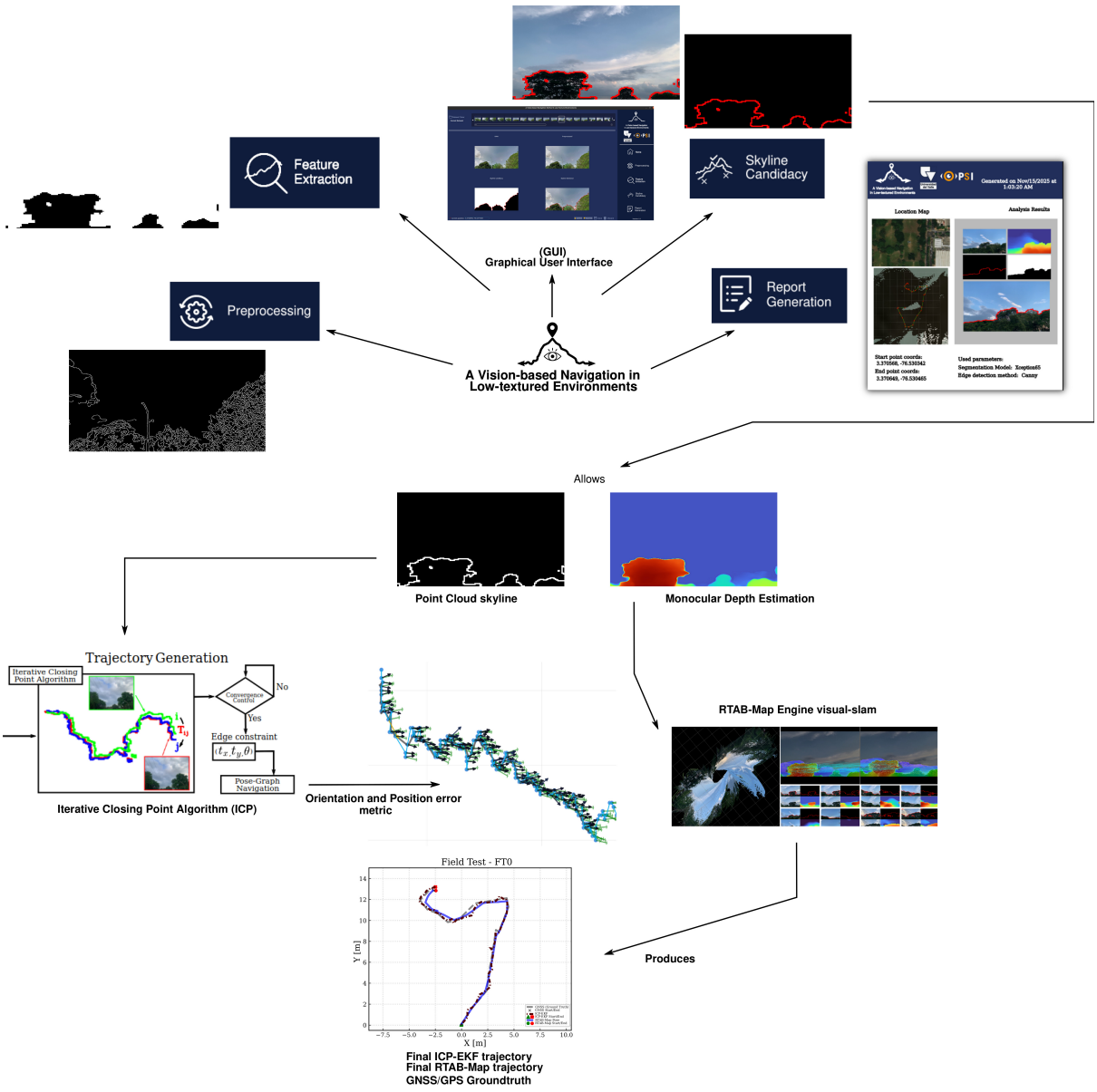


Figure 3.2: Extended conceptual diagram of the proposed vision-based navigation tool. The GUI coordinates data acquisition, preprocessing, feature extraction, skyline candidacy and detection, monocular depth estimation, ICP-based motion estimation (front-end), RTAB-Map pose-graph localization (back-end), and report generation into a modular pipeline for navigation in low-textured outdoor environments

In summary, the conceptual diagram in Fig. 3.2 shows how the proposed tool bridges low-level visual processing with high-level localization and reporting. The GUI orchestrates data acquisition, preprocessing, feature extraction, skyline candidacy and detection, while the monocular depth estimation block contributes complementary depth cues that reinforce skyline continuity and highlight distant landmarks. These perceptual cues feed the ICP-based motion estimation front-end, which transforms consecutive observations into incremental motion estimates for the pose graph. The RTAB-Map back-end then ag-

gregates these increments, fuses exteroceptive cues and GNSS when available, detects loop closures, and optimizes the global trajectory to keep drift bounded. Finally, the report generation module exports a compact PDF summary of the session, closing the loop between interactive experimentation, quantitative analysis, and map-based visualization.

3.1.4 Class Diagram

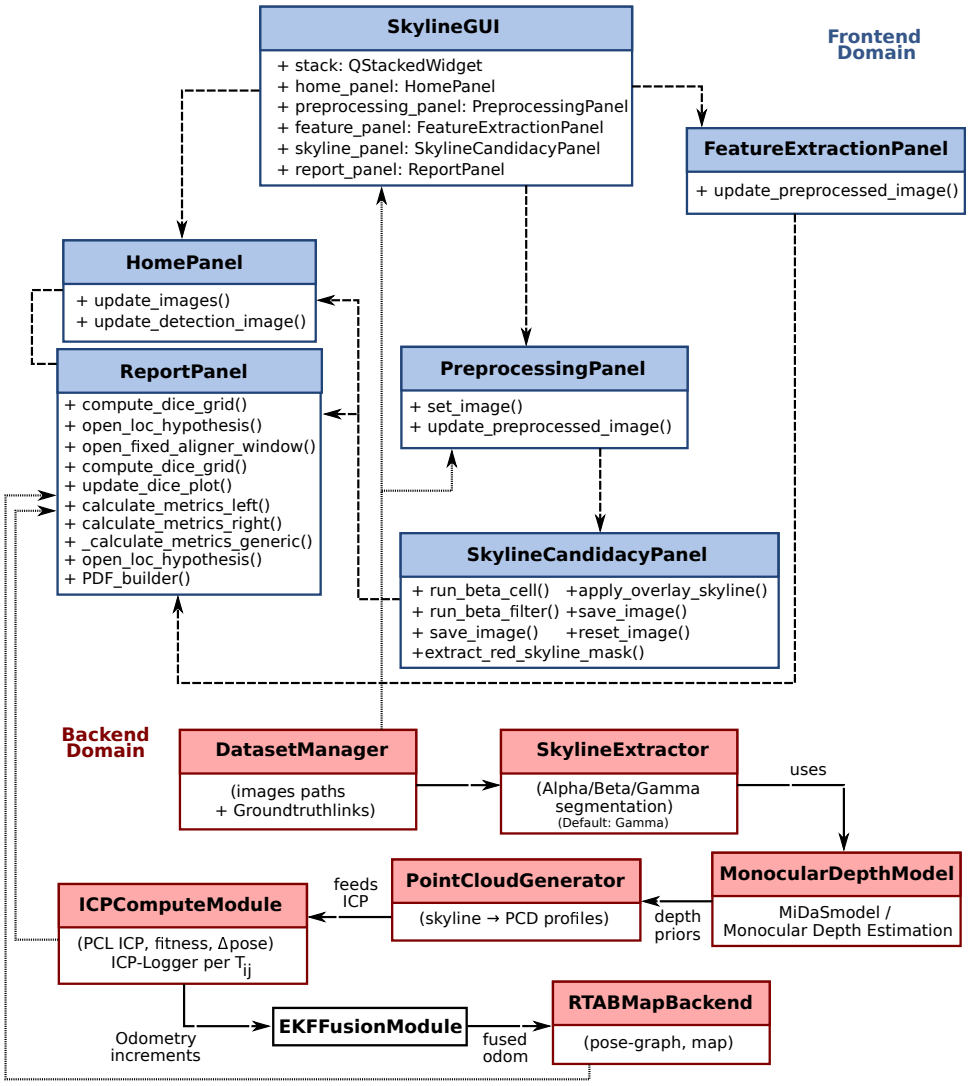


Figure 3.3: UML component diagram of the vision-based localization system. The Frontend Domain (blue) comprises the GUI panels used for dataset browsing, preprocessing, skyline extraction, and quantitative evaluation. The Backend Domain (red) contains the modules responsible for skyline segmentation, depth-augmented point cloud generation, ICP odometry, EKF fusion, and RTAB-Map pose-graph mapping. Dashed arrows indicate frontend dependencies on backend functionality via ROS parameters and intermediate artifacts, while solid arrows represent direct data flow between backend modules.

The shown UML architecture, as in Fig. 3.3, highlights briefly and summarizes the separation between the Frontend Domain, responsible for user interaction, visualization, and dataset navigation, and the Backend Domain, which encapsulates skyline extraction, depth-enhanced point cloud generation, ICP-based odometry, EKF fusion, and RTAB-Map pose-graph optimization. In this diagram, the purpose is to emphasize how the GUI panels interact with backend modules indirectly through ROS parameters, saved artifacts, and metric-driven evaluation workflows. This layered design allows modularity, facilitates debugging and testing of each subsystem, and establishes the potential to adopt adaptive learning mechanisms. Ultimately, the architecture reflects a fully operational pipeline that moves from dataset

preprocessing to skyline segmentation, localization inference, and finally mapping and pose-graph optimization.

3.2 Implementation

Once the relationships between the system's functional and non-functional requirements were defined and the interactions among components were outlined in the conceptual map, the implementation phase was carried out following a modular design strategy. The system was developed under the ROS environment, with communication between processing nodes and graphical elements. As shown in Fig. 3.4, the proposed architecture is organized into three main modules: Configuration, Execution, and Report. Each module encapsulates a specific stage of the skyline-based localization process, which enables independent development and synchronized operation through the ROS parameter server. The following subsections describe in detail the functionality of each module: it starts with the Configuration Module, which manages dataset selection, parameter setup, and algorithm initialization for subsequent execution.

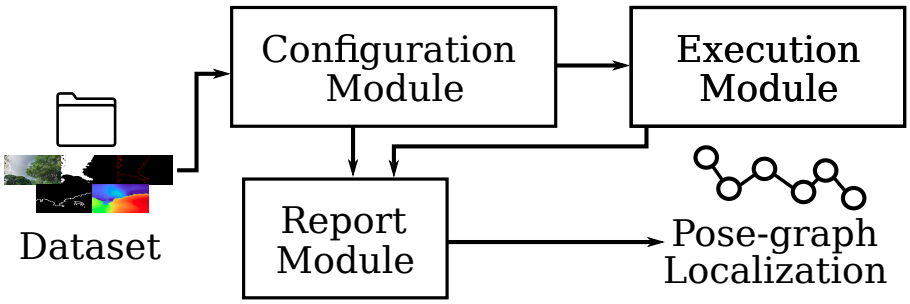


Figure 3.4: Overview of the system architecture showing the interaction between modules. The dataset is configured in the Configuration Module, processed in the Execution Module, and the results are consolidated in the Report Module to support pose-graph navigation

3.2.1 Configuration Module

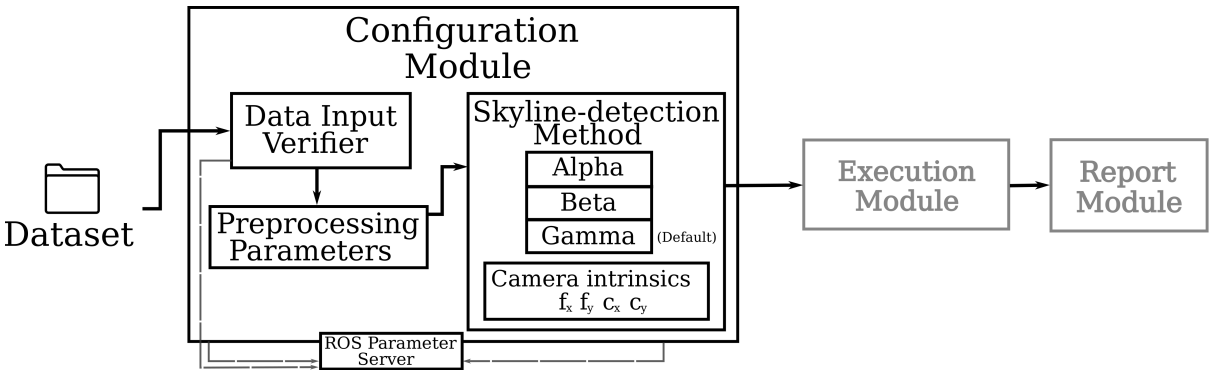


Figure 3.5: Diagram of the Configuration Module. Here, the flow goes from dataset input through data verification, preprocessing parameter setup, and skyline-detection method selection. All parameters are stored in the ROS Parameter Server for synchronization with subsequent modules

The Configuration Module serves, in this work, as the initial interface of the application, where all parameters required before running any skyline–detection or localization process are defined, as shown in Fig. 3.5). Through an integrated graphical environment, the user specifies dataset inputs, preprocessing options, skyline–detection method, system connections, and reporting preferences. This module is purely configurational: it does not perform image processing itself, but instead verifies the dataset, prepares and validates all selected settings, and stores them in the ROS Parameter Server so that subsequent nodes in the Execution and Report Modules operate consistently under the user–defined conditions.

Dataset Input Verifier: The first subsection of the interface is dedicated to select and manage input data. A Dataset Selector window allows users to choose the dataset directory, as shown in Fig. 3.20 (a), and displays all its frames in a carousel view if it is a valid dataset directory. Internally, the application backend verifies that the selected folder contains subfolders with the appropriate structure and that a .csv file with the GNSS logging is present, as shown in (b) of the same figure. This verification is performed through key-name matching. After this step, the user is free to start the pipeline and explore the different tabs of the application, such as in (c), where the preprocessing configuration can be selected, as will be described below. This component of the configuration module ensures that all subsequent processes receive valid and traceable data sources and that the dataset is parsed correctly.

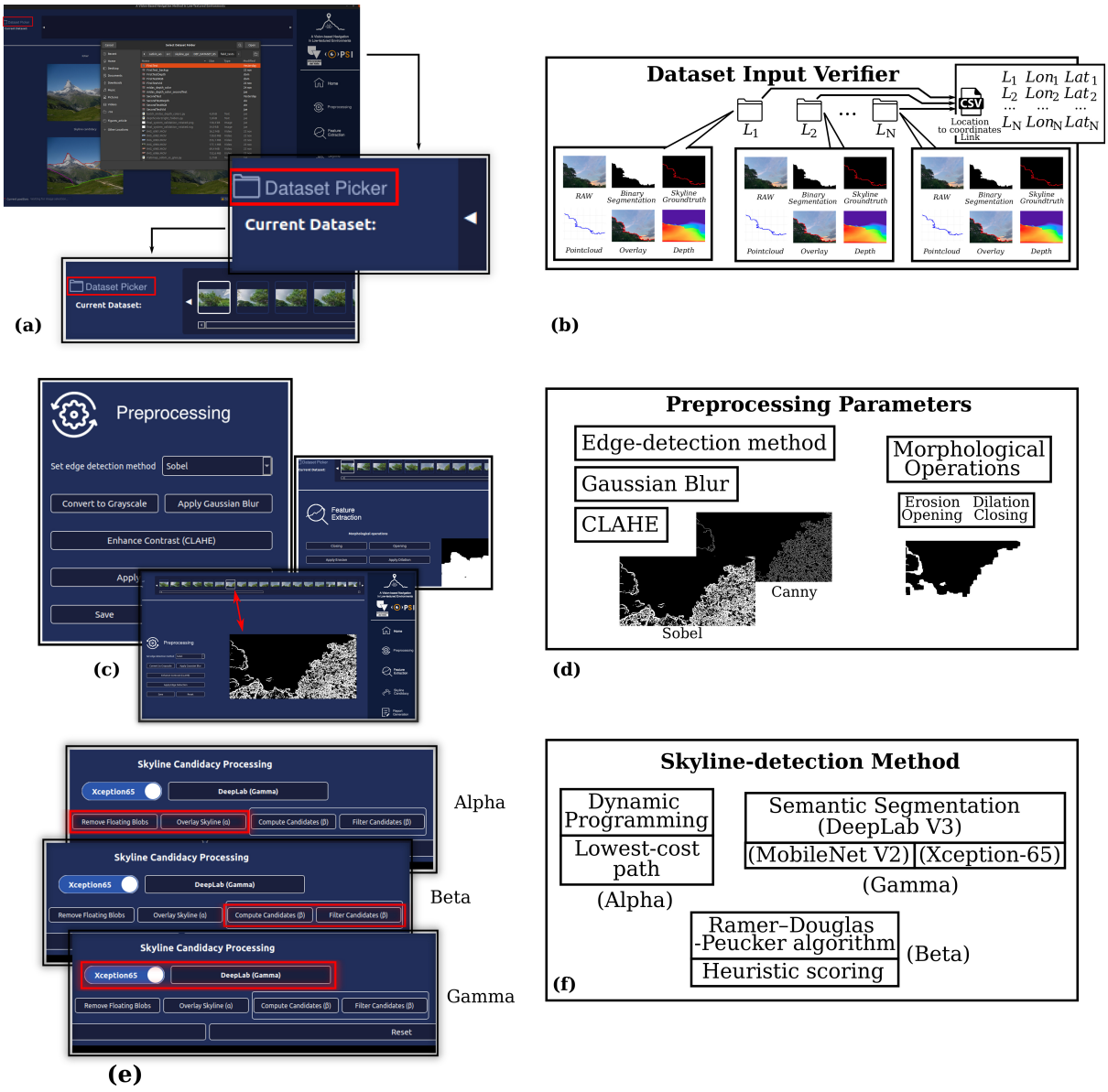


Figure 3.6: Correspondence between the Configuration Module interface in the GUI and its backend process diagram. Panels (a), (c), and (e) show GUI screenshots for dataset selection, preprocessing settings, and skyline-method configuration, respectively. Panels (b), (d), and (f) show the corresponding backend diagrams, which illustrate the data flow and the organization of parameters associated with each of these interface components

Preprocessing Parameters: This panel allows the user to set default values for brightness, the edge-detection method, and optional noise-reduction filters such as Gaussian smoothing and Contrast Limited Adaptive Histogram Equalization (CLAHE). A temporary preview also enables visual inspection of these settings without modifying the dataset, as shown in Fig. 3.20 (c), where the red arrow indicates

Table 3.1: Skyline GUI Configuration Parameters and Default Values

Parameter	Default Value
Dataset Input	
Dataset directory	User selection
Image/frame selector	First thumbnail
Metadata file	`metadata.csv`
GNSS validation	Enabled
Camera Intrinsics (Loaded Automatically)	
Focal length (f_x, f_y)	Loaded from CSV
Principal point (c_x, c_y)	Loaded from CSV
Distortion coefficients $(k_1, k_2, p_1, p_2, k_3)$	Loaded from CSV
Image size (W, H)	Dataset value
Scaling factors (s_x, s_y)	Auto-computed
Intrinsic matrix K	Auto-generated
Preprocessing	
Apply grayscale	Off
Gaussian blur kernel	7×7
Gaussian sigma	2.0
CLAHE enabled	Off
CLAHE clip limit	3.0
Edge detector	Canny
Morphology: erosion	Off
Morphology: dilation	Off
Morphology: opening	Off
Morphology: closing	Off
Skyline Candidacy	
Skyline method	Gamma (DeepLab)
DeepLab model	MobileNetV2
Xception-65 toggle	Off
Sky class ID	3
Pole class ID	88
Boundary thickness	5 px
Beta () Pipeline	
Thresholding method	Otsu
Minimum run length	12 px
Maximum gap columns	3 px
Maximum slope per pixel	3 px
Thickening iterations	2
Alpha () Pipeline	
DP smoothness cost	0.5
DP data cost	1.0
-edge detector	Canny
Gamma (DeepLab) Pipeline	
Model path	MobileNetV2 .tflite
Inference threads	4
Mask extraction	Argmax
Report & Evaluation	
Hausdorff normalization	Enabled
Dice comparison path	Auto-resolved
Auto-fit overlay	Enabled
Localization hypothesis	Enabled
RTAB-Map database path	Not set

the preview area and the original thumbnail image. Although no processing occurs at this stage, these parameters define the conditions that will be applied during execution, which is why both preprocessing and feature-extraction parameters are grouped within this section. Preprocessing options also include configurations such as edge detection, binary mask generation, and morphological operations (dilation, erosion, opening, and closing), as summarized in panel (d) of the same figure.

The Skyline-detection Method Selection: This configuration panel governs how the skyline will later be identified and classified. Three principal detection pipelines are available: Alpha, Beta, and Gamma, each method represents a distinct computational strategy. Through a set of buttons, as in Fig. 3.20 (e), the user can select one of these methods before start the experiment. When Gamma is chosen, a toggle button allows the selection of the neural-network backbone to be used by the DeepLab V3 semantic segmentation model, either MobileNet V2 for lightweight execution or Xception-65 for higher accuracy. Panel (f) of the same figure summarizes the backend strategy used behind the GUI functionalities.

Ros Parameter Server: This mechanism stores all user-selected parameters in a centralized dictionary, which ensures real-time consistency across every module of the application. Preprocessing settings, algorithm choices, and reporting options are published under a structured naming convention, and this allows other nodes and interfaces to access updated configurations dynamically without manual reinitialization or searching through folders or configuration files. As shown in Fig. 3.5, the camera intrinsic parameters used during video acquisition are also loaded from the *.csv* file located at the dataset root. Once read, these intrinsics are made available through the ROS Parameter Server. Offline localization processes rely on the correct calibration. Failure to define these intrinsic parameters can lead to distortion models that directly affect the accuracy and geometric consistency of the generated trajectory.

In summary, the Configuration Module serves as the control center of the application, where complete customization of all operational parameters is possible without any prior processing itself. Its modularity aims to allow the user to try each experiment to be conducted under precise defined conditions before doing the transition to the Execution and Reporting modules. Table 3.1 summarizes all the parameters that the user is able to configure.

Towards a default skyline-detection algorithm:
As described in the Configuration Module, three skyline-detection algorithms were implemented: the Alpha, Beta, and Gamma methods. They represent distinct approaches to skyline extraction, and it goes from classical edge-based and heuristic techniques to deep-learning semantic segmentation. Each of these methods will be briefly below.

Alpha method: This implementation, inspired by the robust DP-based skyline extraction approach proposed in [7], applies a Dynamic Programming (DP) strategy to extract a continuous skyline from edge-detected images. After the generation of a binary edge map using Canny or Sobel filters with Otsu thresholding, a multi-stage graph is constructed where pixels act as nodes, and columns represent stages. The DP algorithm then searches for the lowest-cost path across the graph, which favors smooth, high-position transitions to reconstruct a consistent skyline even under noise, clouds, or low contrast. Within the application, users can configure key parameters such as the chosen edge detector and the thresholding technique.

Beta method: Combines Canny edge detection with adaptive thresholding to extract dominant edges, which are then simplified into piecewise-linear segments using the Ramer–Douglas–Peucker algorithm. Each segment is evaluated through a heuristic scoring process that considers geometric and color cues such as length, contrast with the sky, and similarity to the ground region. This approach follows the segment-based skyline extraction strategy described in [6]. At the end of this method, greedy graph search connects the highest-scoring segments, emphasizing structural continuity while allowing strong visual features to guide the skyline path.

Gamma method: Employs a DeepLab V3 semantic segmentation network implemented through TensorFlow Lite. This pretrained model allows the user to select between MobileNet V2 and Xception-65 backbones. These architectures, trained on the ADE20K dataset, classify each pixel into semantic categories, from which the sky class is extracted for skyline detection. Once an image is selected, the system

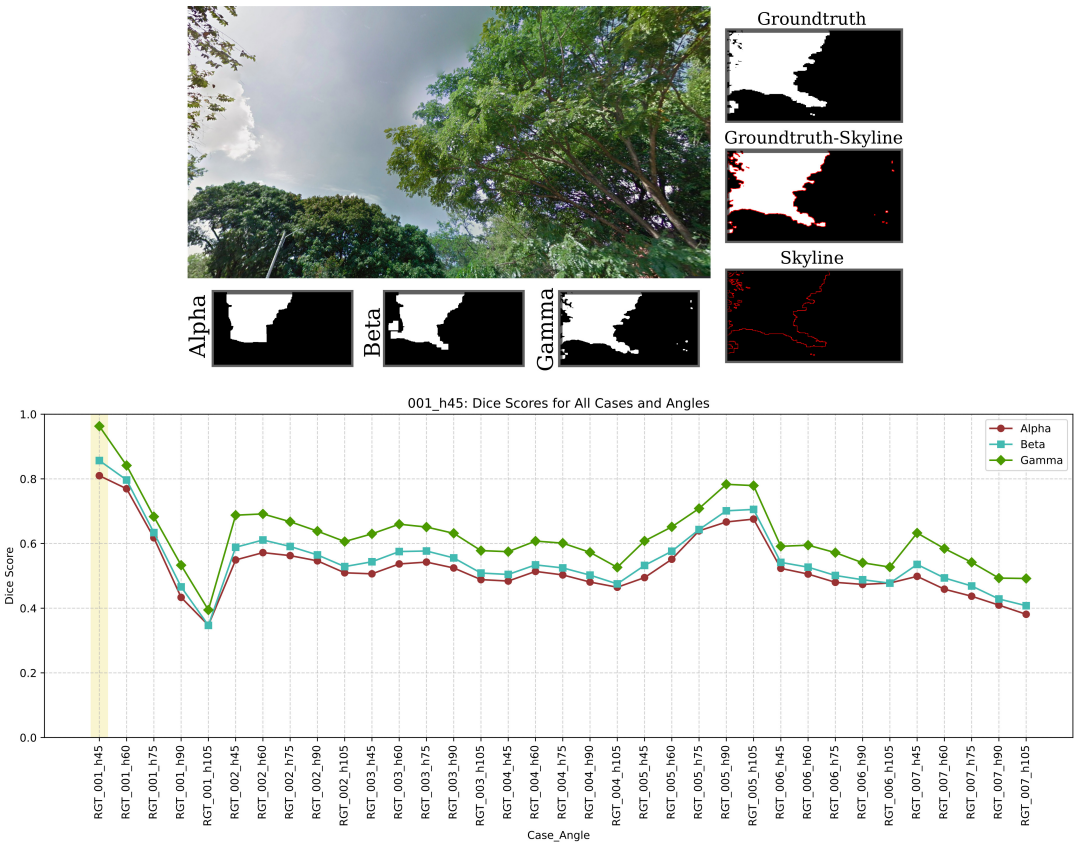


Figure 3.7: In this evaluation, the behavior of the Dice coefficient was analyzed in the presence of skylines composed across multiple planes. The metric's behavior remains stable in the vicinity of the evaluated image/location, which translates into robustness against variability in yaw movement, reaching a maximum of 81% for method Alpha and a maximum of 97% for method Gamma

loads the chosen model, preprocesses the input, performs pixel-wise inference, and decodes the resulting mask. While other works apply semantic segmentation for different targets, such as tree-canopy extraction in [38], here the sky class is isolated and converted into a binary mask, from which a red skyline boundary is obtained using dilation and XOR operations.

To evaluate the behavior of the three skyline-detection methods under different real-world conditions, four representative test locations were selected. Each location presents a distinct type of skyline composition and visual challenge, and each is analyzed independently in Figures 3.7–3.10.

Figure 3.7 corresponds to a scene with multiple depth layers (see the position of each tree canopy), which results in overlapping canopies and a skyline that should become discontinuous across several planes, but successfully merges all the skylines. Figure 3.8 evaluates a low-texture environment dominated by cloudless sky regions and visual artifacts such as power lines. Figure 3.9 focuses on extended cloudiness gradients, where illumination and sky–cloud boundaries vary progressively across the frame. Finally, Figure 3.10 examines a location with isolated tree structures and patchy clouds, which produce irregular skyline silhouettes with abrupt height changes and disruptions in the depth layers.

In each of these four evaluations, the Alpha, Beta, and Gamma methods were applied to the same input sequence, and their outputs were compared both quantitatively and qualitatively using the Dice score coefficient across multiple viewing angles. The lower row of each figure displays the segmentation results produced by each method, along with the corresponding dataset frame, while the accompanying plot illustrates how their Dice scores evolve throughout the rotation sequence. Together, these four case studies provide a representative assessment of the behavior of the three methods under different environmental and visual conditions.

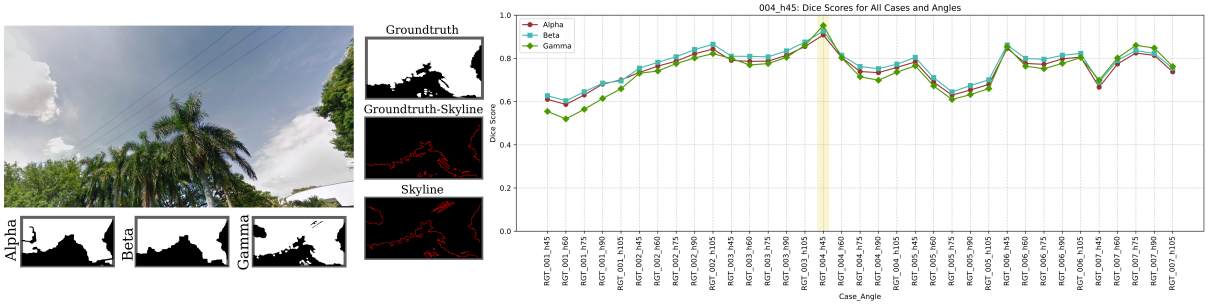


Figure 3.8: In the evaluation of this location, the performance of the methods was analyzed under conditions of low texture caused by clusters with cloudless areas and artifacts such as the presence of non-skyline elements, including power distribution lines and significant cloud cover. The Dice coefficient exhibited stable behavior with no outliers, reaching a maximum of 91% for method Alpha and a maximum of 96% for method Gamma

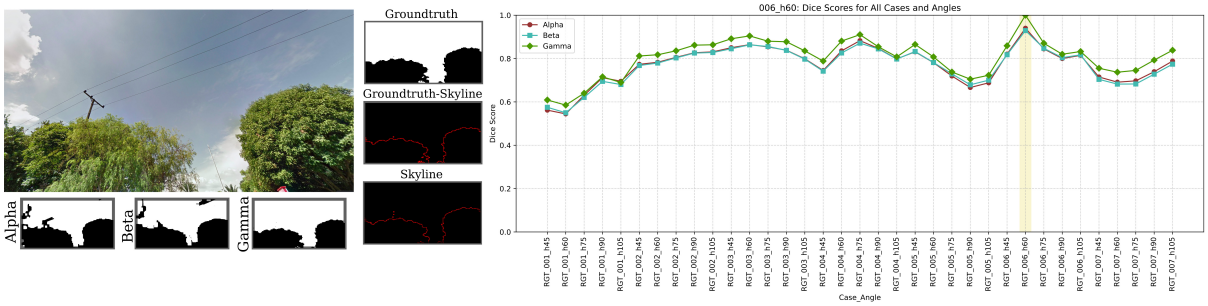


Figure 3.9: This result focused on evaluating the system's measurement performance in extended areas with cloudiness gradients, reaching a maximum of 92% for method Beta and a maximum of 98% for method Gamma

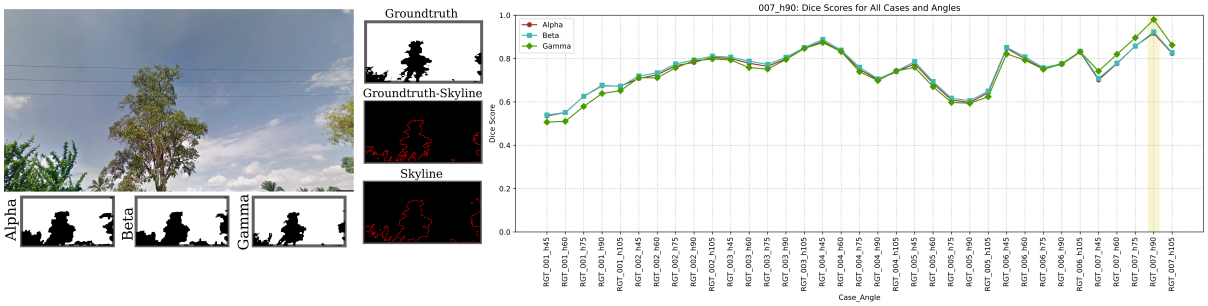
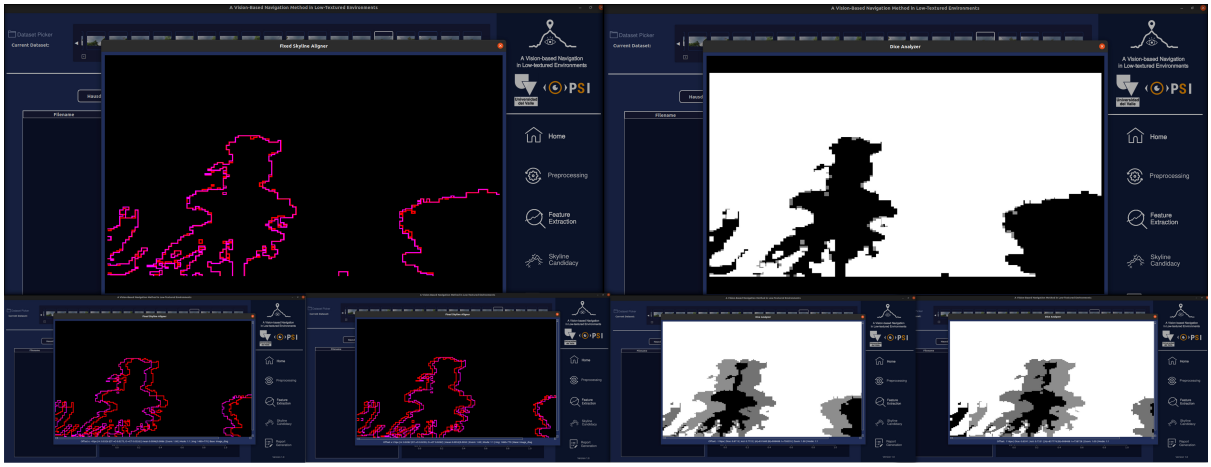


Figure 3.10: In this evaluation, the system was tested on a scene featuring clouds in different shades and isolated elements forming the skyline, reaching a maximum of 92% for method Alpha and a maximum of 97% for method Gamma

Finally, the Configuration Module is complemented by a set of interactive analysis tools (Fig. 3.11) that enables the evaluation of the selected default method (Gamma). Through the Fixed Skyline Aligner and the Dice/Hausdorff analyzers, the user can visually inspect the overlap between detected and ground-truth skylines, verify the numerical scores reported for each frame, and quickly identify failure cases or ambiguous regions. This diagnostic capability not only supports the selection of Gamma as the default method, but also provides a practical means to tune preprocessing parameters and validate new segmentation models before they are integrated into the full localization and trajectory–generation pipeline. The quantitative evaluation for this is provided in Section 4.



(a) (b)

Figure 3.11: Graphical tools for evaluating skyline segmentation performance. The software GUI provides dedicated windows for overlays detected and ground-truth skylines, through the usage of Dice/Hausdorff Analyzer, which visualizes the corresponding binary masks and reports their similarity scores. These tools allow the user to interactively inspect how well the Gamma method reproduces the reference skylines across the dataset

3.2.2 Execution Module

Once the dataset and preprocessing options are loaded from the Configuration Module, the user initiates the execution stage, which performs sequential image analysis and generates the corresponding localization trajectory. As illustrated in Fig. 3.12, this module integrates vision-based segmentation with sensor-data fusion and not only estimates geographic position and orientation from the visual information obtained in each frame, but also constitutes the operational core of the system. It is responsible for running the skyline-segmentation, localization, and trajectory-generation processes according to the configuration parameters previously defined.

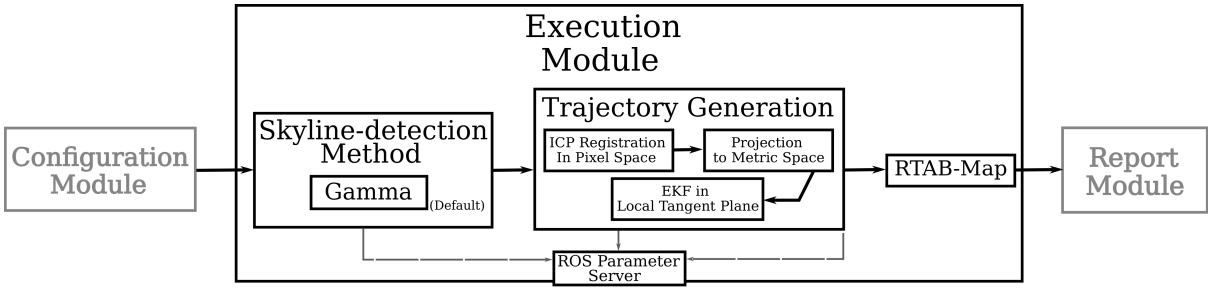
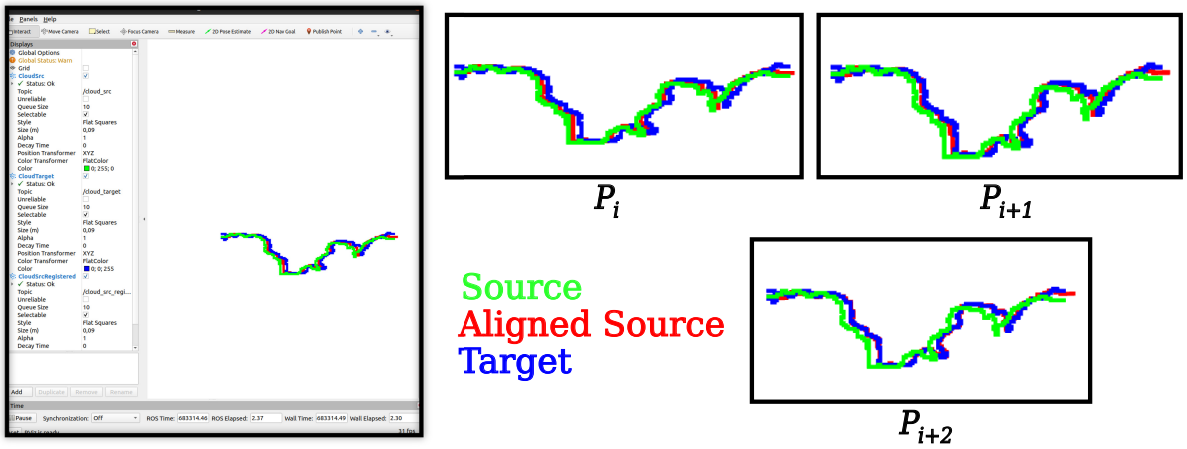


Figure 3.12: Internal workflow for Execution module, from skyline detection to trajectory generation. The selected skyline-detection method (default: Gamma) produces segmentation results used to form a localization hypothesis based on the maximum Dice score and minimum Hausdorff distance. These estimates are then used in the trajectory generation stage, which employs the Iterative Closest Point algorithm and pose-graph navigation to build a spatial path. All parameters are synchronized in real time through the ROS Parameter Server

Trajectory Generation via ICP-EKF Fusion: The transformation from visual skyline features to metric localization proceeds through three interconnected stages: (1) ICP registration in pixel space, (2) scaling to metric units via camera geometry, and (3) fusion with GNSS through an Extended Kalman Filter in a local tangent plane.



(a) (b)

Figure 3.13: Visualization of the ICP registration process in pixel space. (a) Screenshot in RViz of the registration interface that displays an active alignment of skyline point clouds. In (b), Sequential alignment results for consecutive frames (P_i, P_{i+1}, P_{i+2}). The Source cloud (Green) is iteratively transformed to match the Target cloud (Blue), which results in the Aligned Source (Red). The close overlap between the Red and Blue lines demonstrates successful rigid alignment and the recovery of relative motion parameters between frames

Stage 1: ICP Registration in Pixel Space As shown in Fig. 3.13, a typical ICP registration occurs: the source and target skyline point clouds are aligned through iterative correspondence matching, which produces motion estimates with fitness scores typically below 0.01 for high-quality alignments in this work. In the skyline-based localization pipeline, each image produced a discrete pose hypothesis, which represents states as a set of points along a certain path. To bridge the gap between pixel obtained parameters and a meaningful spatial representation gap, the Iterative Closest Point (ICP) algorithm, previously introduced in Section 2.2, first aligns consecutive skyline-derived point clouds $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ to solve the rigid alignment problem. Its rotation encodes the angle increment θ, while the translation vector $\mathbf{t}_{\text{ICP}} = [t_x, t_y]^\top$ represents the in-plane displacement, both expressed initially in image pixel coordinates, as formalized in algorithm 3.2.1. Although the motion is planar, embedding it in $SE(3)$ provides a way of logging positions throughout the system, only if Stages 2 and 3 are being carried out.

Algorithm 3.2.1 ICP Execution Stage for Skyline-Based Metric Odometry

Require: Dataset $\mathcal{D}$, initial global localization guess X_0
Require: Georeferenced skyline models $\mathcal{G}$ (each with known global coordinates)
$\mathcal{E} \leftarrow \emptyset$ ▷ Initialize set of metric odometry edges
for each image I_i in $\mathcal{D}$ **do**
  $S_i \leftarrow \text{extract_skyline}(I_i)$
  $P_i \leftarrow \text{skyline_to_pointcloud}(S_i)$
  $G_i \leftarrow \text{select_closest_georef_skyline}(\mathcal{G}, S_i)$
  $M_i \leftarrow \text{project_to_local_metric_plane}(G_i, X_0)$
  $T^{\text{local_init}}, \text{fit}^{\text{init}} \leftarrow \text{ICP_align}(P_i, M_i)$
end for
for each consecutive pair (P_i, P_{i+1}) **do**
  $T_{i,i+1}, \text{fitness} \leftarrow \text{ICP_align}(P_i, P_{i+1})$
  $(t_x, t_y, \theta) \leftarrow \text{decompose_metric_vector}(T_{i,i+1})$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i \rightarrow i{+}1, t_x, t_y, \theta)\}$
end for
return Pose-Graph Edges $\mathcal{E}$

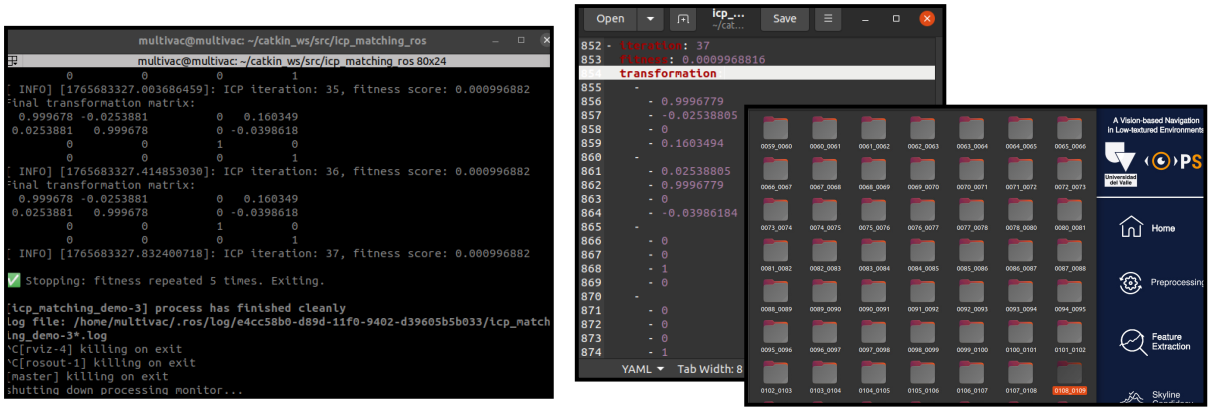
In addition to the transformation, ICP computes a fitness score that measures how well the source and target skyline point clouds align. As evaluated, for instance in Fig. 4.9, this value is logged and

used to color-code edge quality and to detect unreliable registrations. The recovered motion parameters (t_x, t_y, θ), still in pixel coordinates, would become the edge constraints between consecutive pose-graph nodes, and this acts as both, the incremental skyline-based odometry estimate and the foundation for the pose-graph construction shown in Fig. 3.22, once going through Stages 2 and 3.

In this work, the process was configured with a convergence threshold of $\epsilon = 10^{-6}$. Furthermore, a correspondence distance of 10 pixels was established to define the maximum limit for point-to-point matching. For each frame pair, ICP outputs a homogeneous transformation matrix $\mathbf{T}_{i,i+1}$ that contains a rotation $\mathbf{R}$ and translation $\mathbf{t}$. The planar motion parameters were extracted as in 9:

$$\theta = \text{atan2}(r_{21}, r_{11}), \quad \mathbf{t}_{\text{pixel}} = [t_x, t_y]^\top \tag{9}$$

where θ is the yaw angle change and $\mathbf{t}_{\text{pixel}}$ is the 2D displacement in pixel coordinates.



(a) **(b)**

Figure 3.14: Execution and data logging of the ICP registration pipeline. (a) Terminal output shows the iterative convergence process, which results in the final transformation matrix and its fitness score. (b) Systematic logging mechanism of the computed registration results across the dataset, organized by frame pairs

To validate the registration performance across the entire trajectory, the system continuously monitors and records the ICP output. As depicted in Fig. 3.14 (a), the terminal interface reports the iterative convergence, and displays the final homogeneous transformation matrix $\mathbf{T}_{i,i+1}$ and the associated fitness score. This data is systematically archived for every consecutive frame pair, as shown by the directory structure in Fig. 3.14 (b). This logging strategy enables the health-check/evaluation mechanism to detect unreliable edges based on fitness thresholds, so the global pose graph in Stages 2 and 3 can be constructed.

Stage 2: Projection to Metric Space via Camera Geometry The pixel displacements $\mathbf{t}_{\text{ICP}}$ obtained from ICP must be converted into physically meaningful metric distances (meters) for trajectory estimation. This conversion compensates for the fundamental limitation of monocular vision: the inability to recover absolute scale and depth from visual data alone. In this work, the projection relies on the pinhole camera model [39] (Chapter 6, pp. 153-157). Under this model, a 3D point $(X, Y, Z)^T$ in the camera coordinate frame projects onto the image plane as:

$$u = \frac{f_x \cdot X}{Z}, \quad v = \frac{f_y \cdot Y}{Z} \tag{10}$$

where f_x and f_y are the focal lengths expressed in pixels, and Z represents the depth of the observed point. Then, for small changes of horizontal metric displacement mapped to pixel displacement, the relationship is expressed as:

$$\Delta X = \frac{\Delta u \cdot Z}{f_x} \tag{11}$$

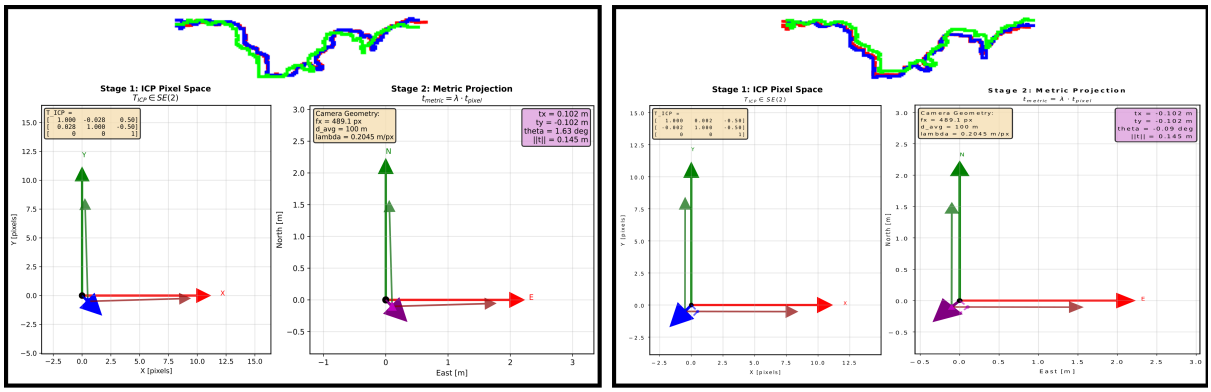
This projection relationship, derived directly from the pinhole camera model, enables metric scale recovery in monocular visual odometry when a depth reference is available. Works such as [40] applies also,

from the pinhole camera model, geometric derivation principle to recover a metric scale factor. Given the limitations of monocular vision, and although the approximation of metric scale recovery for translation lacks representative significance until fused via EKF with GNSS in this work, metric scaling is necessary to position the system within a metric plane, whereas orientation estimation is independent of metric scale. Based on this premise and the following assumptions, metric estimation was performed: a far-field assumption, where the observed skyline features lie at a depth $Z = d$ much greater than the camera's focal length ($d \gg f_x$). As a consequence of the far-field assumption, the depth to the skyline remains approximately constant across highly overlapped-consecutive frames. Therefore, for an object at depth d, this simplifies to:

$$\Delta x = \frac{\Delta u \cdot d}{f_x} \tag{12}$$

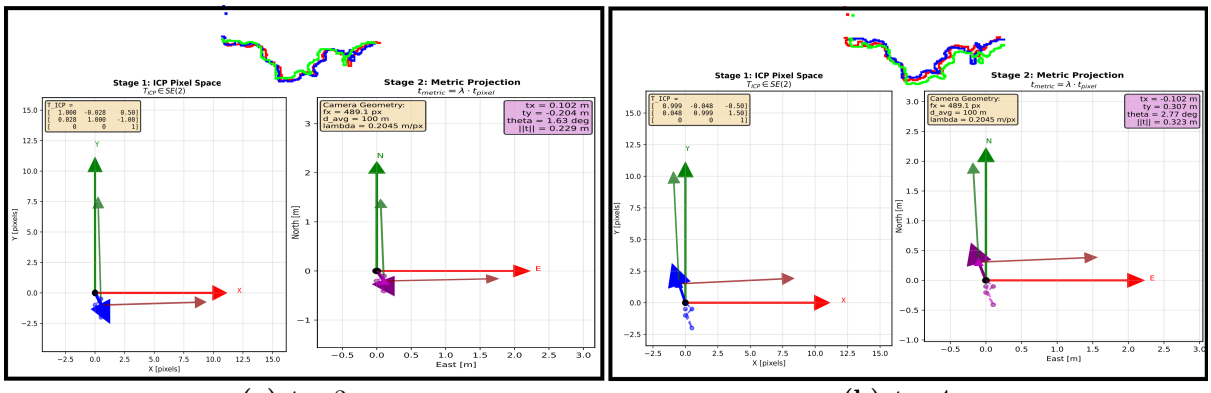
In this work, skylines lie at significant distances relative to the camera's focal length, which satisfies the far-field assumption ($d \gg f_x$). The test environments feature skyline elements (tree canopies, vegetation) located at distances ranges from 50 to 150 m. Thus, a representative average depth of $d_{\text{avg}} = 100$ m is used and the scale factor becomes:

$$\lambda = \frac{d_{\text{avg}}}{f_x} = \frac{100 \text{ m}}{489.07 \text{ px}} \approx 0.204 \text{ m/px} \tag{13}$$



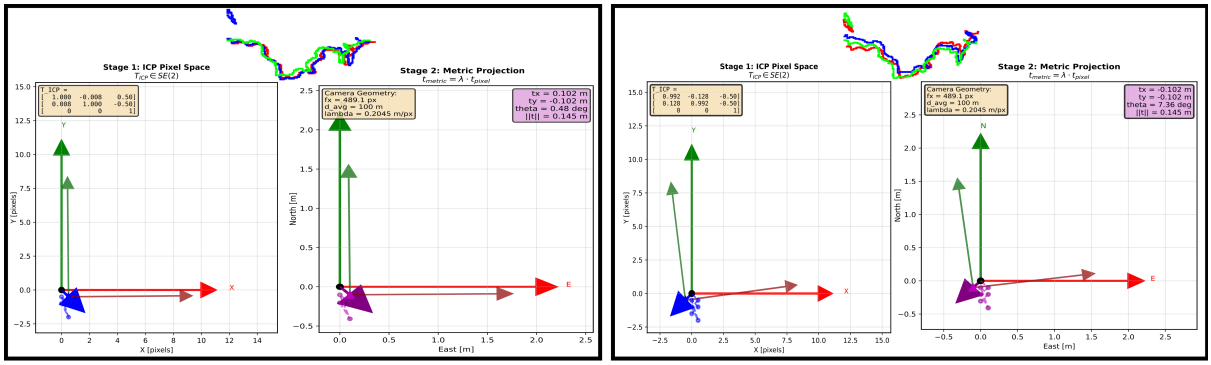
(a) $t = 1$ **s** **(b)** $t = 2$ **s**

Figure 3.15: Sequential skyline alignment which demonstrates pixel-to-metric projection for the first two timesteps. (a): ICP registration results in pixel coordinates showing the source skyline (green), target skyline (blue), and aligned result (red) with displacement vectors. (b): Same alignment transformed to metric coordinates via the scale factor $\lambda = 0.204$ m/px



(a) $t = 3$ **s** **(b)** $t = 4$ **s**

Figure 3.16: Continuation of the sequential alignment series at intermediate timesteps. The progression shows consistent metric projection across the trajectory, with accumulated displacements building the path incrementally. The transformation maintains angular relationships and relative positions, and confirms that the scale factor provides a pixel-to-meter conversion throughout the sequence



(a) $t = 5$ **s** **(b)** $t = 6$ **s**

Figure 3.17: Final timesteps of the alignment sequence that illustrates in (b) the complete pixel-to-metric transformation chain. By $t = 6$ s, the accumulated trajectory in metric space provides a physically meaningful representation for fusion with GNSS measurements in the EKF framework. The consistent scale factor application across all timesteps validates the far-field depth assumption in the projection model

For the datasets evaluated in this work, images were downsampled from the original capture resolution of 1920×1080 px to 640×360 px. To maintain geometric consistency, the camera intrinsic parameters were scaled linearly by the resolution ratio $s = W_{\text{new}}/W_{\text{old}} = 640/1920 \approx 0.333$. The native horizontal focal length $f_x = 1467.22$ px (which corresponds, roughly, to 4.25 mm physical focal length of the visual sensor used for this work, which was a 1/2.55" sensor) was obtained as follows in 14. Similarly, the principal point coordinates and vertical focal length were scaled by the same factor to generate the effective camera matrix.

$$f'_x = f_x \cdot s = 1467.22 \cdot \frac{1}{3} \approx 489.07 \text{ px} \tag{14}$$

Recalling, the test environments (Universidad del Valle sports fields complex) feature skyline elements (that were predominantly tree canopies and vegetation located at distances) ranges from 50 to 150 meters from the camera trajectory. Once again, if a representative average depth of $d_{\text{avg}} = 100$ m is taken, the scale factor λ is estimated as:

$$\lambda = \frac{100 \text{ m}}{489.07 \text{ px}} \approx 0.204 \text{ m/px} \tag{15}$$

This value serves as the link between visual odometry and the physical world. The metric displacement vector is then computed as:

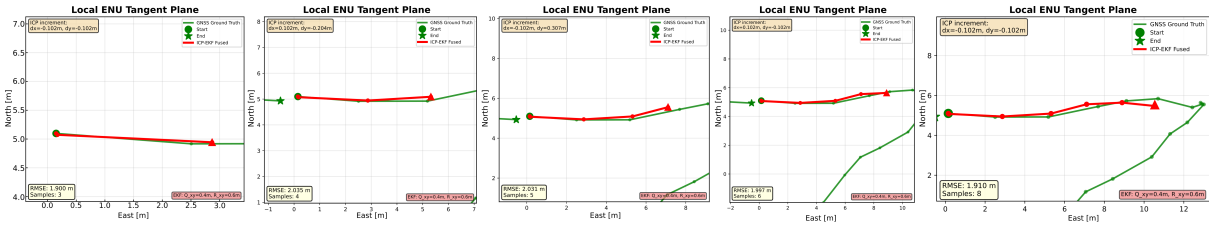
$$\mathbf{t}_{\text{metric}} = \lambda \cdot \mathbf{t}_{\text{ICP}} = [\lambda \cdot t_x, \lambda \cdot t_y]^\top \quad \text{[meters]} \tag{16}$$

This scaled displacement vector, together with the rotation angle θ extracted from the ICP transformation, defines the edge constraints used in the final pose-graph construction and EKF-based trajectory optimization. The empirical validation of this scale factor is presented in Section 4.2.2.

Stage 3: EKF Fusion with GNSS in Local Tangent Plane Both ICP-derived and GNSS trajectories must share a common reference frame for sensor fusion, as illustrated in Fig. 3.18. Following [41], GNSS coordinates are projected onto a local East-North-Up (ENU) tangent plane centered at the initial position $(\text{lat}_0, \text{lon}_0)$. This projection transforms angular differences (degrees) into metric distances (meters) using Earth's radius R_{earth}:

$$\begin{aligned} x^k_{\text{GNSS}} &= R_{\text{earth}} \cdot (\text{lon}_k - \text{lon}_0) \cdot \cos(\text{lat}_0), \\ y^k_{\text{GNSS}} &= R_{\text{earth}} \cdot (\text{lat}_k - \text{lat}_0) \end{aligned} \tag{17}$$

Simultaneously, the ICP-derived trajectory is constructed by cumulative scaled displacements:



(a) $t = 2$ **s** **(b)** $t = 3$ **s** **(c)** $t = 4$ **s** **(d)** $t = 5$ **s** **(e)** $t = 6$ **s**

Figure 3.18: Sequential evolution of EKF-based sensor fusion in the local ENU tangent plane. Each subplot represents a timestep that shows ICP-EKF estimate (red line with triangles) odometry trajectory, (ii) raw GNSS measurements (green line) . The progression demonstrates how the Extended Kalman Filter recursively combines relative visual odometry constraints with absolute GNSS position updates to produce a globally consistent trajectory. RMSE values quantify the alignment quality between the fused trajectory and GNSS ground truth, typically ranging between 1.6-2.0 m across the sequence

$$\mathbf{p}_{\text{ICP}}^{k} = \sum_{i=1}^{k} \lambda \cdot \mathbf{t}_{\text{ICP}}^{i} \tag{18}$$

With both trajectories expressed in meters within their respective local frames, an Extended Kalman Filter (EKF) fuses them by treating ICP increments as the process prediction model and GNSS positions as absolute measurement updates. The validity of this local tangent plane approximation [41] depends on two key conditions: first, the trajectory extent must be much smaller than Earth's radius (R_{earth}), which is satisfied by the datasets evaluated in this work (all < 1 km); and second, the latitude must not be near the poles, where projection distortion becomes significant (tested at 3.37°N, well within the mid-latitude or ecuatorial range). Under these conditions, tangent plane distortion remains not significant, which ensures that the Euclidean approximation do not introduces geometric error for the spatial metric scales considered in this work. The complete offline execution of this pipeline, including coordinate frame alignment and ROS message publication, is formalized in Algorithm 3.2.2.

The sequential evolution of the ICP-EKF fusion process that initially was estimated in Figs 3.15 to 3.16 is illustrated in Fig. 3.18, which presents five consecutive timesteps ($t = 2$ s through $t = 6$ s) of the trajectory estimation. Each subplot shows the state of the local ENU tangent plane at a specific moment, which displays the both, the raw GNSS measurements (green), and the ICP-EKF-fused result derived odometry, with the red line with triangle markers.

As the sequence progresses, the EKF recursively integrates the relative motion constraints from ICP with the absolute position measurements from GNSS. This produces a trajectory that combines the smooth, local motion estimates from visual odometry with the globally referenced corrections from satellite positioning. The progressive alignment visible across the timesteps demonstrates how the filter balances prediction uncertainty (from ICP drift) with measurement uncertainty (from GNSS noise), which converges toward a fused estimate that outperforms either sensor modality independently. The Root Mean Square Error (RMSE) metrics shown in each subplot quantify the agreement between the fused trajectory and the GNSS ground truth, with values typically ranging from 1.6 to 2.0 meters across the sequence. A range of error consistent with the quantitative and field testing trials.

As illustrated in the workflow of Fig. 3.12, once the skyline-detection stage is completed using the Gamma method (DeepLab V3 semantic segmentation), the execution module initiates localization by extracting the skyline contour and binary mask from the first frame. As formalized in Stage 1 of Algorithm 3.2.2 and depicted in Fig. 3.19, the system compares this detected skyline against a logged georeferenced groundtruth skyline dataset $\mathcal{G}$ by doing the estimation of Dice similarity and Hausdorff distance for each candidate location. This estimation produces the initial geographic anchor $(\text{lat}_0, \text{lon}_0, \psi_0)$, which is then visualized in Fig. 3.20. In the absence of groundtruth, this work highlights a dual consequence. A primary strength of this work is that orientation estimation remains agnostic to groundtruth, which validates the potential for purely visual navigation. And also, a system limitation exists regarding absolute georeferencing and metric displacements: while relative skyline-based odometry runs, recovering the metric scale and to set the initial ENU anchor would require alternative initialization methods, such as

Algorithm 3.2.2 Skyline-Based ICP-EKF Localization and Trajectory Generation

Require: Dataset $\mathcal{D}$ with N frames and ground-truth skylines $\mathcal{G}$
Require: Camera intrinsics: $f_x = 489.07$ px, skyline depth $d_{\text{avg}} = 100$ m
Require: Map alignment: θ_{align} [deg], d_x, d_y [m]
Require: EKF parameters: Q (process noise), R (measurement noise)
Ensure: Fused trajectory: $(x_{\text{EKF}}, y_{\text{EKF}}, \theta_{\text{EKF}})$ in map frame

Stage 1: Initial Localization via Skyline Matching
▷ Extract skyline from first frame using Gamma method (DeepLab V3)
$S_0 \leftarrow \text{ExtractSkyline}(\mathcal{D}[0])$
$\mathcal{P}_0 \leftarrow \text{SkylineToPointCloud}(S_0)$
▷ Find best match in ground-truth dataset
$(\text{lat}_0, \text{lon}_0, \psi_0) \leftarrow \text{argmax}_{g \in \mathcal{G}} \left[\text{Dice}(S_0, g) - \alpha \cdot \text{Hausdorff}(S_0, g)\right]$
▷ Initialize EKF state in local ENU tangent plane
$\mathbf{X}_0 \leftarrow [0, 0, \psi_0]^\top$ ▷ Initial pose: (x, y, θ)
$\mathbf{P}_0 \leftarrow \text{diag}([1.0, 1.0, (10\pi/180)^2])$ ▷ Initial covariance

Stage 2: Metric Scale Derivation from Camera Geometry
▷ Compute pixel-to-metric scale factor (Eq. 13)
$\lambda_{\text{scale}} \leftarrow d_{\text{avg}} / f_x \approx 0.204$ m/px

Stage 3: ICP-Based Odometry in Metric Space
for $k = 1$ **to** N **do**
▷ Extract skyline and convert to point cloud
$S_k \leftarrow \text{ExtractSkyline}(\mathcal{D}[k])$
$\mathcal{P}_k \leftarrow \text{SkylineToPointCloud}(S_k)$
▷ ICP registration in pixel space
$\mathbf{T}^k_{\text{ICP}}, f^k_{\text{fitness}} \leftarrow \text{ICP}(\mathcal{P}_{k-1}, \mathcal{P}_k)$
$\mathbf{t}^k_{\text{pixel}} \leftarrow [t_x, t_y]^\top$ from $\mathbf{T}^k_{\text{ICP}}$
$\theta^k_{\text{ICP}} \leftarrow \text{atan2}(r_{21}, r_{11})$ from $\mathbf{T}^k_{\text{ICP}}$
▷ Convert to metric displacement (Eq. 16)
$\mathbf{t}^k_{\text{metric}} \leftarrow \lambda_{\text{scale}} \cdot \mathbf{t}^k_{\text{pixel}}$ [meters]
▷ GNSS measurement in local ENU (Eq. 17)
$x^k_{\text{GNSS}} \leftarrow R_{\text{earth}} \cdot (\text{lon}_k - \text{lon}_0) \cdot \cos(\text{lat}_0)$
$y^k_{\text{GNSS}} \leftarrow R_{\text{earth}} \cdot (\text{lat}_k - \text{lat}_0)$
$\mathbf{z}_k \leftarrow [x^k_{\text{GNSS}}, y^k_{\text{GNSS}}]^\top$
▷ EKF Update Step
$\mathbf{X}_k, \mathbf{P}_k \leftarrow \text{EKF_Update}(\mathbf{X}^-_k, \mathbf{P}^-_k, \mathbf{z}_k)$
$\mathbf{X}_k[2] \leftarrow \text{wrap}(\mathbf{X}_k[2])$ ▷ Normalize heading to $[-\pi, \pi]$
end for

Map Frame Alignment for Visualization
▷ Prepare rotation matrix for alignment
$\theta_{\text{rot}} \leftarrow \text{radians}(\theta_{\text{align}})$
$\cos_\theta \leftarrow \cos(\theta_{\text{rot}})$, $\sin_\theta \leftarrow \sin(\theta_{\text{rot}})$
for $k = 0$ **to** N **do**
▷ Transform EKF trajectory to map frame
$x^k_{\text{map}} \leftarrow \cos_\theta \cdot \mathbf{X}_k[0] - \sin_\theta \cdot \mathbf{X}_k[1] + d_x$
$y^k_{\text{map}} \leftarrow \sin_\theta \cdot \mathbf{X}_k[0] + \cos_\theta \cdot \mathbf{X}_k[1] + d_y$
▷ Compute heading from trajectory direction
if $k > 0$ **then**
$\psi^k_{\text{map}} \leftarrow \text{atan2}(y^k_{\text{map}} - y^{k-1}_{\text{map}}, x^k_{\text{map}} - x^{k-1}_{\text{map}})$
else
$\psi^k_{\text{map}} \leftarrow \theta_{\text{rot}}$
end if
▷ Publish ROS messages for visualization
Publish Odometry$(x^k_{\text{map}}, y^k_{\text{map}}, \psi^k_{\text{map}})$ on `/icp_ekf_odom`
Publish TF: `map` → `base_link`$(x^k_{\text{map}}, y^k_{\text{map}}, \psi^k_{\text{map}})$
Append to Path on `/icp_ekf_path`
▷ Also transform GNSS for comparison
$x^k_{\text{gnss_map}} \leftarrow \cos_\theta \cdot x^k_{\text{GNSS}} - \sin_\theta \cdot y^k_{\text{GNSS}} + d_x$
$y^k_{\text{gnss_map}} \leftarrow \sin_\theta \cdot x^k_{\text{GNSS}} + \cos_\theta \cdot y^k_{\text{GNSS}} + d_y$
Append to Path on `/gnss_path`
end for
return $(x_{\text{EKF}}, y_{\text{EKF}}, \theta_{\text{EKF}})$ and fitness scores $\{f^k_{\text{fitness}}\}$

manual specification or motion sensors. These relative motion estimates are then fused via an Extended Kalman Filter to produce the final localization $(x_{\text{EKF}}, y_{\text{EKF}}, \theta_{\text{EKF}})$.

The skyline is first detected in 2D image space, then converted to a metric point cloud via the scale factor λ. However, to enable comparative evaluation against GNSS ground truth and visualization over satellite imagery, both trajectories must be expressed in a common reference frame. As shown in Fig. 3.21(a), the *NavSatTransformNode* [42] converts GPS measurements from WGS84 (EPSG:4326) into a local East-North-Up (ENU) Cartesian frame—the same local tangent plane in which the ICP-EKF trajectory is constructed. This coordinate transformation is formalized in Stage 4 of Algorithm 3.2.2, where the fused trajectory $(x_{\text{EKF}}, y_{\text{EKF}}, \theta_{\text{EKF}})$ is rotated and translated to align with the RTAB-Map coordinate convention for overlay visualization in *RVizSatellite* [43].

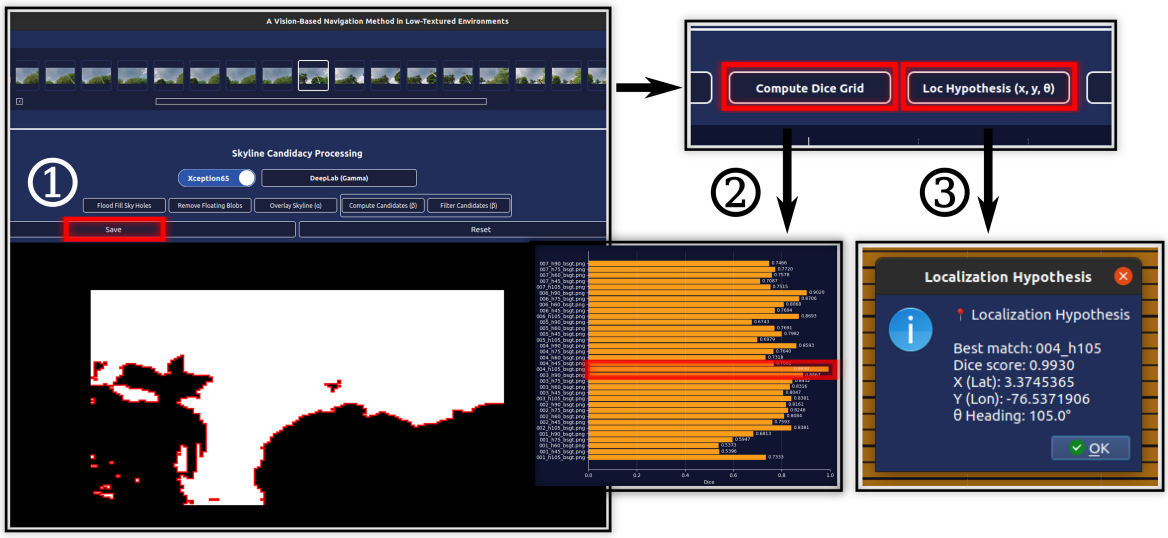
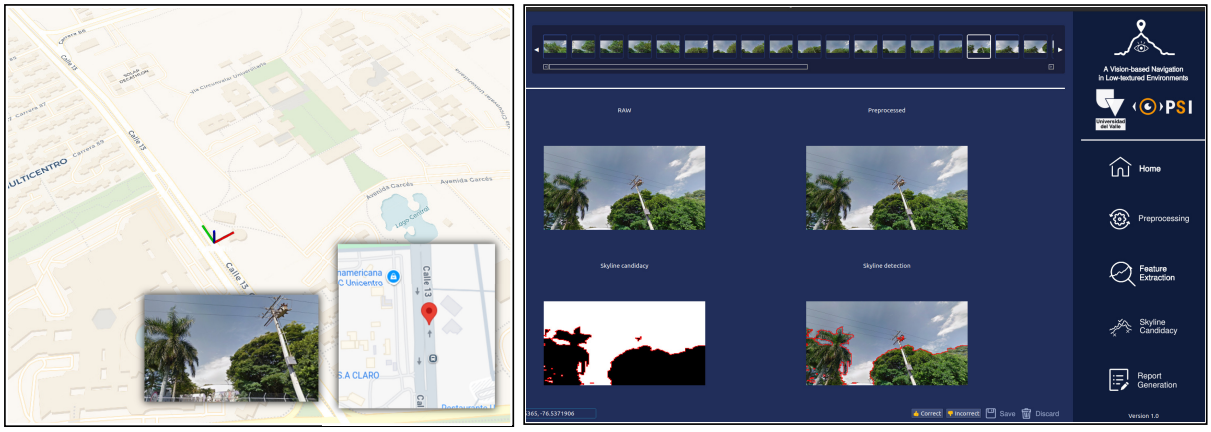


Figure 3.19: Interface of the Localization Process and workflow for initial pose estimation. The system executes Stage 1 of the pipeline by extracting the skyline contour from the current frame and comparing it against the ground-truth dataset. Dice Score and Hausdorff distance identify the optimal match, which establishes the initial localization hypothesis (x, y, θ) required to initialize the EKF state vector



(a) **(b)**

Figure 3.20: Visualization of the localization pipeline. (a) The estimated position projected onto satellite imagery using the NavSat and RVizSatellite packages, which shows the alignment between the estimated path and the physical environment. (b) The Execution Module GUI, it displays the sequential processing chain of raw input, preprocessing, and skyline detection, used to generate the visual odometry inputs for the fusion algorithm

With the accumulation of scaled ICP increments from Stage 3 (Algorithm 3.2.2), the system constructs a local pose-graph in which each node represents a skyline acquisition instant and each edge encodes an ICP-derived relative transform $\mathbf{T}_{i,i+1}$ in metric coordinates. As formalized in such algorithm, this process aligns each detected skyline with its georeferenced counterpar to establish the initial metric anchor, then computes consecutive pairwise registrations to build the edge. The resulting pose-graph defines a locally consistent trajectory within the ENU tangent plane.

To enable quantitative evaluation against GNSS ground truth and visualization over satellite imagery, both trajectories must be expressed in a common reference frame. As shown in Fig. 3.21 (a), the GPS measurements, originally in WGS84 (EPSG:4326), are converted to planar WebMercator/ENU metric coordinates via the *NavSatTransformNode* [42], with the first GPS fix defining the local reference origin.

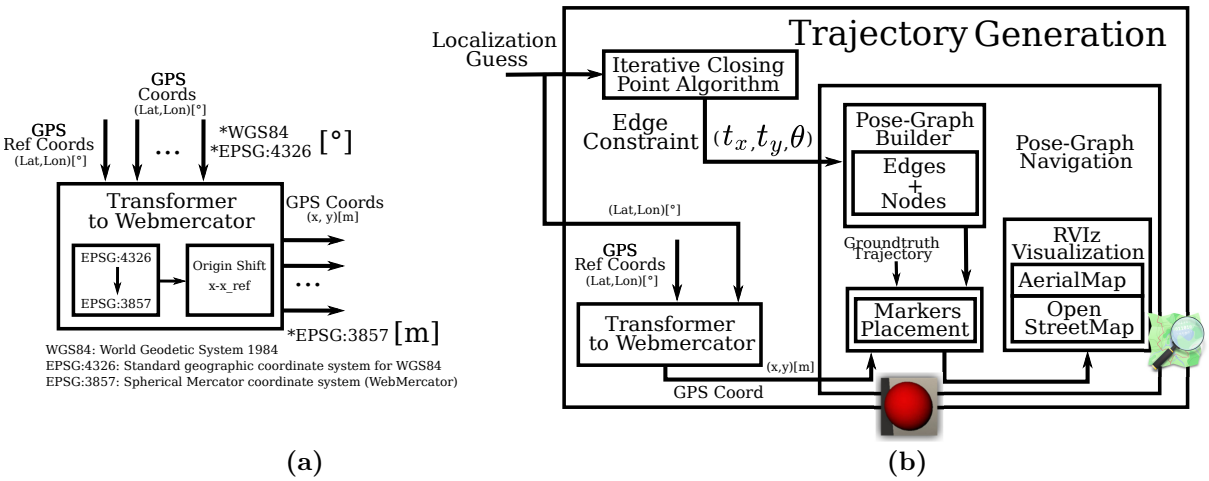


Figure 3.21: Trajectory generation and visualization pipeline. (a) GPS reference coordinates in WGS84 (EPSG:4326) are converted to planar WebMercator metric coordinates (EPSG:3857) and shifted to a local frame via *NavSatTransformNode*. (b) The skyline-based pose-graph trajectory, obtained from ICP edge constraints in the local ENU frame, is visualized alongside the transformed GPS ground truth as markers over OpenStreetMap using the RViz AerialMap plugin. Both trajectories are aligned via Eq. 19 solely for comparative visualization

The skyline-based pose-graph, constructed incrementally in its own local metric frame via ICP, remains independent of this GPS coordinate system, as in 19 and the stage 4 of Algorithm 3.2.2 (Lines 38-50).

$$\begin{aligned} x_{\text{map}} &= \cos\theta_{\text{align}} \cdot x_{\text{EKF}} - \sin\theta_{\text{align}} \cdot y_{\text{EKF}} + d_x \\ y_{\text{map}} &= \sin\theta_{\text{align}} \cdot x_{\text{EKF}} + \cos\theta_{\text{align}} \cdot y_{\text{EKF}} + d_y \end{aligned} \tag{19}$$

where θ_{align}, d_x, and d_y are user-configured parameters that register the ICP-EKF trajectory to the visualization coordinate frame used by RViz and satellite overlay tools.

Finally, this alignment makes both trajectories reside within the same Local Tangent Plane (LTP) approximation [41], which remains accurate for trajectories much more smaller than Earth's radius (< 1 km, satisfied by all datasets in this work) and at mid-to-low latitudes (tested at 3.37°N). Under these conditions, the Euclidean distance between corresponding poses in the aligned frame directly reflects a physical position error, which enables fair comparison between the skyline-based odometry and the GPS ground truth without introducing projection-induced distortions.

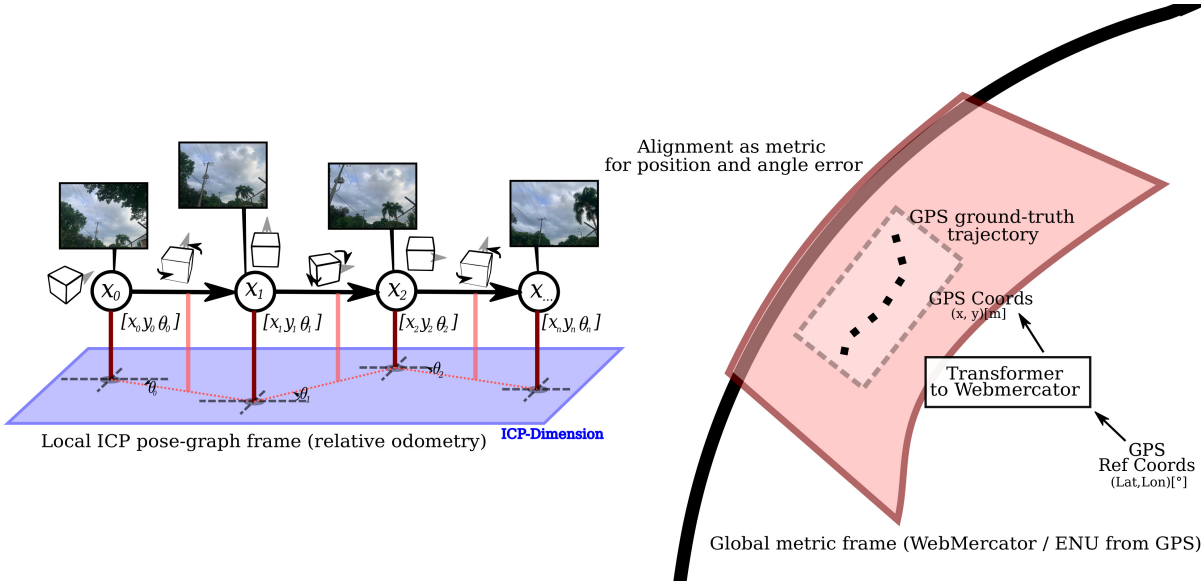


Figure 3.22: Skyline-based pose-graph (left) and GPS ground-truth trajectory (right) after coordinate frame alignment. The skyline pipeline produces relative motion estimates through ICP, while GPS provides absolute georeferenced positions. The alignment via Stage 4 enables direct measurement of position and orientation error without altering the intrinsic trajectory geometry

3.2.3 RTAB-Map as an Independent Evaluation Baseline

Algorithm 3.2.4 RTAB-Map Offline Trajectory Generation (Independent Baseline)

Require: Raw image sequence $\mathcal{I} = \{I_0, I_1, \ldots, I_N\}$ (same as used for skyline pipeline)
Require: Camera calibration: K (intrinsic matrix), depth estimation model
Ensure: RTAB-Map trajectory: $(x_{\text{RTAB}}, y_{\text{RTAB}}, \theta_{\text{RTAB}})$ in map frame

Stage 1: Visual Feature Extraction
for $k = 0$ **to** N **do**
  $\mathcal{F}_k \leftarrow \text{ExtractFeatures}(I_k)$ ▷ SIFT/ORB descriptors from full image
  $D_k \leftarrow \text{EstimateDepth}(I_k)$ ▷ Learned monocular depth estimation
end for

Stage 2: Frame-to-Frame Visual Odometry
for $k = 1$ **to** N **do**
  $\text{matches} \leftarrow \text{MatchFeatures}(\mathcal{F}_{k-1}, \mathcal{F}_k)$
  $\mathbf{T}_k^{\text{VO}} \leftarrow \text{EstimateMotion}(\text{matches}, D_{k-1}, D_k)$ ▷ PnP + RANSAC
  Add edge $(k-1 \rightarrow k, \mathbf{T}_k^{\text{VO}})$ to pose-graph
end for

Stage 3: Loop Closure Detection and Graph Optimization
$\mathcal{L} \leftarrow \text{DetectLoopClosures}(\{\mathcal{F}_k\}, \text{appearance_threshold})$
for each detected loop $(i \leftrightarrow j) \in \mathcal{L}$ **do**
  $\mathbf{T}_{ij}^{\text{loop}} \leftarrow \text{ComputeLoopConstraint}(\mathcal{F}_i, \mathcal{F}_j, D_i, D_j)$
  Add edge $(i \leftrightarrow j, \mathbf{T}_{ij}^{\text{loop}})$ to pose-graph
end for
$\{(x_k, y_k, \theta_k)\} \leftarrow \text{OptimizePoseGraph}(\text{all edges})$ ▷ g2o optimization

Stage 4: Convert to Metric ENU Frame (for comparison with GNSS)
Align RTAB-Map trajectory to local ENU origin (same as ICP-EKF pipeline)
return $(x_{\text{RTAB}}, y_{\text{RTAB}}, \theta_{\text{RTAB}})$

The skyline-based ICP-EKF pipeline presented in Sections 3.2.2 transforms visual silhouettes into metric georeferenced poses through a three-stage process: (1) pixel-space ICP alignment, (2) camera-geometry-based metric scaling, and (3) EKF fusion with GNSS in a local tangent plane. To validate this approach, the evaluation framework includes two independent baseline comparisons: GNSS as the absolute ground truth, the proposed ICP-EKF approach from this work, and RTAB-Map as a continuous visual odometry reference.

Unlike the discrete, skyline-based approach, RTAB-Map [44] operates on the full image sequence to produce dense frame-to-frame visual odometry with loop closure detection and graph-based trajectory optimization. RTAB-Map does not use skyline features, it relies instead on feature rich image regions (SIFT, ORB descriptors) and a learned depth estimation model to build a globally consistent 3D map and trajectory, as formalized in Algorithm 3.2.4. This fundamental difference in feature extraction, depth recovery, and spatial representation makes RTAB-Map an independent baseline for the assessment of the geometric consistency and drift characteristics of the skyline-based method.

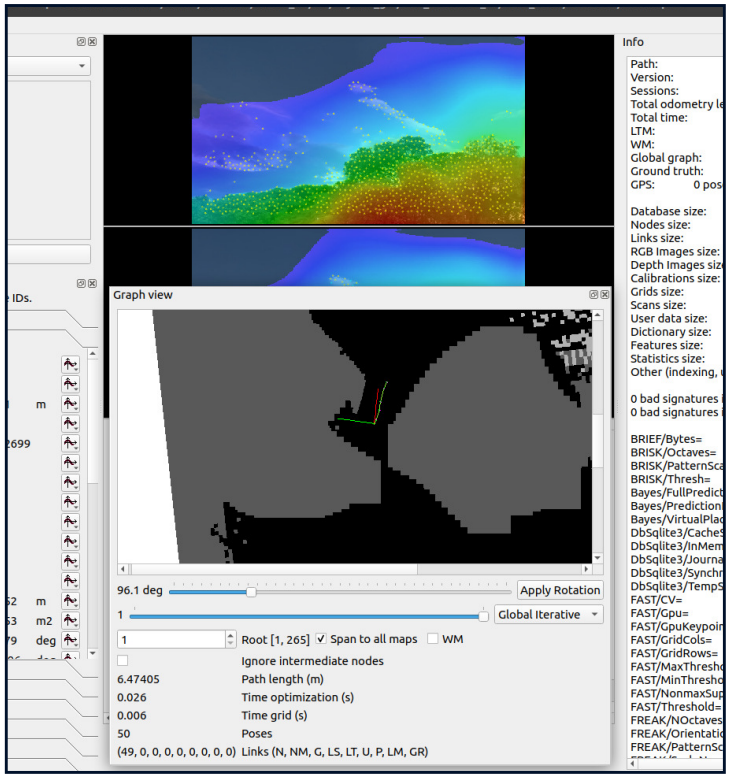


Figure 3.23: RTAB-Map executed offline on the raw image sequence. It produces an independent trajectory and occupancy-grid map that serve as a secondary reference for evaluating the skyline-based odometry

In this work, RTAB-Map is not used as an online odometry engine during data acquisition. Instead, it is executed offline on the same raw image sequences used by the skyline pipeline, which operates as a post-processing tool that reconstructs the complete trajectory and perform localization after the fact. This offline execution mode allows RTAB-Map to leverage its full graph optimization and loop closure capabilities without real-time constraints, and this produces a drift-corrected reference trajectory and occupancy-grid map (Fig. 3.23).

The resulting RTAB-Map trajectory provides several key capabilities: it offers frame-to-frame motion estimates, derived from dense visual features (rather than skyline-based features), performs global trajectory optimization through loop closure and subsequent pose-graph refinement, enables drift behavior analysis, which is independent of the quality of skyline segmentation, and generates a depth-aware spatial reconstruction facilitated by the provided Deep Learning Monocular Depth Estimation Model for monocular depth estimation. This framework ends in a three-way evaluation process: first, ICP-EKF

(Skyline-Based) vs. GNSS measures the absolute position error of the proposed method against the ground truth; second, RTAB-Map vs. GNSS measures the absolute position error of the dense visual odometry baseline; and finally, ICP-EKF vs. RTAB-Map performs a comparative analysis of geometric consistency as shown in Section 4.3.

3.2.4 Report Module

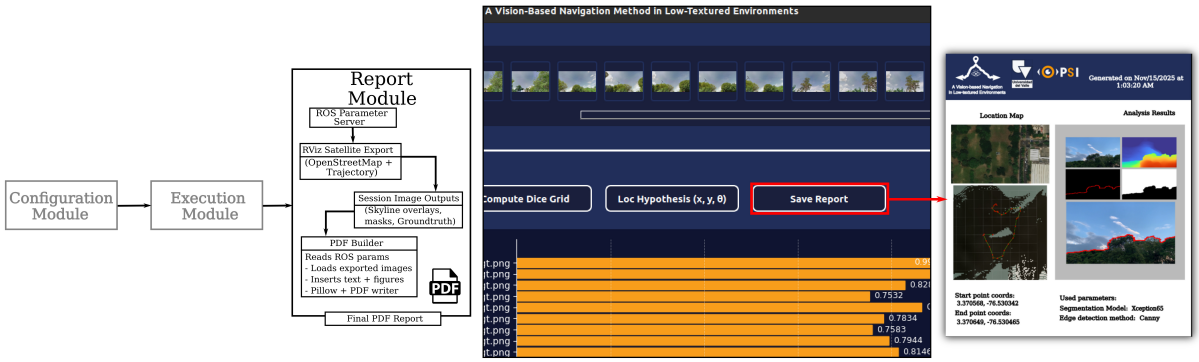


Figure 3.24: Report Module workflow. (a) Internal pipeline showing how ROS parameters, RViz Satellite exports, and session image outputs are collected and passed to a lightweight Python PDF builder to generate the final report. In (b), User interface view, which illustrates the preview of processed images, the 'Save Report' option acts as a trigger to start the processing of report generation

The architecture of the Report Module follows a structured workflow designed to consolidate every element generated during a localization session. As illustrated in Fig. 3.24, the module acts as the final stage that receives all relevant outputs from the Configuration and Execution modules, including the synchronized parameters stored in the ROS Parameter Server, the satellite-based visualizations exported from RViz, and the collection of processed images produced throughout the session. By organizing these heterogeneous inputs into a single coherent flow, the system ensures that all numerical values, overlays, trajectories, and configuration settings are available in a unified and reproducible form, ready to be assembled into a formal report.

In summary, the reporting pipeline transforms the complete session state into a portable document. The ROS Parameter Server guarantees that configuration values remain consistent across the entire workflow, while the RViz satellite export provides geospatial context for the reconstructed trajectory. These, together with the session image outputs, are collected by the PDF builder, which inserts text, figures, and metadata into a structured template using only minimal Python dependencies. The result is a final PDF report that reflects the experiment, which enables traceability, and evaluation.

3.3 Conclusions

This chapter presented the design and implementation of the Execution Module, which constitutes the operational core of the skyline-based localization system. The module integrates vision-based segmentation with data fusion to transform raw skyline-detected images sequences into georeferenced trajectory estimates through a processing pipeline.

The Execution Module operates through four interconnected stages: (1) skyline extraction using the Gamma method (DeepLab V3 semantic segmentation), which consistently achieves Dice scores above 96%; (2) initial pose estimation via comparison against the georeferenced ground-truth dataset using Dice similarity and Hausdorff distance metrics; (3) trajectory generation through Iterative Closest Point (ICP) registration, which computes frame-to-frame motion increments in pixel space before scaling to metric units via the camera-geometry-derived scale factor λ; and (4) fusion with GNSS measurements

through an Extended Kalman Filter (EKF) operating in a local East-North-Up (ENU) tangent plane.

The mathematical formulation presented in this chapter establishes how skyline silhouettes are converted into metric odometry constraints for pose-graph navigation. Algorithm 3.2.2 formalizes the complete offline pipeline, which includes coordinate frame alignment and ROS message publication, while Algorithm 3.2.1 details the ICP execution stage that produces the edge constraints for trajectory estimation. The scale factor derivation, based on the pinhole camera model and far-field depth assumption, enables the conversion from pixel displacements to metric increments without requiring direct depth measurements. Again, under the far-field depth assumption given the distances of the skylines in this work.

The integration of this module with the Configuration and Report modules, as illustrated in the system architecture (Fig. 3.4), ensures consistent parameter propagation through the ROS Parameter Server. The visualization capabilities provided by NavSatTransformNode and RVizSatellite allow direct comparison between the ICP-EKF trajectory and GNSS ground truth over satellite imagery, which facilitates both qualitative assessment and quantitative error analysis.

In summary, the Execution Module bridges the gap between visual perception and metric localization through a systematic approach to extract geometric constraints from skyline features. The modular design allows independent development and testing of each processing stage, while the standardized interfaces ensure integration with the broader software framework. The quantitative evaluation of this pipeline, including position and orientation error analysis under real field conditions, is presented in Chapter 4.

4 Results

This section presents the validation of the proposed system across three dimensions: integration, quantitative, and field testing. The integration results confirm that the software meets all functional requirements defined for each module of the graphical interface and validated the correct interaction among preprocessing, feature extraction, skyline candidacy, and reporting components. Quantitative results analyze the performance of the skyline-based localization method, measuring its accuracy through metrics such as Hausdorff distance and Dice coefficient to assess correspondence between predicted and reference skylines, which also has an effect on the geolocation hypothesis. Finally, the field results demonstrate the method's behavior under real environmental conditions by using panoramic image inputs, providing a preliminary validation of its applicability beyond controlled datasets.

4.1 Integration Testing

The integration testing focused on the verification for each component of the developed system, to operate cohesively within the overall workflow. The main objective was to ensure that every functional requirement established, such as image preprocessing, feature extraction, skyline generation, and performance evaluation executes correctly and interacts without conflicts. Each module was validated through a set of defined test cases, with proper data flow between panels and consistent synchronization of parameters through ROS. The results show that the integration process was successfully completed. The graphical interface allowed full operation of all modules and the expected outputs displayed at each processing stage.

1 - Configuration Stage

System Initialization, Dataset Handling, and Preprocessing

Type of Test: This test validates all configuration-level functionalities, including GUI initialization, dataset loading, metadata association, file integrity validation, preprocessing, and parameter synchronization across panels. It covers dataset picker operation, GUI responsiveness, metadata parsing, preprocessing pipelines (Gaussian, Median, Bilateral filters, CLAHE), morphological filtering, automatic Region Of Interest (ROI) definition, and the ability to adjust image parameters (brightness, contrast, exposure). The goal is to ensure that all configuration steps correctly prepare the images and metadata for the skyline extraction stage.

Required Hardware: Desktop or laptop computer with at least 8 GB RAM, running Ubuntu 20.04 LTS or later.

Required Software:

- ROS Noetic
- Python 3.8+ with PyQt5, OpenCV, NumPy
- Skyline GUI application (current version 1.0)
- Scikit-Image 0.21.0, Scipy 1.10.1

Objective: To verify that all configuration operations, from dataset loading to preprocessing, execute without conflicts, generate uniform inputs, and propagate parameters consistently through the GUI and ROS nodes.

Test Data: A directory that contains valid dataset folders, each with sample skyline image sequences, and binary segmented ground-truth masks.

Procedure:

1. Launch the Skyline GUI.
2. Enable Dataset Picker and load both valid and corrupted datasets.
3. Validate automatic parsing and association of metadata files.
4. Confirm image resizing, ROI cropping, and adaptive thresholding.
5. Apply Gaussian filters; toggle brightness and contrast sliders.
6. Apply CLAHE and morphological operations (opening, closing, dilation, erosion).

7. Switch among panels (Home, Preprocessing, Feature Extraction) to verify synchronization.
8. Save and reopen configuration to confirm persistent settings.
9. Check ROS console logs for confirmation of parameter updates.

Expected Result: All images are standardized, metadata correctly linked, GUI responsive, and preprocessing filters applied consistently. Any corrupted files trigger a readable error dialog. The resulting preprocessed images exhibit enhanced contrast, reduced noise, and a defined skyline region of interest.

Obtained Result: The GUI successfully loaded and synchronized datasets, automatically resized images, applied filters, and preserved session parameters. Corrupted or missing files prompted appropriate error messages. The preprocessing stage produced uniform, high-contrast images suitable for skyline extraction. (see Integration Testing folder)

Comments: All configuration routines operated as intended. Future improvements could include checksum-based validation and numeric entry for fine-tuning preprocessing parameters.

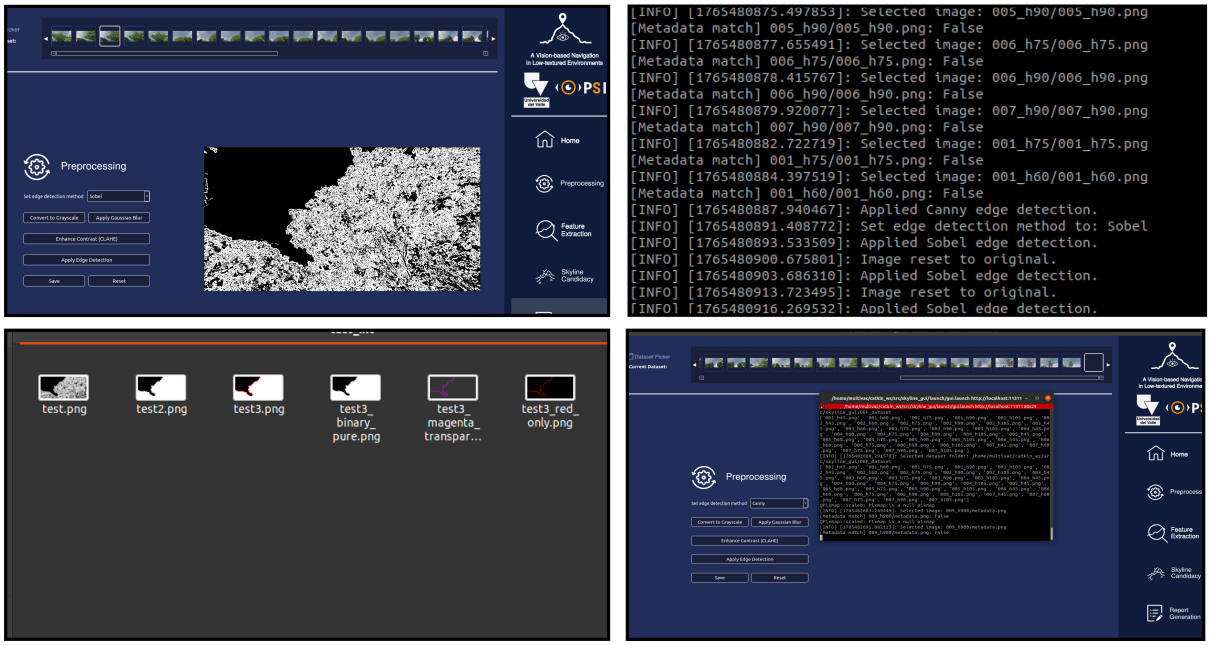


Figure 4.1: Configuration Stage integration testing screenshots. Top-left: GUI main interface showing the preprocessing panel with dataset carousel, filter options, and navigation sidebar. Top-right: ROS console output displaying metadata matching logs, image selection confirmations, and edge detection method applications (Canny/Sobel). Bottom-left: Dataset picker view showing available preprocessed image files with binary masks and transparency variants. Bottom-right: Preprocessing panel displaying the active dataset with thumbnail error image in the carousel and the message in the Command Line Interface

2 - Execution Stage

Feature Extraction, Skyline Detection, and Validation

Type of Test: This test consolidates all functional evaluations of the execution stage, which includes feature extraction, skyline candidate computation, classification, and validation. It covers contour segmentation, candidate ranking, Machine Learning model for skyline-detection, and skyline continuity validation. The purpose is to verify that skyline detection methods (Alpha, Beta, Gamma) operate correctly, that the system prioritizes accurate candidates, and that false skylines (clouds, occlusions, noise) are rejected.

Required Hardware: Desktop or laptop computer with at least 8 GB RAM, running Ubuntu 20.04 LTS or later.

Required Software:

- Ubuntu 20.04 LTS or later
- ROS Noetic
- Python 3.8+ with PyQt5, OpenCV, NumPy
- Skyline GUI v1.0

Objective: Ensure the execution pipeline correctly detects, ranks, and validates skyline contours with integration of dynamic programming (Alpha), geometric heuristics (Beta), and DeepLab V3 semantic segmentation (Gamma) methods within the GUI environment.

Test Data: A dataset of skyline images in standard lighting conditions.

Procedure:

1. Load the preprocessed dataset.
2. Select a skyline detection method (Alpha, Beta, or Gamma).
3. Execute skyline computation.
4. Validate contour segmentation and morphological refinement.
5. Apply ranking heuristics to prioritize continuous, high-contrast skylines.
6. Run Machine-Learning classification for skyline vs non-skyline edges.
7. Compare detected skyline against ground-truth using Dice/Hausdorff metrics.
8. Visually confirm red-only predicted skyline overlay and magenta ground-truth alignment.

Expected Result: All methods execute without crashes; overlays display correctly. Dice coefficients exceed 90% for valid detections, with smooth continuity and minimal false positives. Discontinuous or noisy skylines are rejected automatically.

Obtained Result: Execution of Alpha, Beta, and Gamma pipelines completed successfully. Gamma achieved highest Dice 97%, Alpha/ Beta averaged 92%. False skyline segments due to clouds were effectively filtered. GUI indicators updated correctly during processing.

Comments: Execution stage validated end-to-end integration. Future work may include automated model selection and confidence-based adaptive weighting among methods.

3 - Reporting Stage

Results Export, Auto-Save, and Session Management

Type of Test: This test verifies the full reporting workflow that consolidates processed data, quantitative results, and user configurations. It verifies YAML/JSON/CSV/PDF report generation, periodic auto-save, and user confirmation on session closure. It ensures that numerical metrics (Dice, Hausdorff), configuration parameters, and processed image overlays are saved consistently.

Required Hardware: Desktop or laptop computer with at least 8 GB RAM, running Ubuntu 20.04 LTS or later.

Required Software:

- Ubuntu 20.04 LTS or later
- ROS Noetic
- Python 3.8+ with PyQt5, OpenCV, NumPy
- Skyline GUI v1.0

Objective: Validate that the reporting subsystem generates correct structured outputs and preserves progress at the moment of doing experiments.

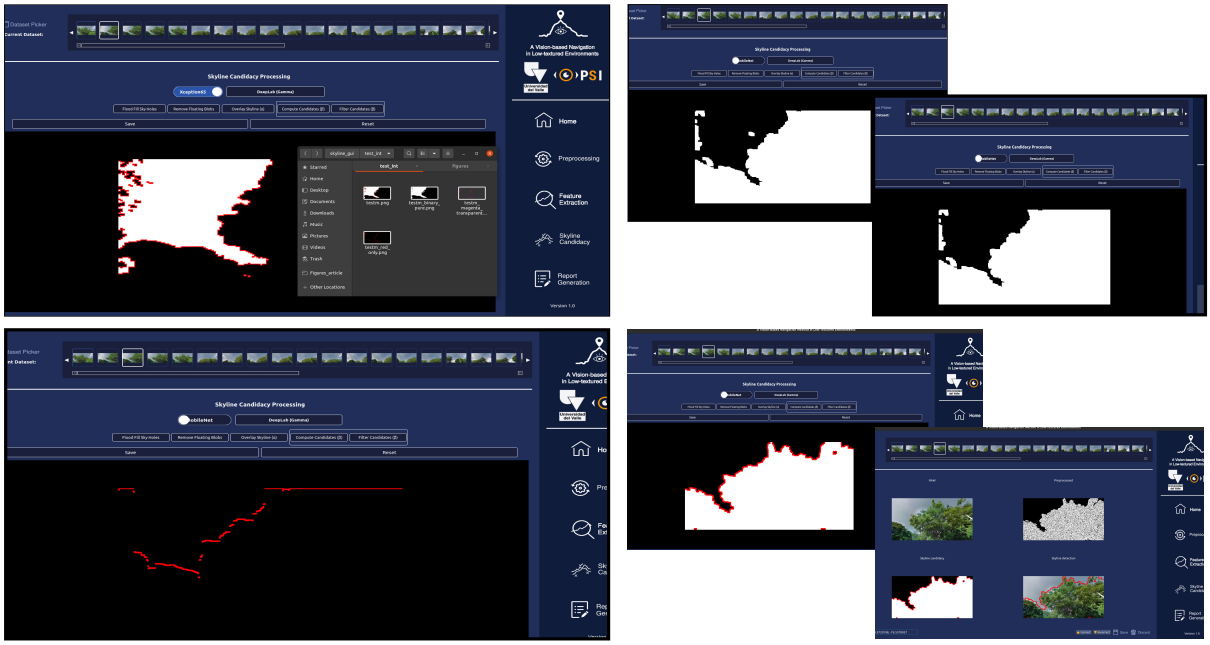


Figure 4.2: Execution Stage integration testing screenshots. Top-left: Region masking and processing panel showing a binary skyline mask with adjustment controls for brightness, contrast, blur, and edge detection parameters after doing the Gamma pipeline. Top-right: Skyline candidacy processing window displaying the extracted binary silhouette with validation controls from Alpha pipeline. Bottom-left: Skyline candidate visualization panel with the refined contour extraction result overlaid on a dark background from Beta method. Bottom-right: Report annotation interface showing the predicted skyline overlay (red contour) on the original image after Alpha method, alongside sample dataset thumbnails and vegetation reference images for visual comparison

Test Data: Completed sessions from the Execution stage containing processed images, metric results, and configuration logs.

Procedure:

1. Perform a full skyline-based localization session.
2. Trigger "Save Session" and choose report format (YAML/JSON/XML).
3. Wait five minutes to confirm automatic backup creation.
4. Close the session and verify confirmation dialog (Save / Discard).
5. Reopen saved report to confirm inclusion of all relevant data.
6. Cross-check exported overlays (red vs magenta skylines) for visual consistency.

Expected Result: The system generates a complete report containing paths, parameters, metrics, and image outputs. Auto-save functions at 5-minute intervals. Session closure dialog prevents accidental loss.

Obtained Result: Reports exported successfully in PDF, including skyline metrics and overlay paths. Auto-save verified at defined interval. User prompt correctly displayed on closing.

Comments: Reporting stage fully functional. Future versions could include summary statistics aggregation and cloud synchronization.

The integration tests demonstrated that the three core modules of the system Configuration, Execution, and Report interact consistently across the full processing pipeline. Parameter propagation from the Configuration panel to the Execution engine occurred without conflicts, which confirms the ROS Parameter Server and the GUI layer are correctly synchronized. The Execution module processed the selected images, produced skyline predictions, and generated intermediate outputs exactly as expected.



Figure 4.3: Reporting Stage integration testing screenshot. Generated PDF report preview shows the complete session output: header with timestamp, location map panel displays satellite imagery with the georeferenced trajectory overlay and start/end coordinates, and analysis results panel that contains depth colormap visualization, binary skyline mask, original scene with detected skyline contour (red), and vegetation reference image. The report consolidates segmentation model parameters (Xception65) and edge detection method (Canny) used for this session

This enabled the Report module to retrieve and display results without discrepancies. The observed behavior across all test cases indicates that the system components operate cohesively, with data flowing as intended between stages and without evidence of interface mismatch. Overall, the integration testing validates that the system is successfully operational.

4.2 Quantitative Testing

This section presents the quantitative evaluation of the skyline-based localization pipeline through numerical analysis. The evaluation follows a systematic approach: Section 4.2.1 describes the dataset construction methodology, including the monocular video capture and synchronized GNSS logging that provide the ground-truth reference for all analyses. Section 4.2.2 validates the camera-geometry-derived scale factor $\lambda = 0.204$ m/px against empirical measurements obtained from the GNSS trajectory. The quantitative metrics established in this section includes ICP fitness scores, iteration counts, angular errors, and position RMSE, which positions the next section, which is the field testing evaluation presented in Section 4.3, where the complete pipeline is evaluated under real outdoor conditions.

4.2.1 Data Acquisition

Regarding the construction of the dataset, a monocular camera was used to record uninterrupted sequences while the GNSS module simultaneously logged global position. The resulting video stream typically contains between 2000 and 7000 frames per dataset, in function on the traversal length. Each frame is later log stored in a root-level. This enables high-resolution skyline tracking across dense sequences rather than isolated viewpoints.

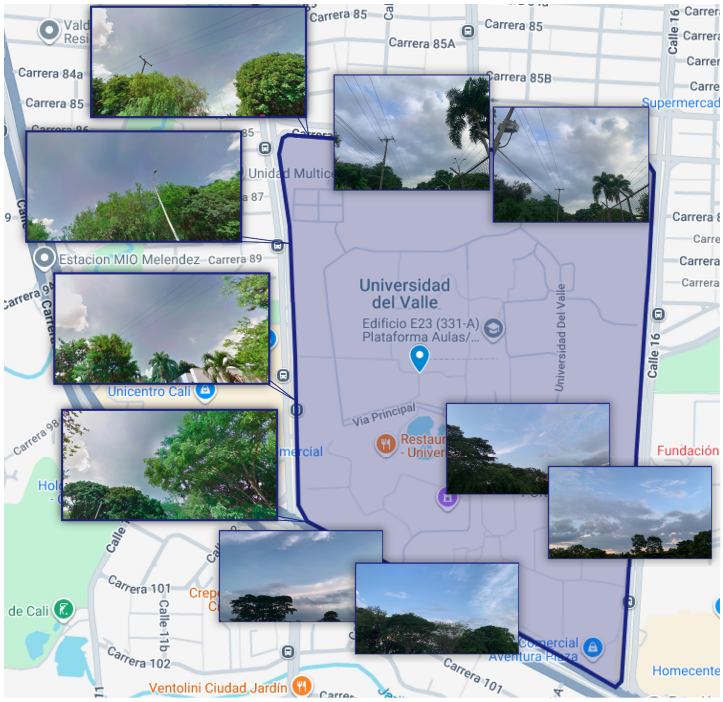


Figure 4.4: Dataset acquisition process around and within the Universidad del Valle campus, Cali

Algorithm 4 Continuous Video–Based Skyline Dataset Generation Pipeline

Require: Video file V, GNSS log $\mathcal{G}$

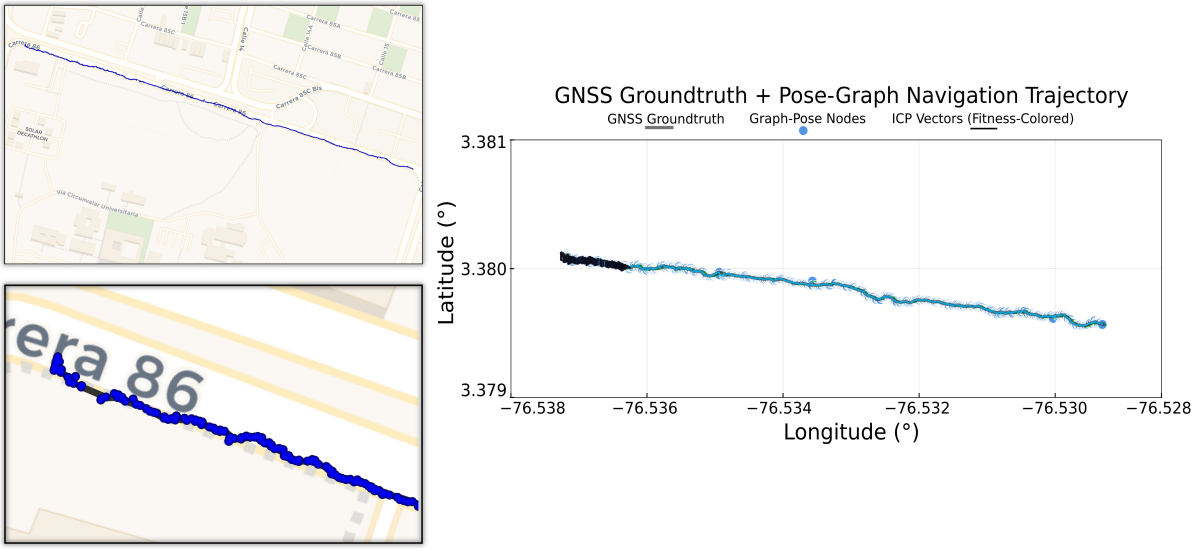
frames $\leftarrow$ ExtractFrames(V) ▷ Typically 2000–7000+ frames
for each frame F_k in frames **do**
  $t_k \leftarrow$ Timestamp(F_k)
  SaveRawFrame(F_k, Lat_k, Lon_k, Yaw_k)
  $B_k \leftarrow$ ComputeBinaryMask(F_k)
  $C_k \leftarrow$ SkylineCandidacy(B_k)
  $P_k \leftarrow$ GeneratePointCloud(C_k)
end for
for each consecutive pair (F_k, F_{k+1}) **do**
  $(R_k, t_k) \leftarrow$ ICP(P_k, P_{k+1})
  $\epsilon_k^{pos}, \epsilon_k^{yaw} \leftarrow$ ComputeLocalizationErrors($R_k, t_k, \mathcal{G}$)
  $d_k \leftarrow$ MonocularDepthEstimation(F_k)
  LogResults($k, \epsilon_k^{pos}, \epsilon_k^{yaw}, d_k$)
end for
return Complete dataset: raw frames, metadata CSV, skyline masks, point clouds, ICP logs

For the dataset, the frames were collected using the same video-based pipeline described in the algorithm 4, captured directly on site using a monocular camera. A GNSS receiver logger operated in parallel, recording latitude, longitude, and yaw at high frequency. After acquisition, the yaw angle remains the most relevant variable for skyline analysis, as it determines the horizontal displacement of the skyline profile in the image and directly influences the evaluation across frames, wether inliers or/and ICP. Each local dataset also contains between 2000 and 7000 frames. The previous fact adds temporal density for skyline extraction, monocular depth estimation, and ICP-based localization experiments.

Dataset Organization: Both the remote and local datasets follow a unified folder structure in which each dataset corresponds to a location folder L_i. Each folder contains: the raw extracted frames; their associated skyline binary masks; the computed skyline proposals; point-cloud files derived from candidate skylines; and ICP logs capturing frame-to-frame registration behavior. A root-level metadata *.csv* file provides synchronized latitude, longitude, and yaw for every frame, which enables scalability and

maintains a strict correspondence between frame index and skyline-derived features.

Limitations and Considerations: The main limitation lies in the biological variability and temporal instability of tree-dominated skylines. Unlike rigid skylines found in mountainous or urban regions, vegetation silhouettes vary with wind, growth, lighting, and seasonal changes. This introduces noise that affects segmentation and skyline alignment. Moreover, the low-texture nature of the sky, especially in cloudless conditions, limits sometimes the amount of visual information available for monocular localization and increases ambiguity in skyline matching. These factors could also influence the stability of ICP-based and inliers registration when point-clouds are derived from visually inconsistent silhouettes across frames. Finally, fast movements at the moment of the recording could damage a proper position/-heading estimation, as long as both inliers and ICP-metrics are too sensitive to overlapping between frames.



(a) **(b)**

Figure 4.5: Visual characterization of the GNSS groundtruth and the skyline-based pose-graph trajectory. (a): trajectories projected onto OpenStreetMap tiles at different zoom levels. (b): same trajectories in longitude–latitude space, with ICP motion vectors between consecutive nodes drawn on top ready for quantitative analysis

To start with a visual characterization for the results section, the four panels of Fig. 4.5 provide a joint view of the estimated motion and the GNSS reference. In the top-left, groundtruth and pose-graph trajectories are projected onto RVIz OpenStreetMap tiles after the WGS84→WebMercator conversion, while the top-right shows the same paths in longitude–latitude coordinates with ICP motion vectors drawn between consecutive nodes and colour–coded by fitness score. The bottom-left panel zooms in around highlight the spatial density of the pose-graph markers in RVIz, and the bottom-right shows a close-up of the groundtruth orientation arrows in green and ICP vectors colour–coded by fitness score. Here, the orientation of each arrow represents the incremental yaw estimate.
The results are based on more than 140 trials performed along the generated trajectory. Since in this work each node/pose is subject to a constraint given by the registration angle computed by the ICP algorithm, the angular error was measured with respect to the heading angle obtained from the groundtruth GNSS/GPS signal at the point where the ICP algorithm converged, as illustrated in the following figure.

While Fig. 4.5 (b) already provides an overview of the full trajectory in longitude–latitude space, the spatial distribution of the graph nodes and the local consistency between ICP increments and GNSS headings are easier to appreciate at a finer scale. For this reason, Fig. 4.6 presents a magnified view of a representative segment of the path where the trials were performed. In this increased resolution, the plot shows the GNSS groundtruth drawn as a grey reference track, the skyline-based pose-graph nodes

are shown as blue markers, and the ICP motion vectors between consecutive nodes are overlaid and colour–coded by their fitness score. The green arrows depict the GNSS heading orientation at each node, which allows a direct visual comparison between the incremental yaw estimated by the ICP registration and the absolute heading provided by the GNSS receiver.

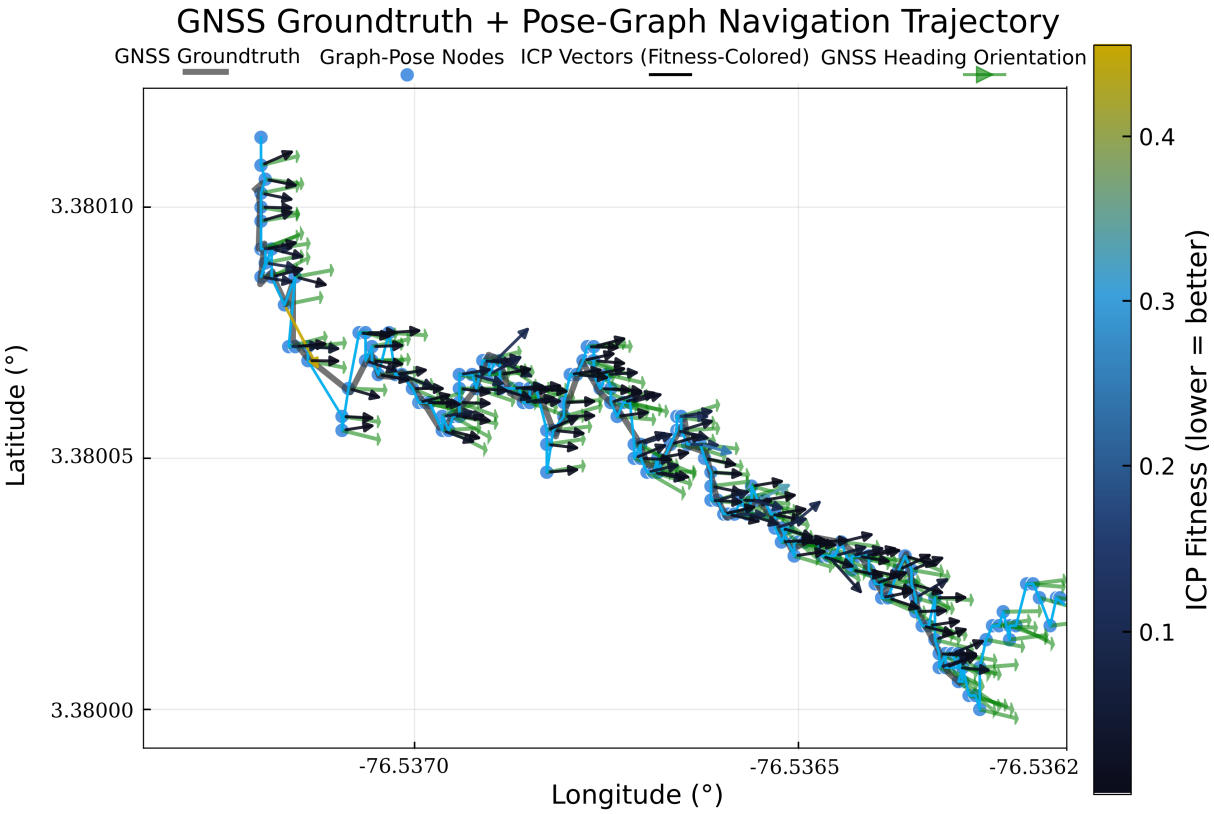


Figure 4.6: Increased-resolution view of the GNSS groundtruth and skyline-based pose-graph trajectory. Grey line shows the GNSS path, blue markers indicate pose-graph nodes, black ICP motion vectors are colour–coded by fitness, and green arrows represent the GNSS heading at each node

In reference to the trajectory shown in Fig. 4.6, the Fig. 4.7 compares the ICP heading estimates with the GNSS reference headings for each trial. The bar pairs in the latter figure illustrate the alignment between both, source and target, which shows the variability of orientation along the trajectory. As can be seen, while the ICP and GNSS headings are generally well-aligned, discrepancies are observed in certain trials, which provides insight into the behavior of the ICP algorithm under conditions without stabilization.

Taking into account the previous context and characterization, and as can be seen in Fig. 4.7, an absolute mean angular error of 3.05° was obtained for all 147 trials. To obtain these values, for each trial, it was first computed the angular error between the ICP registration angle and the corresponding GNSS heading, both expressed in the same local frame. The difference was then wrapped to the interval $[-180°, 180°)$ to account for the circular nature of orientations, and the signed mean bias was calculated as the arithmetic mean of these wrapped errors. Under this definition, the ICP-derived angles exhibit a mean bias of 0.66° counterclockwise with respect to the heading angle recorded from the GNSS signal.

From the results obtained in Fig. 4.7, the trials in which clear outliers appear were selected in order to analyze their causes and implications in more detail. In particular, trials 13, 48, and 111 exhibit significant angular deviations between the ICP and GNSS headings, and are highlighted in Fig. 4.8 (a), (b), and (c), respectively. In all three cases the GNSS heading remains relatively consistent with its local neighbourhood, whereas the ICP estimate shows a different excursion, suggesting a local registration failure, rather than a systematic drift of the trajectory. These outliers therefore behave as isolated faults among otherwise well-aligned sequence of headings.

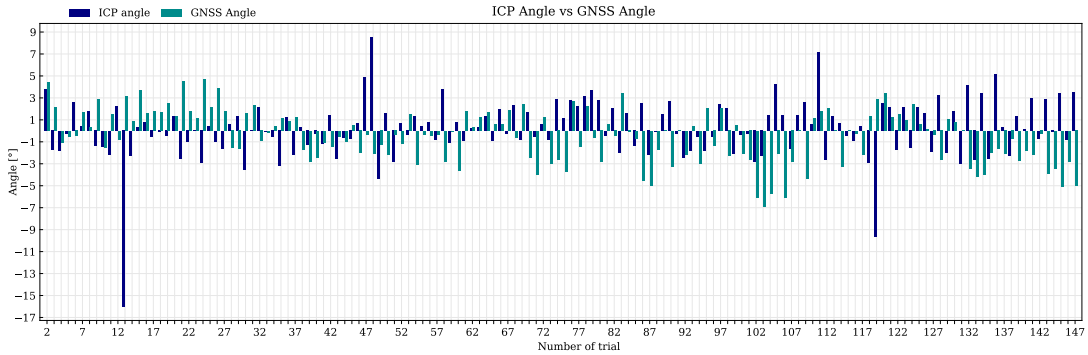


Figure 4.7: ICP heading estimates and corresponding GNSS reference headings for each trial, expressed in the common local frame. The bar pairs show the trial-wise agreement between both sources and the variability of orientation along the trajectory.

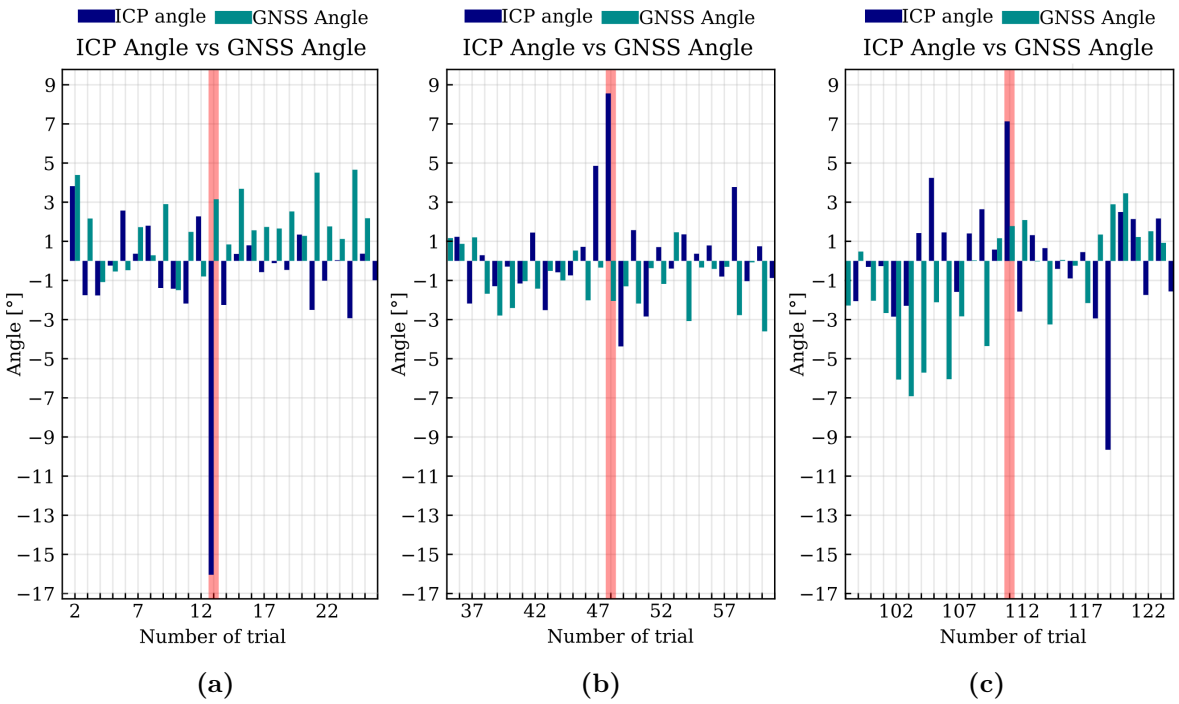


Figure 4.8: Detailed analysis of ICP and GNSS heading discrepancies. The subfigures (a), (b), and (c) highlight specific trials with large angular errors (trial 13 in (a), trial 48 in (b), and trial 111 in (c)) marked by red vertical lines. These outliers are shown alongside the comparison of ICP and GNSS heading estimates, where ICP vectors are visibly misaligned with respect to the GNSS heading

Fig. 4.9 presents a detailed profile of the evaluation process for trial 13, as highlighted in panel (a). The ICP registration process and its associated fitness score, which reaches an unusually high value with rapid speed, are examined in panels (c) and (d), where erroneous convergence is observed, particularly in the yaw angle evolution shown in (c). It is also observed in (d) that the resulting masks exhibit significant differences, such as the omission of a post and an unusual photo-taking angle. The resulting misalignment and the impact of this registration failure on the trajectory are highlighted in panel (b), where the ICP vectors are clearly distorted compared to the GNSS groundtruth. This analysis helps identify the root causes of the discrepancies observed in trial 13.

The Fig. 4.10 presents a detailed profile evaluation centered on trial 48, the frame highlighted (a) in the ICP-vs-GNSS angular comparison plot. This trial has significant differences in heading between the skyline-derived ICP estimate and the GNSS reference, making it a representative case for inspecting the internal behavior of the registration process. Panel (a) highlights the bar corresponding to trial 48 with a

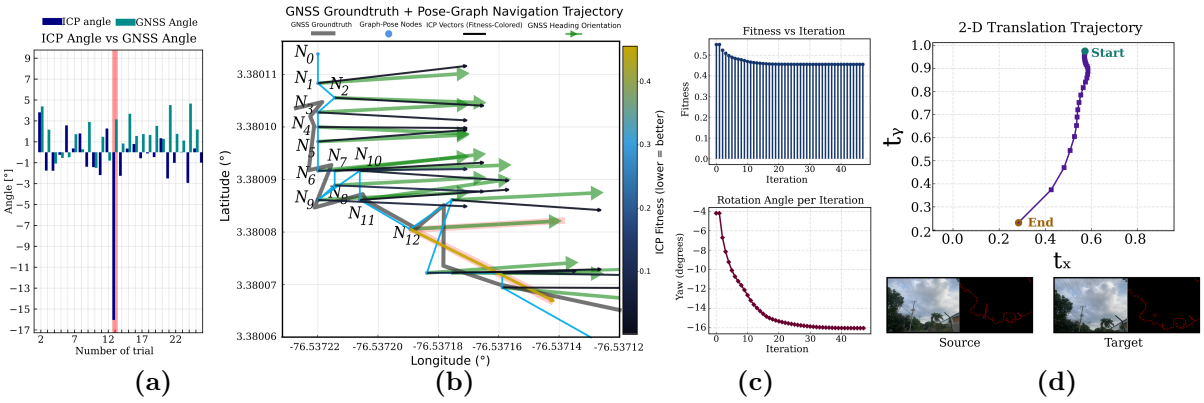


Figure 4.9: Detailed evaluation of trial 13. Panel (a) shows the angular comparison between ICP and GNSS headings for the trial, which highlights discrepancies. Panel (b) presents the GNSS groundtruth trajectory along with pose-graph nodes and ICP vectors, color-coded by fitness score. Panel (c) shows the evolution of fitness speed, while panel (d) visualizes the 2-D translation trajectory, with the source and target images displayed for reference

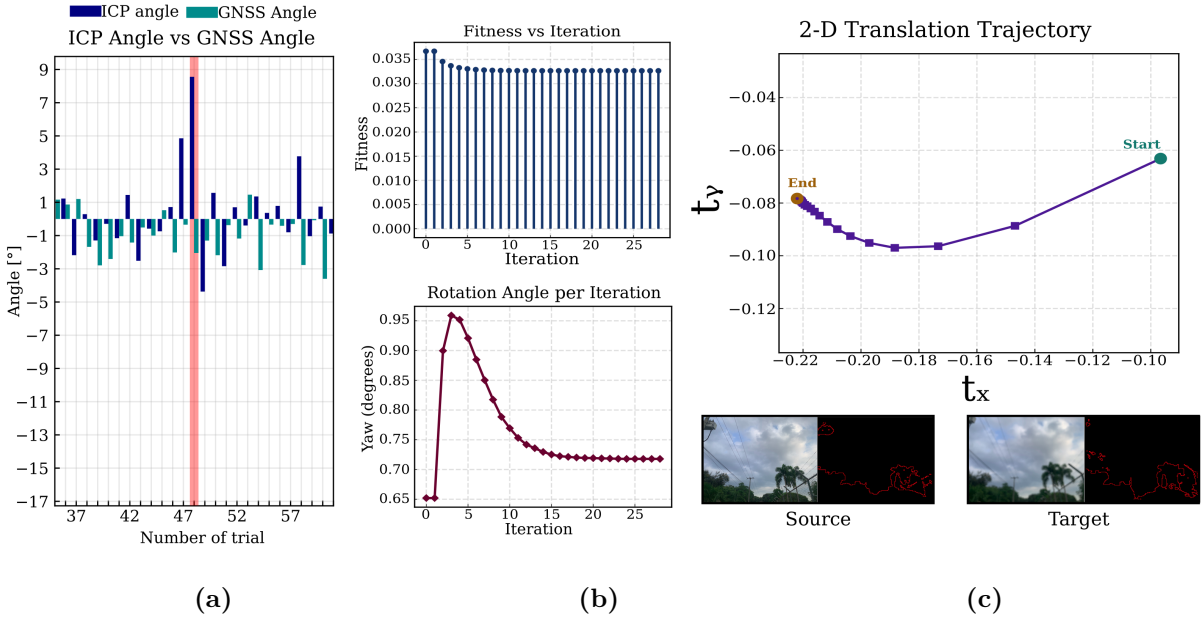


Figure 4.10: Profile evaluation of Trial 48. Panel (a) highlights the trial in the angular comparison plot; panel (b) shows ICP convergence through fitness stabilization and a decaying rotational correction; panel (c) depicts the resulting 2-D translation trajectory and the source–target skyline pair, where appeared ambiguous tree silhouettes

notable angular deviation, which motivates the deeper analysis shown in panels (b) and (c). The Fitness vs. Iteration curve reveals that the ICP alignment converges with fitness values being steady around 0.033, which indicates that the point-cloud correspondence remains consistent but not optimal. Simultaneously, the Rotation Angle per Iteration plot shows an initially large corrective rotation (0.95°) that progressively decays as the algorithm refines the transform. This suggest that the initial misalignment between the two skyline profiles was significant. Panel (c) then visualizes the recovered 2-D translation trajectory, and follows a smooth path but terminates at a displacement direction that differs from the GNSS-implied heading. Together, these subplots explain why Trial 48 accumulates a larger angular error: although ICP converges, the registration landscape for this frame is shallow and influenced by ambiguous silhouette structure. This implies a transform consistent in fitness score space but misaligned in absolute orientation.

The evaluation study for 4.11 extends the profile-based evaluation to trial 111, a case where the ICP and GNSS headings diverge again in a less severe way. In panel (a), Trial 111 is highlighted within the sequence of angular ICP-GNSS comparisons and shows a moderate but consistent deviation that warrants closer inspection. The internal ICP behavior shown in panel (b) indicates a sharper convergence pattern: the Fitness vs. Iteration curve drops rapidly during the first iterations, stabilizing near 0.006, which reflects a more favorable alignment landscape than the previous trial. However, the Rotation Angle per Iteration profile shows a steady corrective rotation behavior that focuses around 1.35°, suggesting that the initial misorientation between the skyline profiles was substantial. This combination of rapid numerical convergence but high rotational correction clarifies why Trial 111 produces a consistent and less severe orientation difference relative to the GNSS heading.

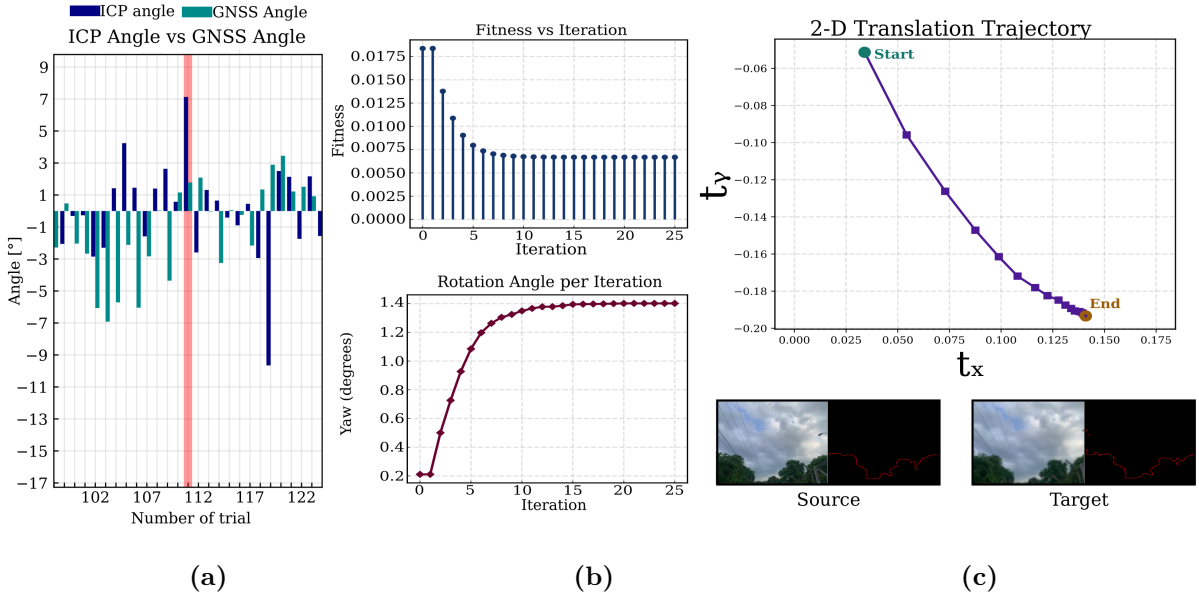


Figure 4.11: Profile evaluation of trial 111, which shows a less severe ICP–GNSS heading mismatch. Panel (a) highlights the trial within the angular comparison plot; panel (b) shows rapid ICP fitness convergence paired with a large corrective rotation; panel (c) depicts the resulting 2-D translation trajectory and source–target skyline images, where smooth canopy shapes limit distinctive alignment cues

When looking across all skyline registrations in the trajectory, a pattern emerged in the number of ICP iterations required for each transform. Most alignments settle quickly, which suggests that the skyline provides enough geometric cues for stable convergence in the majority of cases. Occasional spikes in iteration count appear only when the scene becomes more challenging, typically when there is not an steady focusing over the camera at the moment of the source/target capture. The Fig. 4.12 illustrates one of these harder cases, where subtle changes in silhouette force the optimizer to refine the alignment for significantly longer, which is exactly the trial studied in FIg. 4.11. Overall, the distribution reflects a predictable relationship: well-defined skylines lead to fast convergence, while low-contrast or structurally irregular scenes demand greater computational effort.

Across the entire sequence of ICP transforms, nearly all registrations converge within a narrow band between roughly 15 and 45 iterations. This interval contains the majority of cases, forming a dense and stable cluster throughout the trajectory. Only a few transforms exceed this region (typically rising to 60–80 iterations) and just a single extreme outlier surpasses 200 iterations.
This consistent concentration around the 15–45 iteration range shows that ICP convergence remains computationally light and generally reliable, with higher iteration counts only under structurally ambiguous skyline conditions. To complement this view of algorithmic performance analysis, the Fig. 4.13 introduces the fitness distribution for all transforms across all the trials performed, which provides an additional perspective on alignment quality of the studied trajectory.

Finally, in this overall analysis, the fitness distribution shown in 4.13 provides a complementary view of how reliably ICP aligns consecutive skyline profiles throughout the trajectory. Most transforms register fitness values close to small values as successfully expected, which indicates that the majority of sky-

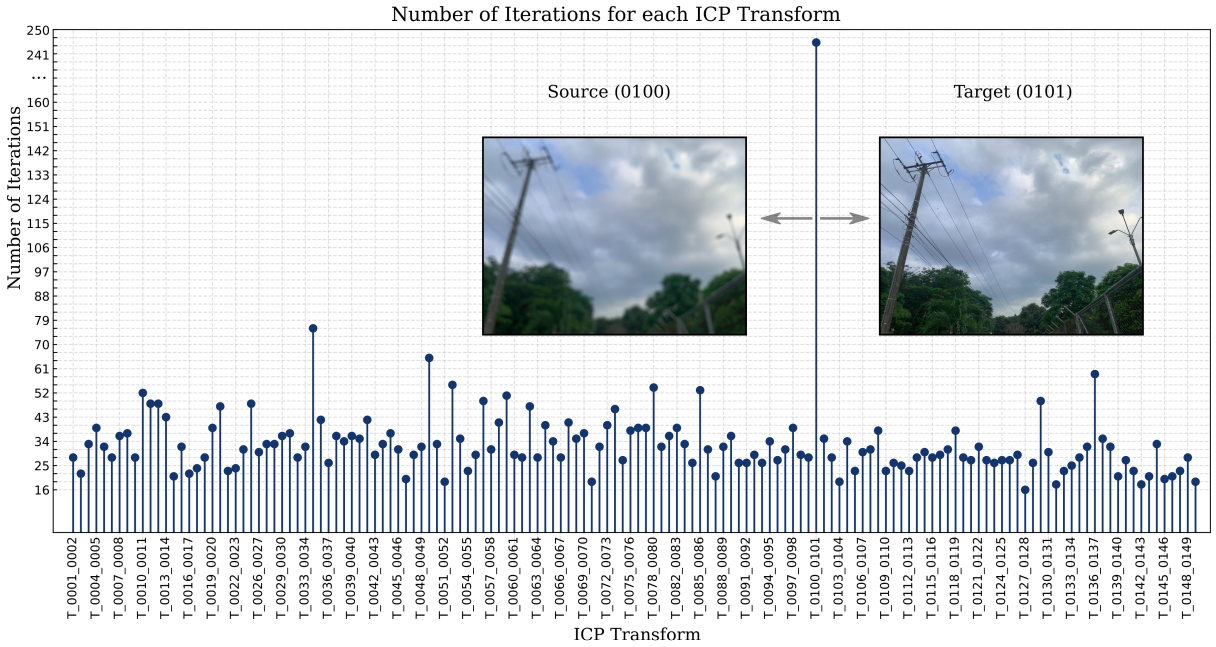


Figure 4.12: Number of iterations required for each ICP transform along the trajectory. Most registrations converge within a compact range of approximately 15–45 iterations, while only a few cases demand substantially more refinement. A single outlier exceeds 200 iterations, the one that belongs to a skyline pair with ambiguous structure, as illustrated by the embedded source and target images

line pairs achieve tight correspondence once convergence is reached. This pattern supports the earlier conclusion that ICP consistently solves well-conditioned alignment problems whenever the skyline offers clear geometric cues. However, several pronounced spikes appear across the sequence, most notably exceeding 0.45, 0.25, and 0.32. These peaks reflect alignment scenarios dominated by low-contrast tree canopies, smooth cloud contours, or asymmetrical silhouettes that reduce the reliability of point-to-point matching. When viewed together with the iteration trends, the fitness profile reveals a stable baseline performance regarding to localized difficulty zones. This confirms that ICP's accuracy depends directly on the structural clarity of the skyline visible in each frame.

In conclusion, the different trials confirm that the few misaligned cases came from local registration difficulties rather than a systematic drift in the method. While the vast majority of transforms converge within 15–45 iterations and maintain fitness scores close to zero-vicinity, the outliers show markedly different behavior: iteration counts rising to 60–80 and, in one extreme case, exceeding 200, together with fitness peaks reaching 0.25–0.45. These values reflect ambiguous, unfocused moments or low-contrast skylines that momentarily destabilize the alignment but significantly accumulate the drift regarding the overall trajectory.

4.2.2 Scale Factor Empirical Validation

To verify the camera-geometry-derived scale factor $\lambda = 0.204$ m/px, a validation was performed using the GNSS ground-truth trajectory. The empirical scale factor was computed as:

$$\lambda^{j}_{\text{empirical}} = \frac{\|\mathbf{p}^{j+1}_{\text{GNSS}} - \mathbf{p}^{j}_{\text{GNSS}}\|_2}{\left\|\sum_{i\in\mathcal{I}_j} \mathbf{t}^{i}_{\text{ICP}}\right\|_2} \tag{20}$$

where $\mathbf{p}_{\text{GNSS}}$ denotes the position in the local ENU frame, $\mathcal{I}_j$ denotes the set of ICP frame pairs falling within the j-th GNSS interval, and the summation represents the vector aggregation of pixel displacements.
The empirical distribution resulted in:

$$\lambda_{\text{empirical}} = 0.286 \pm 0.126 \text{ m/px} \quad (\text{mean} \pm \text{std}) \tag{21}$$

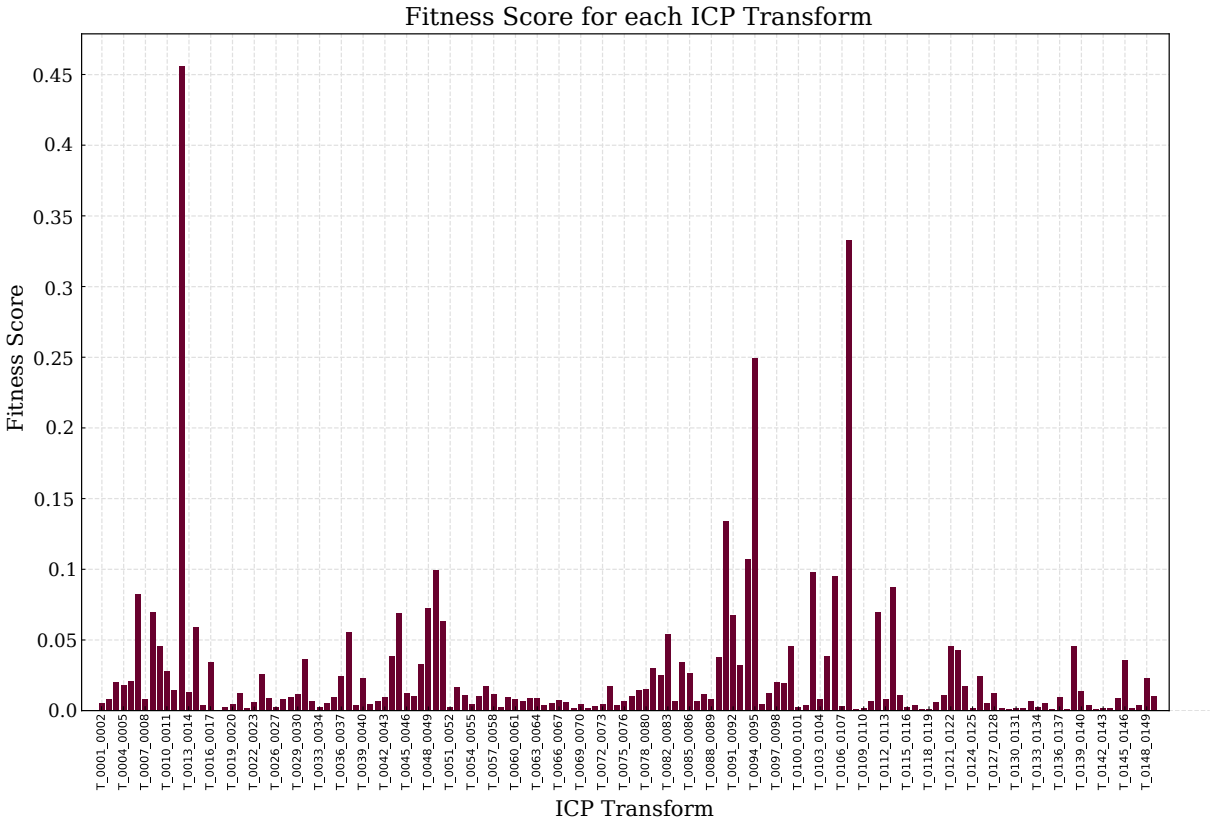


Figure 4.13: Fitness score for each ICP transform along the trajectory. Most skyline registrations achieve low fitness values, and this indicates good alignment quality, while a small number of pronounced peaks mark frames with ambiguous or low-contrast silhouettes. Some of these outliers align with the high-iteration cases observed previously, and this show how skyline structure affects the difficulty of ICP alignment

Figure 4.14 presents the temporal evolution of the local scale factor estimates throughout the dataset. The plot shows frame-by-frame scale factor estimates computed from consecutive ICP transformations, with the theoretical value (0.204 m/px, red dashed line) and empirical mean (0.286 m/px, orange) indicated as horizontal references.

The scale factor evolution exhibits significant temporal variability, particularly in the first half of the trajectory (frames 0-1200), where estimates fluctuate between 0.1 and 1.0 m/px. This high variance is attributed to: rapid yaw maneuvers during initial field operations, short-baseline displacement accumulation relative to GNSS measurement uncertainty (±2-3 m), and variable skyline feature geometry as the camera viewpoint changes and the blur effect constantly appeared.

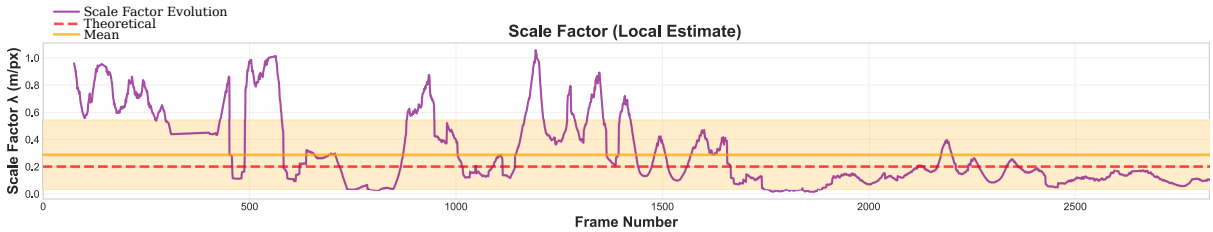


Figure 4.14: Temporal evolution of local scale factor estimates across the entire dataset

The relative difference between the theoretical value (0.204 m/px) and the empirical mean (0.286 m/px) is approximately 40%. This difference suggests that the far-field assumption ($d_{\text{avg}} = 100$ m) underestimates

the true distance to the skyline features. From the empirical scale factor, the implied effective depth is:

$$d_{\text{avg}}^{\text{implied}} = \lambda_{\text{empirical}} \cdot f'_x = 0.286 \times 489.07 \approx 140 \text{ m} \tag{22}$$

This means that the depth estimate tends to align with the maximum distance estimated in the field environment. The persistent offset between theoretical and empirical values suggests a systematic bias in the far-field and depth assumption.

4.3 Field Testing



Figure 4.15: Panoramic images captured during the field test at Universidad del Valle's C17 football field for Field Testing trial FT0

Field Testing 0 (FT0): The first field trial was conducted in the Centro Deportivo Universitario from the Universidad del Valle C17 football field, in Cali, Colombia. This dataset, which comprises over 2,800 poses, was captured under clear sunny conditions, with strong natural illumination, which provided high contrast between the skyline silhouettes and the sky. This environment was selected as it offers a non-constrained trajectory with a distant, complex horizon ideal for texture evaluation on cloudless zones. The overlap between the GNSS reference and the PoseGraph ICP–EKF estimate in Fig. 4.16 hints the accuracy and robustness of this approach for potential future real deployments, under proper conditions.

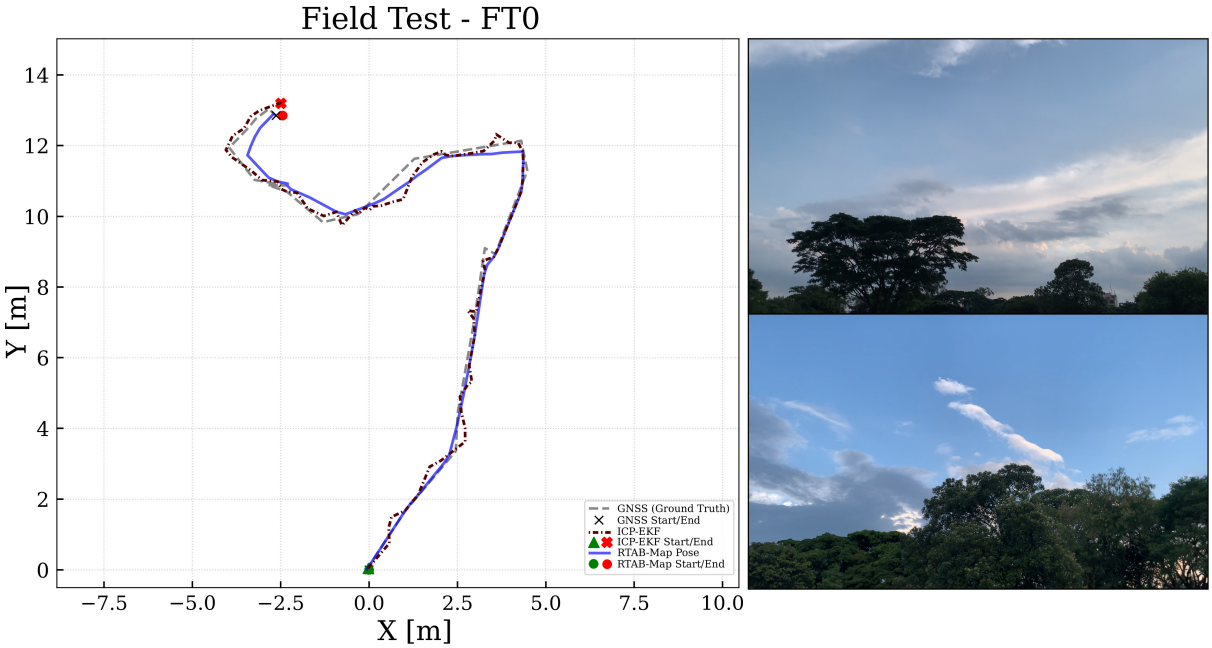
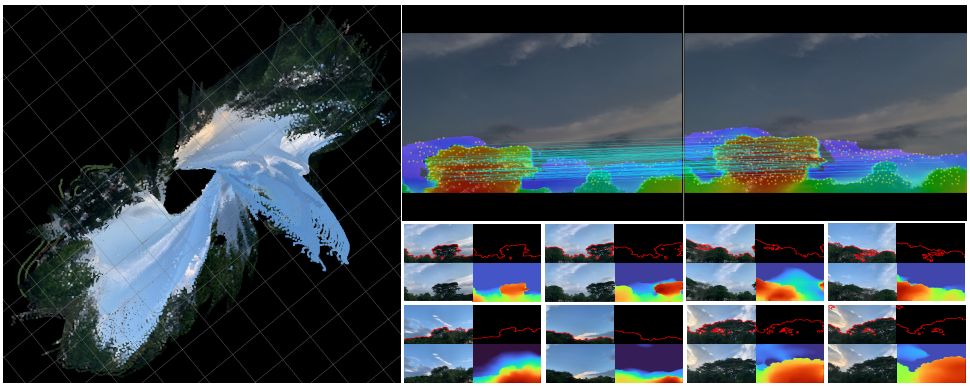


Figure 4.16: Field test at Universidad del Valle's football field. In the left panel, a comparison between GPS ground-truth trajectory (solid gray) and skyline-based ICP odometry aligned in the same planar frame for evaluation (dashed brown), with start/end markers, as well as its corresponding optimized posegraph from RTAB-Map engine. (b) Sample sky and tree-line images illustrating the visual conditions that define the skyline

In the comparison between both trajectories, the estimate trajectory stays close to the GNSS reference trajectory, and finishes the trajectory with small endpoint drift, which suggests that the fused pose remains metrically consistent throughout the experiment. These results raise a natural question: how does this trajectory embed itself inside the global map that RTAB-Map reconstructs from the same image sequence?

Figure 4.17 addresses the previous question. In such figure, (a) panel presents the 3D RTAB-Map occupancy grid for Fig. 4.16 together with the optimized camera poses, which trace a trajectory across the football field. In the right panel, the corresponding RGB-colored point cloud is shown, where the tree canopy, open grass corridor, and surrounding structures emerge with geometric correspondence according to the non-steady movements regarding the pitch and roll dimensions, However, those movements were ignored and just the yaw-related implications coming from this particle occupancy-grid was took into account. The agreement between the trajectory inferred from skyline alignment and the dense 3D reconstruction indicates that the proposed localization method not only tracks the GNSS trajectory but also remains consistent with the underlying visual map.



(a) **(b)**

Figure 4.17: The figures illustrate the results of the RTAB-Map system. Figure (a) displays a 3D aerial view of the generated occupancy grid. Figure (b) demonstrates the skyline matching process, showing how RGB images and MiDaS depth projections are processed to extract stable skyline features (red traces). These features are used to align consecutive observations via ICP, which demonstrates the method's ability to maintain tracking across different viewpoints

Additionally, Fig. 4.17 illustrates how the proposed skyline pipeline feeds the RTAB-Map style map frame in the field test. Each group show the original RGB view, the MiDaS depth estimate with a blue–to–warm scale, and the final skyline mask with the ICP correspondence set overlaid. Warm colors indicate closer structures, whereas the red traces mark the skyline samples that enter the ICP objective. Together with the occupancy grid and 3D point cloud built in the style of RTAB-Map's RGB-D graph SLAM framework [45], these examples show that the system extracts a stable skyline signal under different viewpoints and that ICP successfully aligns consecutive observations with the global map frame.

Figure 4.18 illustrates how the skyline-based ICP odometry behaves under different uniform scale factors when compared against the GPS ground-truth trajectory. Because the ICP odometry provides only relative motion increments, it does not recover the absolute metric scale of the environment. However, its geometric structure remains consistent across the sequence. (a) shows the unscaled ICP trajectory (1.0×), which preserves the overall shape of the motion but does not match the GPS extent. When the same relative trajectory is scaled by 2.5× (center) and 5.0× (right), the path expands toward the spatial envelope defined by the GPS ground-truth, which reveals that the ICP geometry captures the underlying spatial pattern even though absolute scale is not observable from skyline silhouettes alone. Across all scale factors, the trajectory topology remains similar, and this confirms that the skyline-based odometry is geometrically coherent and that scale ambiguity is the principal unresolved component.

Beyond the scale evaluation, Fig. 4.19 provides a complementary view of the global consistency of the proposed localization framework. Panel (a) contrasts the blind ICP-based odometry—accumulated with-

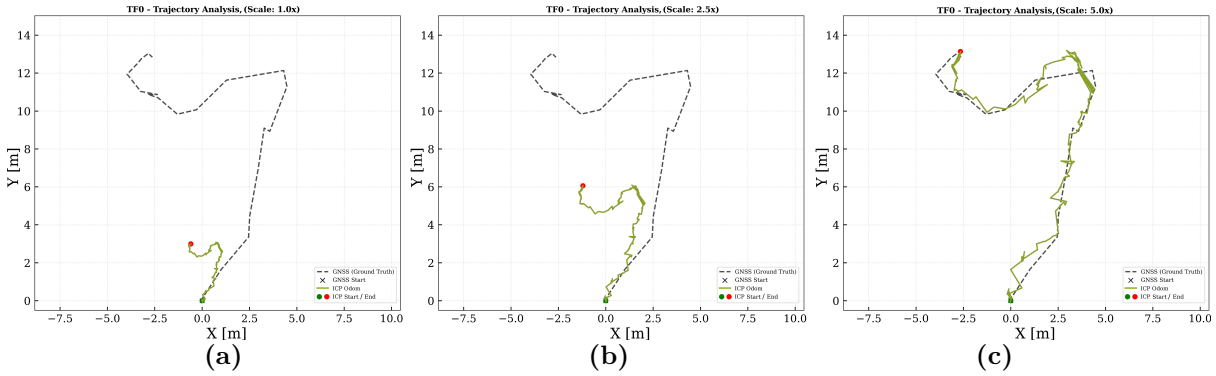


Figure 4.18: Comparison of the optimized FT0 pose-graph under increasing global scale factors. The trajectory transitions from an underscaled reconstruction (a) to a geometry that progressively aligns with the GNSS ground truth (b) and (c). This shows the scale-ambiguous nature of skyline-based odometry

out any global correction, with the trajectory obtained from the offline RTAB-Map baseline. Although RTAB-Map is not part of the pipeline, its globally optimized path offers an external reference that highlights drift trends in the raw skyline-based odometry. Panel (b) places the RTAB-Map trajectory in the georeferenced map region where the FT0 experiment was conducted, showing how the optimized path corresponds to the actual spatial layout.

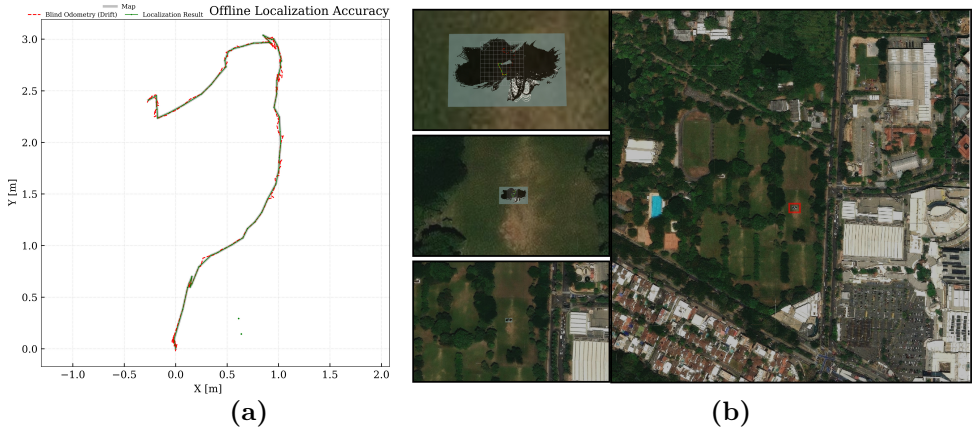


Figure 4.19: In (a): Comparison between the blind odometry drift and the optimized localization result that shows the improved alignment with the RTAB-Map reference map. In (b): the trajectory in the georeference place were the FT0 was taken

To quantitatively and qualitatively validate the performance of the proposed localization framework, an offline analysis was conducted over the full recorded dataset. This evaluation considers two complementary indicators: the internal confidence of the visual front-end and the global geometric consistency of the estimated trajectory. The internal confidence is measured as the number of RANSAC-verified visual inliers at each keyframe, which reflects the stability of skyline alignment and the reliability of the motion estimate. As illustrated in Fig. 4.20, the inlier count typically oscillates between 150 and 350 matches, with peaks exceeding 500 in well-textured or structurally rich regions or steady movements. Transient drops below 50 and are caused by changes on the blur stability, partial occlusions, or skyline degradation.

In parallel, the global consistency of the localization output is assessed by comparing the reconstructed trajectory against both the GNSS ground truth and the uncorrected (blind) odometry, as shown in Fig. 4.19. The correlation between the inlier profile with and the trajectory behavior shows a clear pattern: low-inlier intervals often coincide with small local distortions in the raw ICP odometry, while periods with sustained inlier counts above 250 reliably produce smooth, drift-free segments. The evidence from these results demonstrates that inlier density is not only a frame-by-frame confidence indicator, but also a meaningful predictor of downstream trajectory accuracy and the effectiveness of global pose-graph

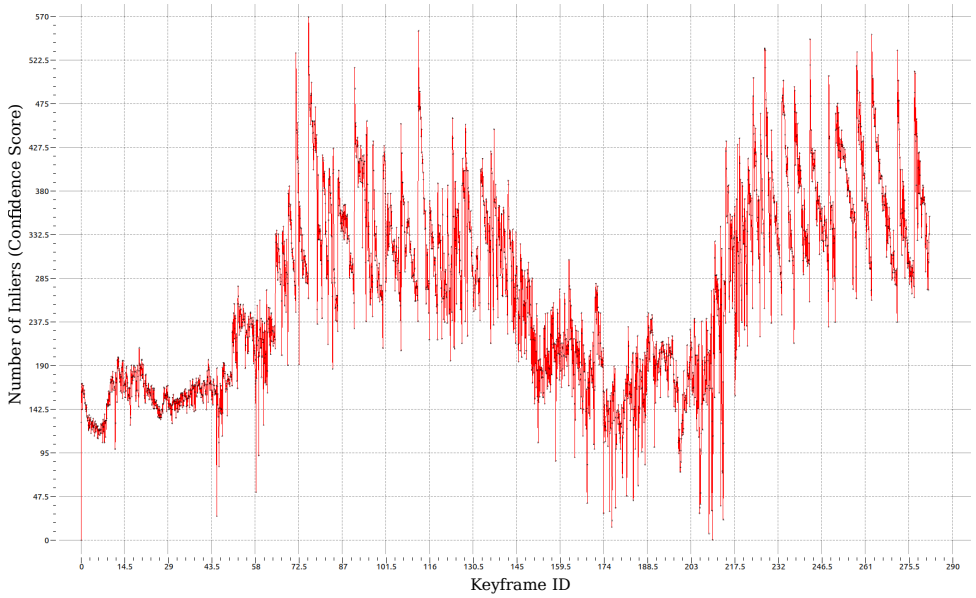


Figure 4.20: RANSAC-verified inlier count across the full sequence. It shows typical confidence levels between 150 and 350 inliers, peaks above 500 during highly stable skyline conditions, and a short trend below 100 inliers

optimization.



Figure 4.21: Offline localization visualization for Field Test FT0 at Universidad del Valle's C17 football field. (a) RViz display showing the reconstructed pose-graph trajectory overlaid on the occupancy grid, where colored markers indicate the ICP-derived poses and the connecting edges represent the relative transformations between consecutive skyline frames. The trajectory follows the path captured during the field test at Universidad del Valle's C17 football field. (b) Contextual view of the skyline-based alignment projected onto the satellite imagery, which illustrates the spatial correspondence between the estimated trajectory and the physical environment. (c) Aerial perspective of the FT0 testing location. It shows the open sports field area characterized by distant tree lines and clear horizon structures that provide the skyline features used for localization

In the statistical analysis, as shown in Fig. 4.22, all skyline-based ICP transforms confirm that the method produces a highly stable alignment signal throughout a trajectory composed of more than 2,800 frames. The heading error distribution in Fig. 4.22 (b) shows that regardless of distance traveled, the yaw-angle error recovers consistently after accumulated spikes and the absolute orientation error remains predominantly below 15°, with an overall RMSE of 12.41°. Regarding the orientation distribution error: While this value reflects occasional excursions during challenging skyline conditions, the error recovers when well-defined horizon features become available. Overall, the approach of this work shows that both the positional drift and the heading deviation remain bounded throughout the field test, which confirms that vision-based estimates maintain a healthy accuracy along the entire trajectory.

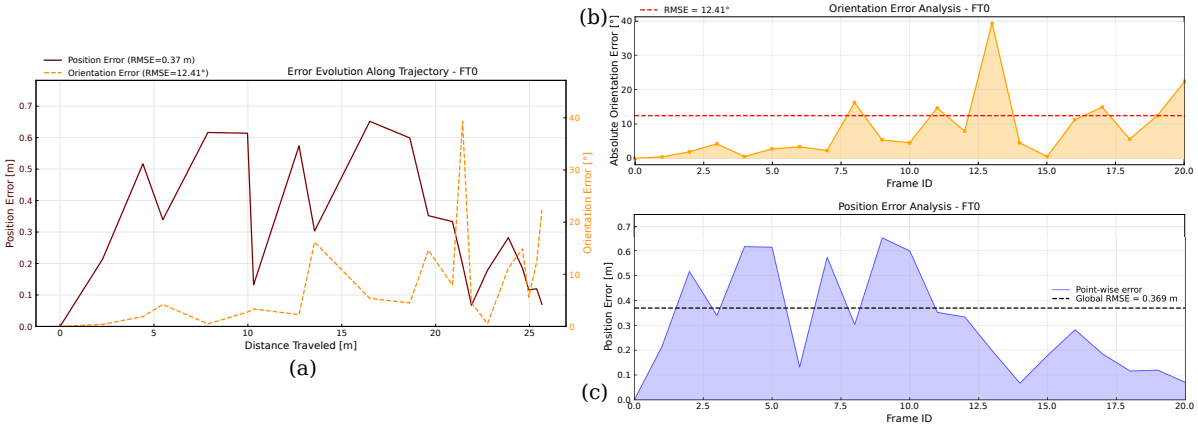


Figure 4.22: Error analysis for Field Test FT0. (a) Error evolution along the trajectory showing position error (RMSE = 0.37 m) and orientation error (RMSE = 12.41°) as functions of distance traveled. The position error remains bounded below 0.8 m throughout the 25 m trajectory, while orientation error exhibits periodic recovery after transient spikes. (b) Orientation error analysis shows the absolute orientation error across all frames, with a global RMSE of 12.41° indicated by the dashed line. (c) Position error analysis displaying the point-wise error distribution with a global RMSE of 0.369 m. The shaded area highlights the error envelope, which demonstrates that the majority of frames achieve sub-0.7 m accuracy

As illustrated in Fig. 4.22, panel (a) provides a numerical overview of the position error characteristics observed during the field test. The point-wise position error oscillates with most values clustering in the 0.1-0.6 m range, which is consistent with the error evolution shown in panel (c). The position RMSE of 0.37 m confirms that meter accuracy is bounded throughout the FT0 sequence. Panel (a) further demonstrates the correlation between position and orientation errors along the trajectory, where the position error (RMSE = 0.37 m) remains stable while orientation errors (RMSE = 12.41°) exhibit occasional spikes that recover as the skyline quality improves.

Figure 4.23 provides a complementary view of the error accumulation behavior. Panel (a) shows the sliding window position RMSE across the frame sequence, where the error remains consistently bounded below 0.7 m throughout the entire trajectory. Panel (b) illustrates the cumulative position RMSE as a function of distance traveled, which demonstrates a final RMSE of 0.369 m after approximately 25 m of traversal. The monotonically increasing trend with a gradual slope confirms that drift accumulation is controlled by the ICP-EKF fusion, and the system maintains metric consistency over the full extent of the field test.

This field-test evaluation shows how the fused ICP-EKF trajectory remains metrically consistent with respect to the GNSS reference under real-world conditions. As discussed, the absolute position error remains bounded, with a final RMSE of 0.369 m and typical frame-wise errors between 0.1-0.6 m. The orientation error, while exhibiting an RMSE of 12.41°, demonstrates the system's ability to recover heading estimates when well-structured skyline features are visible. The RTAB-Map constructed trajectory via Visual-SLAM shows that the fused pose integrates coherently within a dense vision-based map, which preserves the structure of the surrounding terrain, regardless of the scale mismatch. The agreement between the GNSS reference, the ICP-EKF solution, the Visual-SLAM generated trajectory, and the 3D reconstruction demonstrates that the proposed method tracks the motion, under the assumptions discussed in Section 3.2.2. This integration validates that the skyline signal extracted in this work support long-term mapping and localization tasks in large outdoor spaces.

Figure 4.24 illustrates the overall localization workflow: (a) the initial setup showing the skyline extraction and point cloud alignment used to initialize the trajectory and intermediate stages where consecutive skyline frames are matched and integrated into the pose-graph; and (b) the final refined localization result, where the optimized trajectory aligns consistently with the reconstructed map. This sequence visually captures how the system evolves from raw visual input to a globally localization solution.

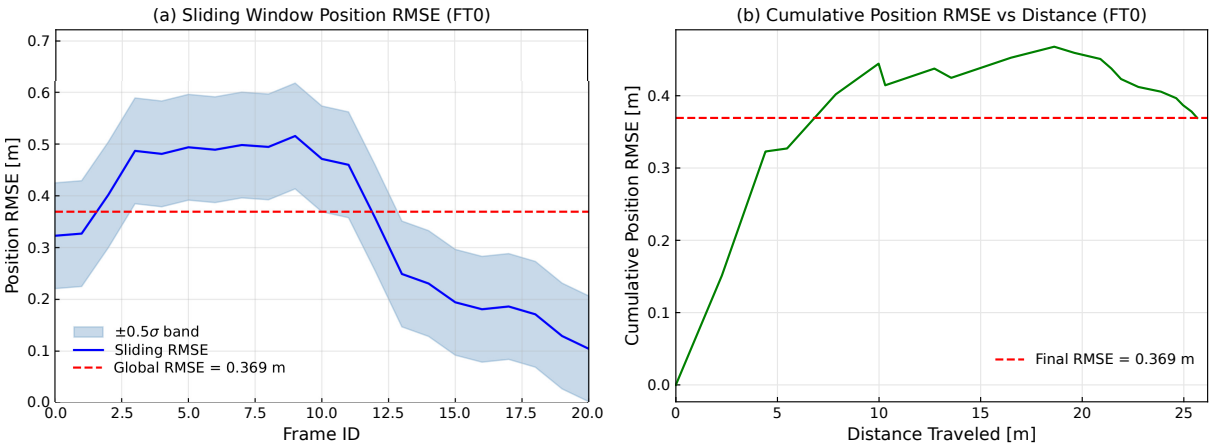
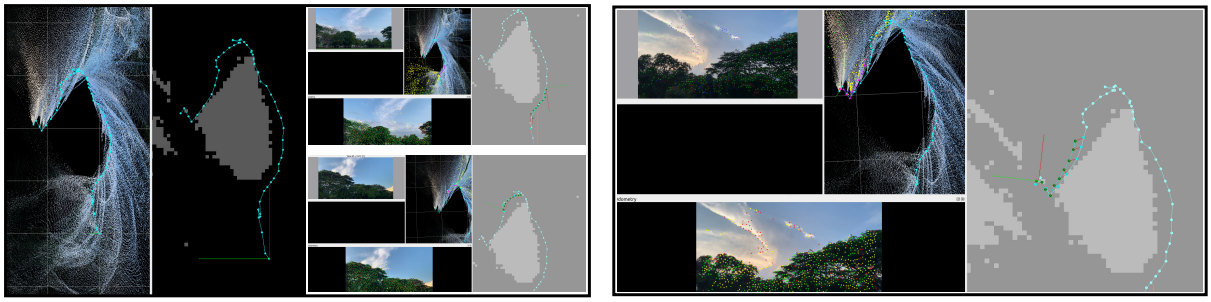


Figure 4.23: Sliding window and cumulative RMSE analysis for Field Test FT0. (a) Sliding window position RMSE across the frame sequence, where the blue line represents the instantaneous RMSE and the shaded region indicates the $\pm 0.5\sigma$ uncertainty band. The global RMSE of 0.369 m is shown as a dashed reference line. The sliding RMSE fluctuates between 0.3 m and 0.6 m, with peak values occurring around frames 8-12 where skyline visibility was temporarily reduced. (b) Cumulative position RMSE as a function of distance traveled, exhibits growth from near-zero to a final RMSE of 0.369 m over approximately 25 m. The gradual slope confirms controlled drift accumulation throughout the trajectory



(a) **(b)**

Figure 4.24: Visualization of the skyline-based localization process: initial setup, intermediate pose-graph updates (a), and final optimized trajectory (b)

The quantitative evaluation confirms that the skyline-based localization pipeline achieves the expected levels of accuracy and robustness. ICP statistics show that both rotational and positional errors remain well bounded, and this implies that the skyline geometry provides a reliable orientation cue even under challenging outdoor conditions. Convergence behavior is equally steady, with 75% of registrations within 12 iterations and fewer than 10% reaching the 50-iteration limit, typically in frames with degraded skylines. The fitness analysis further supports this consistency: over 90% of alignments fall under 0.01. Together, these metrics verify that the skyline-based ICP front-end supplies low-noise, dependable constraints suitable for downstream fusion.

Field Testing 1 (FT1):
The second field trial (FT1), conducted at a different location and time of day (also within the Universidad del Valle sports complex), introduced greater variability in skyline structure and illumination. The sequence contains over 7,700 poses, which corresponds to a trajectory of approximately 45.28 m, and provides a more challenging benchmark than FT0. Despite these conditions, the numerical results confirm bounded performance: as shown in Fig. 4.28, position error relative to the GNSS reference generally remained within the 0.5-2.0 m range, with the position RMSE reaching 1.26 m. The orientation error exhibited greater variability than in FT0, with an RMSE of 30.47°, this reflects the more challenging visual conditions including stronger illumination changes and partial skyline occlusions, along with greater accumulated drift.



Figure 4.25: Panoramic images captured at the field test trial FT1 in the area between football fields C14 and C15 at the Universidad del Valle

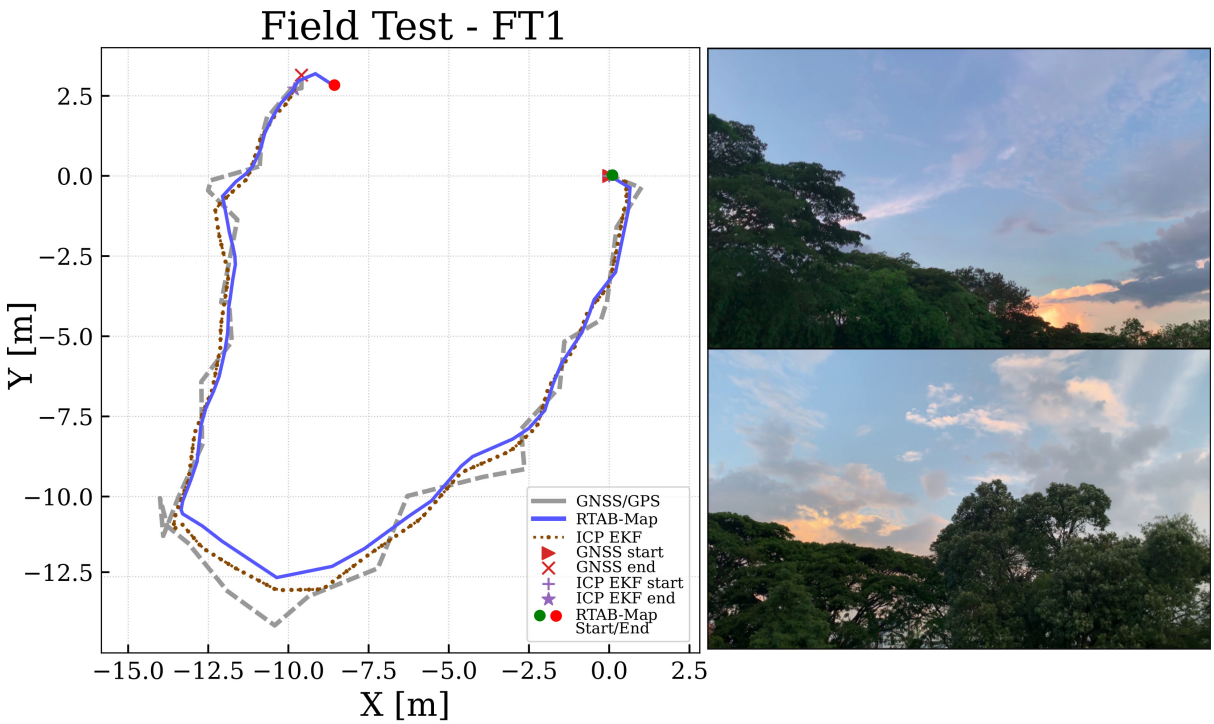
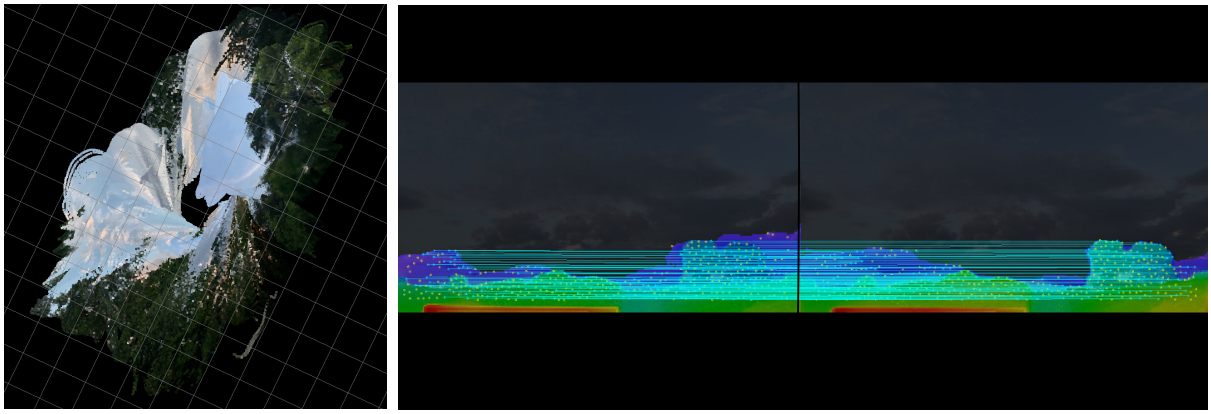


Figure 4.26: Comparison of the FT1 trajectory estimated by the ICP–EKF fusion against GNSS ground truth, alongside representative skyline views captured during the run

Given the previous results on the trajectory, it is equally important to inspect how the system behaves internally during the mapping and alignment process. Figure 4.27 provides a detailed view of these components. In (a), it is shown how the three-dimensional occupancy grid is generated after processing the full FT1 sequence, where the reconstructed structure captures the surrounding vegetation and sky contours with sufficient density to support global pose-graph optimization. This volumetric map illustrates how skyline-derived odometry contributes to a spatially coherent reconstruction despite operating with monocular input. Panel (b) complements this due to the correspondence inliers matching between two consecutive skyline frames. Here, depth-enhanced silhouettes, pixel-level matches, and geometric constraints can be seen interacting to form the incremental ICP transformation that drives the trajectory forward along with the matching descriptors.

When adding moderate scaling factors to the resulting ICP trajectory (2.5× to 5.5×), just as in the progression of the Fig. 4.30, the alignment of the optimized trajectory with the GNSS footprint is possible, and this reveals the matching geometry, which increase the accuracy and shows that most of the discrepancy lies in a global multiplicative factor rather than in local distortions or drift. This observation confirms that the ICP-based skyline constraints are reliable in capturing the shape and topology of the path, even under field conditions with complex vegetation and partial skyline occlusions. However, it also underscores that absolute scale must be supplied through external cues, such as GNSS or additional



(a) **(b)**

Figure 4.27: (a) Three-dimensional occupancy grid generated from the full FT1 sequence, which shows how the reconstructed structure captures vegetation and skyline contours with sufficient density to support global pose-graph optimization. (b) Correspondence analysis between consecutive frames, where depth-enhanced silhouettes and pixel-level matches illustrate the inlier set that constrains the incremental ICP transformation during trajectory estimation

monocular depth priors, since the skyline-derived ICP increments encode purely relative transformations. Together, such findings demonstrate both the strengths and the limitations of the proposed visual pipeline: the trajectory geometry is repeatable, yet metrically ambiguous without fusion/optimization.

Figure 4.28 presents the detailed error analysis for the FT1 sequence. Panel (a) shows the frame-by-frame position error, which oscillates predominantly between 0.5 m and 2.0 m, with occasional excursions toward 2.5 m during segments with degraded skyline visibility. This panel also illustrates the error evolution along the trajectory, where the position error (RMSE = 1.26 m) and orientation error (RMSE = 30.47°) are plotted against distance traveled. The correlation between position and orientation errors reveals that larger heading deviations often coincide with periods of decreased positional uncertainty. Panel (b) confirms the orientation error distribution, which shows that while most frames exhibit errors below 40°, the extended sequence includes challenging segments that push the orientation error toward higher values.

As illustrated in Fig. 4.29, the sliding window analysis reveals the temporal evolution of localization accuracy throughout the FT1 sequence. Panel (a) displays the sliding window position RMSE with a ± 0.5 m uncertainty band (σ), and this shows how the error fluctuates between 0.5 m and 2.5 m upon on local skyline quality. The global RMSE of 1.259 m represents the aggregate performance across all frames. Panel (b) shows the cumulative position RMSE as a function of distance traveled, which increases gradually from near-zero at the start to a final RMSE of 1.259 m after approximately 40 m of traversal. The relatively linear growth in cumulative error indicates consistent drift accumulation rather than sudden failures. This confirms that the localization pipeline degrades under challenging conditions rather than a sudden catastrophic drift.

After the scale analysis in Fig 4.30, it is essential to assess how well the proposed localization framework performs when both, the raw drift-prone odometry and the refined trajectory are compared directly against the reconstructed map.

The relationship between localization accuracy and feature quality becomes evident when comparing the error profiles in Figs. 4.28 and 4.29 with the RANSAC inlier statistics shown in Fig. 4.31. When contrast drops or the horizon becomes partially occluded, the ICP front-end produces weaker motion estimates that accumulate into the deviations observed in the RMSE plots. In contrast, stable skyline segments produce constraints that the pose-graph optimizer can use to maintain global consistency. This relationship becomes clear in such Fig. 4.31, where fluctuations in the RANSAC inlier count (typically between 220-441 inliers) align with the drift regions visible in Fig. 4.28, confirming that the localization accuracy of the ICP vision-based approach is directly governed by the quality of the skyline-derived correspondences.

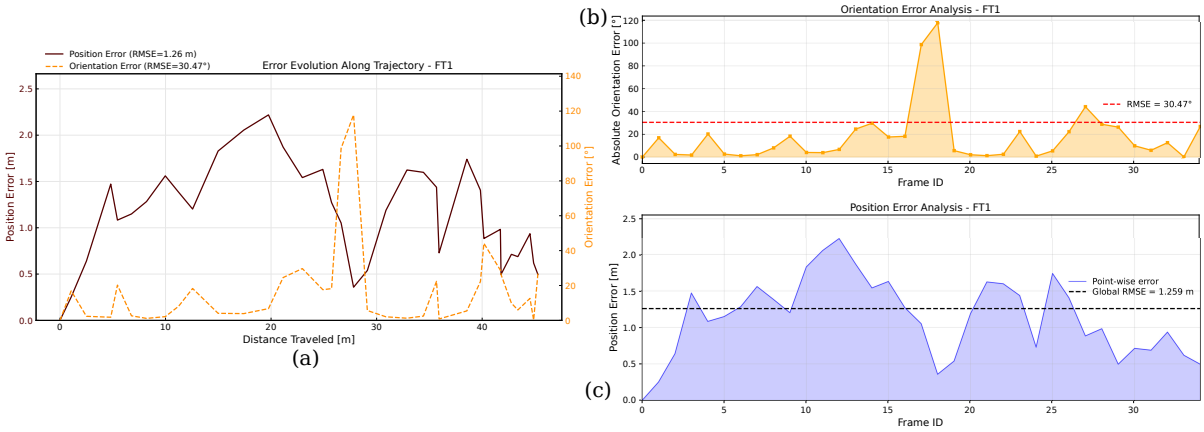


Figure 4.28: Error analysis for Field Test FT1. (a) Error evolution along the trajectory shows position error (RMSE = 1.26 m) and orientation error (RMSE = 30.47°) plotted against distance traveled. The position error exhibits greater variability than FT0, ranging from 0.1 m to 2.5 m, with notable excursions in the 15-30 m segment where illumination conditions were most challenging. (b) Orientation error analysis displays the absolute orientation error across all frames. The RMSE of 30.47° reflects the increased difficulty of keep consistent heading estimates under heterogeneous lighting, with prominent spikes around frames 15-20 that corresponds to partial skyline occlusions. (c) Position error analysis shows the point-wise error distribution with a global RMSE of 1.259 m. The error profile reveals a characteristic pattern where accuracy degrades in the middle portion of the trajectory before partially recovering toward the end

While Fig. 4.31 shows that the skyline front-end remains mostly stable, with 220-440 inliers across the sequence and only brief drops during visually difficult moments. These trends confirm that the skyline generally provides enough structure for reliable ICP alignment despite occasional degradation, Fig. 4.33 illustrates how these fluctuating inlier levels are handled by the localization pipeline: raw skyline matches feed into ICP odometry, and the pose-graph backend compensates for local inconsistencies to produce a globally coherent trajectory. Together, the figures show that the system remains robust even when short-term skyline quality varies.

4.4 Conclusions

Across both field trials (FT0 and FT1), the skyline-based localization pipeline demonstrated bounded accuracy and repeatable behavior under real outdoor conditions. Table 4.1 summarizes the key performance metrics from both experiments.

In FT0, the fused ICP-EKF trajectory maintained a position RMSE of 0.369 m, with frame-wise errors consistently within the 0.1-0.6 m range and never exceeding 0.7 m. The orientation RMSE of 12.41° reflects periodic heading variability, though the system demonstrates reliable recovery when well-defined skyline features are visible. These metrics confirm that under a clean and well-structured skyline, the ICP front-end provides highly reliable geometric constraints, and the EKF fusion preserves metric consistency along the entire trajectory.

FT1, a more challenging test with over 7,700 poses, stronger illumination changes, and greater skyline variability, revealed the limits of the approach while still demonstrates bounded performance errors. The position RMSE of 1.259 m represents approximately 3.4× degradation compared to FT0, this reflects the increased difficulty of the extended sequence. Position errors were typically in the 0.5-2.0 m range, with brief excursions toward 2.5 m through segments with degraded skylines. The orientation RMSE of 30.47° indicates greater heading variability under difficult illumination, though the system continued to provide usable orientation estimates throughout the sequence. Table 4.1 summarizes this.

Although the alignment matches visual features (skylines), the displacement obtained is not used as raw pixels. The system operates on a Local Tangent Plane (LTP) where the visual odometry is projected into metric space via linear scaling. As demonstrated in the results, the local coordinates are first normalized

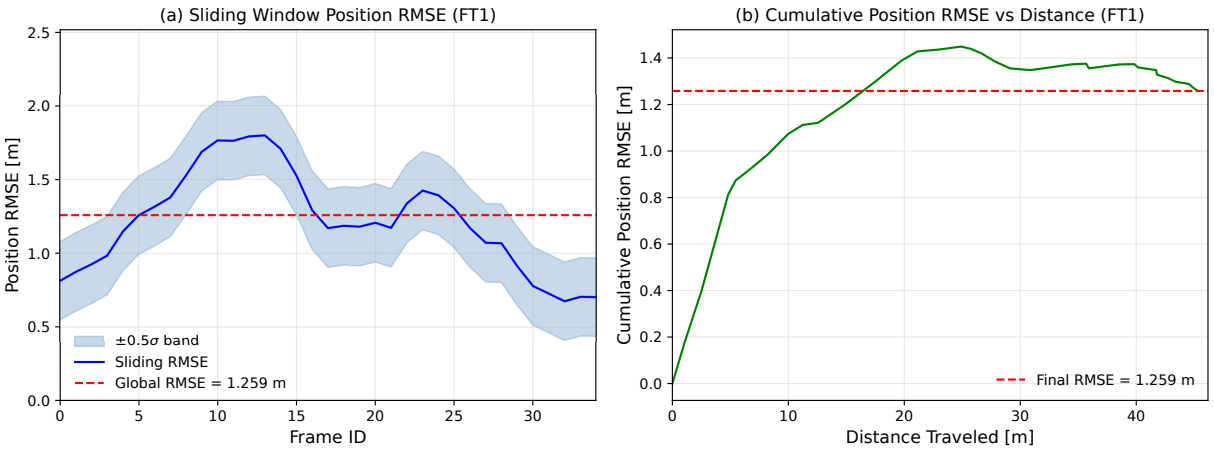


Figure 4.29: Sliding window and cumulative RMSE analysis for Field Test FT1. (a) Sliding window position RMSE across the frame sequence, with the $\pm 0.5\sigma$ uncertainty band shown in light blue. The global RMSE of 1.259 m serves as a reference. The sliding RMSE varies between 0.5 m and 2.0 m, and reaches peak values around frames 10-15 and 20-25 during periods of degraded skyline quality. The recovery toward the end of the sequence (frames 28-35) demonstrates the system's ability to regain accuracy when favorable visual conditions return. (b) Cumulative position RMSE versus distance traveled, shows growth from near-zero to a final RMSE of 1.259 m over approximately 40 m. The curve exhibits an initial rapid rise followed by a plateau around 30 m, and this indicates that most drift accumulation occurs in the middle portion of the trajectory

Table 4.1: Summary of Field Testing Results

Metric	**FT0**	**FT1**
Number of poses	∼2,800	∼7,700
Trajectory length	∼25 m	∼40 m
Position RMSE	**0.369 m**	**1.259 m**
Position error range	0.1–0.6 m	0.5–2.0 m
Maximum position error	<0.7 m	<2.5 m
Orientation RMSE	**12.41°**	**30.47°**
ICP convergence (<12 iter.)	>75%	>75%
Fitness < 0.01	>90%	>90%
RANSAC inlier range	150–350	220–441

by the camera-geometry derived scale factor $\lambda = 0.204$ m/px to achieve metric consistency.

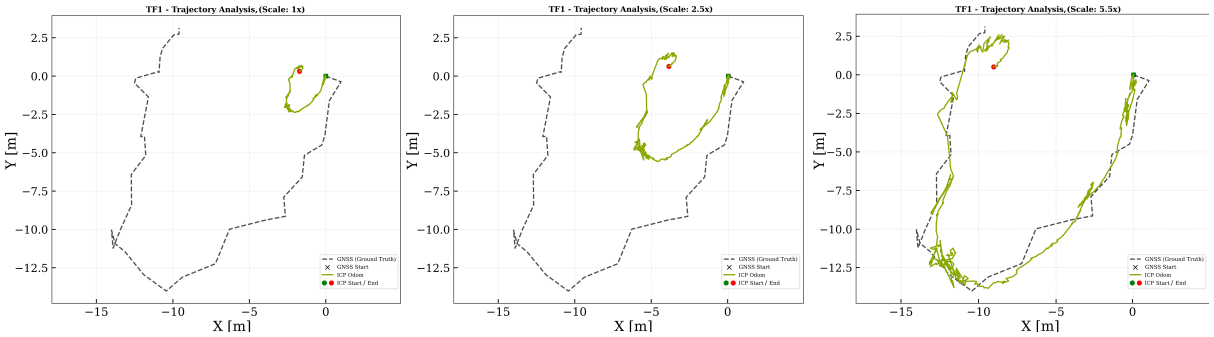


Figure 4.30: Comparison of the FT1 pose-graph under increasing global scale factors. The trajectory transitions from an underscaled reconstruction at Scale 1× (left), through intermediate scaling at 2.5× (center), to a geometry that progressively aligns with the GNSS ground truth at Scale 5.5× (right). The gray dashed line represents the GNSS reference trajectory, while the colored trajectory shows the ICP-derived path with start (green) and end (red) markers. This progression demonstrates the scale-ambiguous nature of skyline-based odometry: while the trajectory shape and topology are mostly captured, absolute metric scale must be recovered through external cues such as GNSS fusion or known landmark distances

5 Conclusion and Future Work

This work presented the development and validation of a software tool for vision-based navigation in low-textured outdoor environments. The system integrates three core modules: Configuration, Execution, and Reporting to deliver a pipeline capable of extracting skylines, generating localization hypotheses, and producing a globally consistent trajectory based solely on visual cues and pose-graph navigation. The results confirm that skyline information, traditionally regarded as a qualitative descriptor, can be transformed into a quantitative navigation signal when coupled with semantic segmentation, numeric similarity metrics, and Iterative Closest Point (ICP) alignment.

The comparative evaluation of the three implemented skyline-extraction strategies demonstrated a clear performance hierarchy. The Alpha (Dynamic Programming) and Beta (heuristic edge-based) methods exhibited accuracy in the range of 81-92% Dice coefficient across the evaluated scenarios. However, the Gamma method, based on DeepLab V3 semantic segmentation, consistently outperformed both, achieving an average Dice score of approximately 96% and reaching peaks of 97-98% in environments with cloud gradients or complex horizon structures. These results substantiate the decision to adopt Gamma as the default skyline-detection method in the Execution Module.

The system's localization process was further strengthened by the integration of a monocular depth estimation model, which complemented the skyline-based geometric cues with dense depth priors for each frame. Instead of relying solely on the Dice- and Hausdorff-based matching between detected skylines and the georeferenced dataset, the pipeline incorporated the estimated depth structure to refine the initial pose hypotheses and enhance the robustness of the silhouette-to-map association. This additional modality helps disambiguate visually similar skyline profiles and improves the spatial coherence of the generated point clouds, particularly in regions where horizon geometry alone exhibits low variance.

The field experiment, spanning more than 2800 frames, demonstrated the stability and reliability of the proposed localization pipeline under real outdoor conditions: the fused ICP–EKF odometry showed point-wise position errors predominantly in the 0.1-0.6 m range, with 99% of samples remaining below 0.7 m, while heading errors stayed bounded within $\pm 0.1^\circ$ and rarely exceeded $\pm 0.1^\circ$ after transient spikes. These results, combined with successful integration tests confirms consistent ROS parameter synchronization, coherent GUI behavior, and correct generation of structured reports. Overall, the findings confirm that visually extracted skylines, enhanced through modern segmentation, pose-graph methods builds a dependable exteroceptive cue for navigation in low-textured environments, offering a compelling alternative to feature-based or LiDAR-dependent approaches when GPS reliability, sensor payload, or environmental texture impose significant constraints.

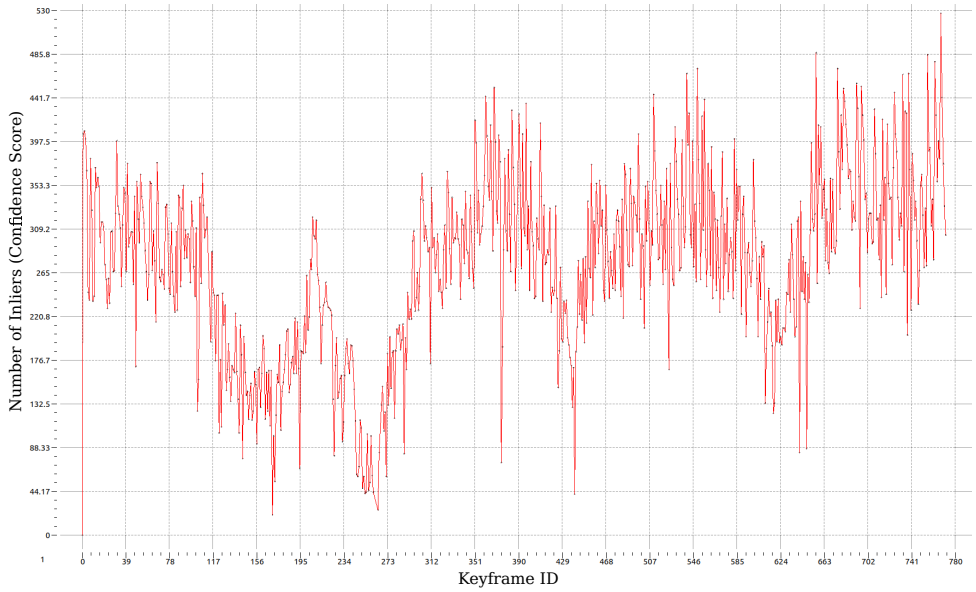


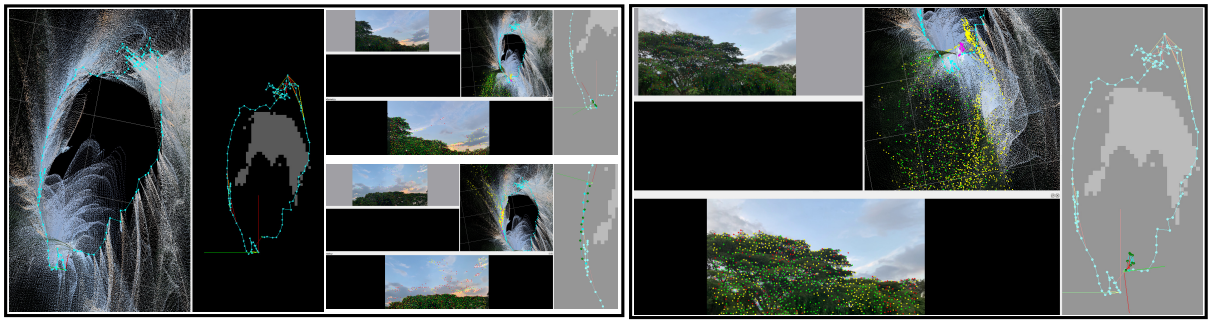
Figure 4.31: RANSAC-verified inlier count across the extended sequence. This inlier regimee dominates levels between 220 and 441 inliers. Few intermittent behavior linked to challenging skyline conditions or blurred camera, and peaks above 500 occasionally at the end of the localization session



Figure 4.32: Offline localization visualization for Field Test FT1 in the area between football fields C14 and C15 at the Universidad del Valle. (a) RViz display of the reconstructed pose-graph trajectory for the extended sequence (over 7,700 poses), showing the ICP-derived path overlaid on the three-dimensional occupancy grid. The trajectory exhibits the characteristic shape of the route captured between football fields C14 and C15 at Universidad del Valle. (b) Contextual alignment view projecting the estimated trajectory onto satellite imagery, which demonstrates the spatial consistency between the skyline-based localization output and the ground-truth environment layout. (c) Aerial view of the FT1 testing area, featuring a more complex environment with denser vegetation, a swimming pool, and varied skyline structures that introduce greater variability in the localization challenge compared to FT0

5.1 Future Work

A direction for future work is to extend the current skyline-based framework with learning-driven components that adapt online to environmental changes. Reinforcement learning could be used to refine pose-hypothesis selection, optimize viewpoint transitions, or adapt ICP parameters based on registration quality. Likewise, integrating stronger semantic models and improved depth priors, along with tighter coupling to global pose-graph methods such as RTAB-Map, may enhance robustness in challenging lighting or even more low-texture conditions. Finally, to deploy and evaluating the system on embedded or multi-robot platforms would help explore its applicability to large-scale, GPS-degraded outdoor environments.



(a) (b)

Figure 4.33: Visualization of the skyline-based localization process: initial setup, intermediate pose-graph updates (a), and final optimized trajectory (b)

References

[1] K. W. Coble, "Skyline delineation for localization in occluded environments," M.S. thesis, KTH Royal Institute of Technology, 2023.

[2] P. De Cristóforis, M. Nitsche, T. Krajník, T. Pire, and M. Mejail, "Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments," *Pattern Recognition Letters*, vol. 53, pp. 118–128, 2015. DOI: https://doi.org/10.1016/j.patrec.2014.10.010.

[3] J. Tang, C. Gong, F. Guo, Z. Yang, and Z. Wu, "Automatic geo-localization framework without gnss data," *IET Image Processing*, vol. 16, no. 8, pp. 2180–2195, 2022. DOI: https://doi.org/10.1049/ipr2.12482.

[4] V. R. Emma, "Skyline matching: Absolute localisation for planetary exploration rovers," M.S. thesis, Universitat Politècnica de Catalunya, BarcelonaTech, 2022.

[5] Q. Peng, X. Zhao, R. Dang, and Z. Xiang, "Rwt-slam: Robust visual slam for weakly textured environments," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, 2024, pp. 913–919. DOI: 10.1109/IV55156.2024.10588822.

[6] S. Williams, "Visual arctic navigation: Techniques for autonomous agents in glacial environments," Ph.D. dissertation, Georgia Institute of Technology, 2011.

[7] W.-N. Lie, T. C.-I. Lin, T.-C. Lin, and K.-S. Hung, "A robust dynamic programming algorithm to extract skyline in images for navigation," *Pattern Recognition Letters*, vol. 26, no. 2, pp. 221–230, 2005. DOI: https://doi.org/10.1016/j.patrec.2004.08.021.

[8] T. Ahmad, G. Bebis, M. Nicolescu, A. Nefian, and T. Fong, "Horizon line detection using supervised learning and edge cues," *Computer Vision and Image Understanding*, vol. 191, p. 102 879, 2020. DOI: https://doi.org/10.1016/j.cviu.2019.102879.

[9] Y.-J. Liu, C.-C. Chiu, and J.-H. Yang, "A robust vision-based skyline detection algorithm under different weather conditions," *IEEE Access*, vol. 5, pp. 22 992–23 009, 2017. DOI: 10.1109/ACCESS.2017.2728826.

[10] S. Fefilatyev, V. Smarodzinava, L. O. Hall, and D. B. Goldgof, "Horizon detection using machine learning techniques," ICMLA '06, pp. 17–21, 2006. DOI: 10.1109/ICMLA.2006.25. [Online]. Available: https://doi.org/10.1109/ICMLA.2006.25.

[11] R. Verbickas and A. Whitehead, "Sky and ground detection using convolutional neural networks," in *Proceedings of the International Conference on Machine Vision and Machine Learning (MVML)*, Paper No. 64, Prague, Czech Republic, 2014.

[12] M. Aladem and S. A. Rawashdeh, "Evaluation of image enhancement techniques for vision-based navigation under low illumination," *Journal of Robotics*, vol. 2019, p. 5 015 741, 2019. DOI: 10.1155/2019/5015741.

[13] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, 2021, ISSN: 1424-8220. DOI: 10.3390/s21062140.

[14] R. Guo, D. Zhou, K. Peng, and Y. Liu, "Plane based visual odometry for structural and low-texture environments using rgb-d sensors," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2019, pp. 1–4. DOI: 10.1109/BIGCOMP.2019.8679500.

[15] S. Daftry et al., "Robust vision-based autonomous navigation, mapping and landing for mavs at night," in *Proceedings of the 2018 International Symposium on Experimental Robotics*, J. Xiao, T. Kröger, and O. Khatib, Eds., Cham: Springer International Publishing, 2020, pp. 232–242, ISBN: 978-3-030-33950-0.

[16] B. Jinsong, Y. Dili, H. Xiaofeng, J. Ye, and D. Sanhu, "The study of hybrid vision-based navigation for lunar rover in virtual environment," in *2009 International Conference on Mechatronics and Automation*, 2009, pp. 655–659. DOI: 10.1109/ICMA.2009.5246437.

[17] B. Nagy, "A new method of improving the azimuth in mountainous terrain by skyline matching," in *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 88, 2020, pp. 121–131. DOI: 10.1007/s41064-020-00093-1.

[18] S. Williams and A. M. Howard, "Horizon line estimation in glacial environments using multiple visual cues," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5887–5892. DOI: 10.1109/ICRA.2011.5980006.

[19] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: Types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct. 2016, ISSN: 2193-1801. DOI: 10.1186/s40064-016-3573-7.

[20] V. Gupta and S. Brennan, "Terrain-based vehicle orientation estimation combining vision and inertial measurements," *J. Field Robotics*, vol. 25, pp. 181–202, Mar. 2008. DOI: 10.1002/rob.20233.

[21] E. P. Herrera-Granda, J. C. Torres-Cantero, and D. H. Peluffo-Ordóñez, "Monocular visual slam, visual odometry, and structure from motion methods applied to 3d reconstruction: A comprehensive survey," *Heliyon*, vol. 10, no. 18, e37356, 2024, ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2024.e37356.

[22] A. Ostovar, O. Ringdahl, and T. Hellström, "Adaptive image thresholding of yellow peppers for a harvesting robot," *Robotics*, vol. 7, no. 1, 2018, ISSN: 2218-6581. DOI: 10.3390/robotics7010011.

[23] A. L. Trifan, A. J. R. Neves, and M. B. Cunha, "The importance of color spaces in robotic vision," *Internal Report*, 2012, Accessed 2025-06-03.

[24] R.-C. Chen, C. Dewi, Y.-C. Zhuang, and J.-K. Chen, "Contrast limited adaptive histogram equalization for recognizing road marking at night based on yolo models," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2023. DOI: 10.1109/ACCESS.2023.3309410.

[25] R. Srisha and A. Khan, "Morphological operations for image processing : Understanding and its applications," Dec. 2013.

[26] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. DOI: 10.1023/B:VISI.0000029664.99615.94.

[27] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[28] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 3, pp. 1623–1637, 2022. DOI: 10.1109/TPAMI.2020.3019967.

[29] J. F. Ferreira, D. Portugal, M. E. Andrada, P. Machado, R. P. Rocha, and P. Peixoto, "Sensing and artificial perception for robots in precision forestry: A survey," *Robotics*, vol. 12, no. 5, 2023, ISSN: 2218-6581. DOI: 10.3390/robotics12050139.

[30] Z. Peng et al., "Rtg-slam: Real-time 3d reconstruction at scale using gaussian splatting," in *ACM SIGGRAPH 2024 Conference Papers*, ser. SIGGRAPH '24, Denver, CO, USA: Association for Computing Machinery, 2024, ISBN: 9798400705250. DOI: 10.1145/3641519.3657455.

[31] R. Ren, H. Fu, and M. Wu, "Large-scale outdoor slam based on 2d lidar," *Electronics*, vol. 8, no. 6, 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8060613.

[32] A. H. Kassem and M. Asem, "A low-cost robotic system for simultaneous localization and mapping," *Journal of Engineering and Applied Science*, vol. 71, no. 1, p. 158, Jul. 2024, ISSN: 2536-9512. DOI: 10.1186/s44147-024-00486-8.

[33] H. Malakouti-Khah, N. Sadeghzadeh-Nokhodberiz, and A. Montazeri, "Simultaneous localization and mapping in a multi-robot system in a dynamic environment with unknown initial correspondence," *Frontiers in Robotics and AI*, vol. 10, p. 1 291 672, 2024. DOI: 10.3389/frobt.2023.1291672.

[34] Y. Shi, Z. Wang, S. Fang, and Y. Zhang, "A survey of point cloud registration based on ICP algorithm," *Procedia Computer Science*, vol. 147, pp. 706–712, 2019. DOI: 10.1016/j.procs.2019.01.205. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919302388.

[35] T. Karimpanal, M. Chamambaz, W. Li, T. Jeruzalski, A. Gupta, and E. Wilhelm, "Adapting low-cost platforms for robotics research," in *2017 International Conference on Advanced Robotics and Mechatronics (ICARM)*, Preprint available at arXiv:1705.07231, IEEE, 2017, pp. 403–408. DOI: 10.1109/ICARM.2017.8273199.

[36] W. Dai et al., "Integrated edge detection and terrain analysis for agricultural terrace delineation from remote sensing images," *International Journal of Geographical Information Science*, vol. 34, no. 3, pp. 484–503, 2020. DOI: 10.1080/13658816.2019.1650363.

[37] F. Guo, J. Tang, C. Gong, and Z. Liu, "Vision-based geo-localization in mountainous regions," in *2022 International Conference on Intelligent Systems and Computational Intelligence (ICISCI)*, 2022, pp. 80–85. DOI: 10.1109/ICISCI53188.2022.9941418.

[38] D. Pedley and J. Morgenroth, "Detecting and measuring fine-scale urban tree canopy loss with deep learning and remote sensing," *ISPRS Open Journal of Photogrammetry and Remote Sensing*, vol. 15, p. 100 082, 2025. DOI: 10.1016/j.ophoto.2025.100082. [Online]. Available: https://doi.org/10.1016/j.ophoto.2025.100082.

[39] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, UK: Cambridge University Press, 2004, ISBN: 0521540518.

[40] A. Geiger, J. Ziegler, and C. Stiller, "Monocular visual odometry using a planar road model to solve scale ambiguity," in *European Conference on Mobile Robots (ECMR)*, 2011, pp. 43–48.

[41] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors*. New York, NY: McGraw-Hill, 2008, ISBN: 978-0-07-149329-1.

[42] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, Jul. 2014.

[43] G. Cross and contributors, *Rviz_satellite: Satellite map tiles in rviz*, https://docs.ros.org/en/iron/p/rviz_satellite/, Version as used in this work (e.g., 4.2.1); Apache-2.0 license, 2025. [Online]. Available: https://docs.ros.org/en/iron/p/rviz_satellite/.

[44] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. DOI: 10.1002/rob.21831.

[45] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual SLAM library for large-scale and long-term online operation," *The International Journal of Robotics Research*, vol. 38, no. 14, pp. 1463–1480, 2019. DOI: 10.1177/0278364919859423. [Online]. Available: https://introlab.3it.usherbrooke.ca/rtabmap/.

6 Annexes

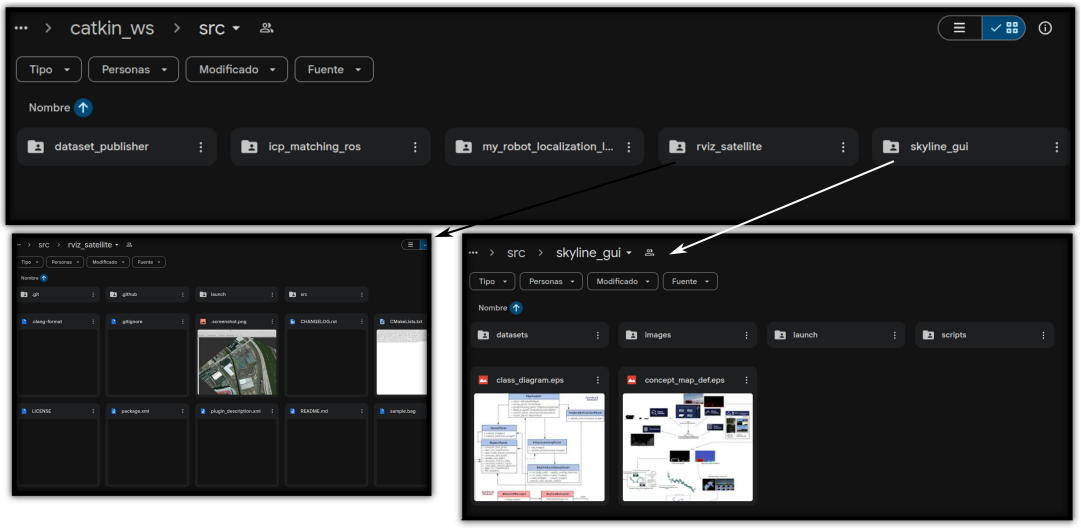


Figure 6.1: Overview of the project repository structure and directory hierarchy

The purpose of this annex folder is to house all relevant materials related to the project, which includes class and conceptual diagrams, documentation, and code. It serves as a central repository for the essential resources that provide insight into the project's design. The purpose of this annex folder is to house all relevant materials related to the project, which includes class and conceptual diagrams, documentation, and code. It serves as a central repository for the essential resources that provide insight into the project's design. It is organized under the following hierarchy: catkin_ws > src. The src directory acts as the central folder that contains the following directories: dataset_publisher, icp_matching_ros, my_robot_localization_launch, rviz_satellite, and skyline_gui. Each of these folders contains the necessary components to execute the main program. Visit the Annex Folder.

6.1 Annex A: User Stories

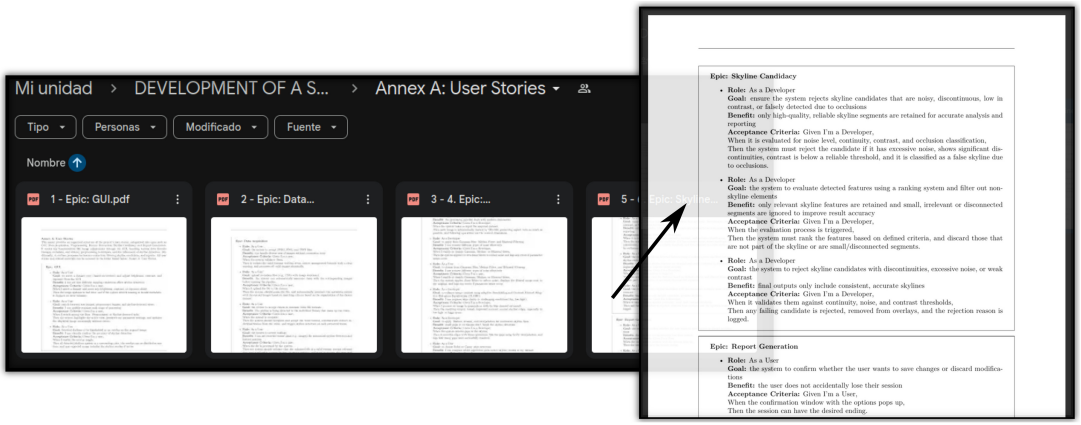


Figure 6.2: Collection of user stories organized by project epics

This annex provides an organized structure of the project's user stories, categorized into epics such as GUI, Data Acquisition, Preprocessing, Feature Extraction, Skyline Candidacy, and Report Generation. It covers key functionalities like image adjustments through the GUI, handling various data formats (images, metadata, and videos), preprocessing techniques, and the refinement of skyline detection. Additionally, it outlines processes for feature extraction, filtering skyline candidates, and reports. All user stories and related materials can be accessed in the folder linked below: Annex A: User Stories.

6.2 Annex B: Literature review backward-forward analysis

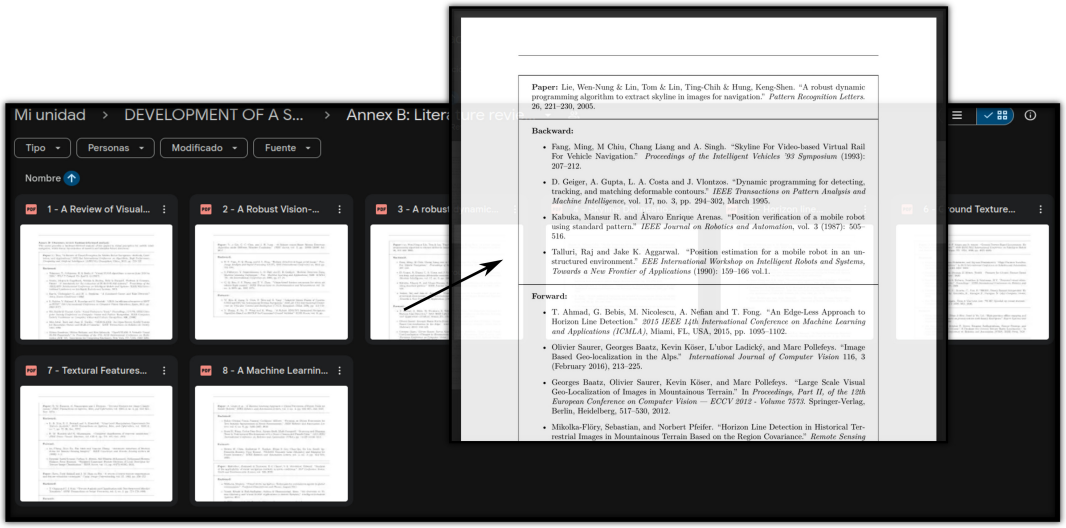


Figure 6.3: Backward-forward analysis records for key literature

This annex provides a backward-forward analysis of key papers in visual perception for mobile robot navigation, which traces the evolution of research and identifies future directions where such key papers are relevant. For more details, visit the full literature review: Annex B: Literature review backward-forward analysis.