UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RAPHAEL VIVACQUA CARNEIRO

GENERATING ROAD GRID MAPS AND PATH PLANS FOR SELF-DRIVING CARS USING LASER REMISSION DATA AND DEEP NEURAL NETWORKS

VITÓRIA – ES

2024

RAPHAEL VIVACQUA CARNEIRO

GENERATING ROAD GRID MAPS AND PATH PLANS FOR SELF-DRIVING CARS USING LASER REMISSION DATA AND DEEP NEURAL NETWORKS

Tese apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Doutor em Ciência da Computação.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ORIENTADOR: PROF. DR. ALBERTO FERREIRA DE SOUZA

VITÓRIA – ES

2024

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Carneiro, Raphael Vivacqua, 1965-
C289g Generating road grid maps and path plans for self-driving cars using laser remission data and deep neural networks / Raphael Vivacqua Carneiro. - 2024.
100 f. : il.

Orientador: Alberto Ferreira De Souza.
Tese (Doutorado em Ciência da Computação) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Inteligência artificial. 2. Redes neurais (Computação). I. De Souza, Alberto Ferreira. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

Generating Road Grid Maps and Path Plans for Self-Driving Cars Using Laser Remission Data and Deep Neural Networks

Raphael Vivacqua Carneiro

Tese de Doutorado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em 24 de setembro de 2024.

gov.br
Documento assinado digitalmente
ALBERTO FERREIRA DE SOUZA
Data: 25/09/2024 19:55:51-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Alberto Ferreira de Souza
Orientador

gov.br
Documento assinado digitalmente
CLAUDINE SANTOS BADUE
Data: 26/09/2024 17:00:12-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Claudine Santos Baduê
Membro Interno

gov.br
Documento assinado digitalmente
THOMAS WALTER RAUBER
Data: 27/09/2024 08:34:53-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Thomas Walter Rauber
Membro Interno

gov.br
Documento assinado digitalmente
KARIN SATIE KOMATI
Data: 27/09/2024 10:49:16-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Karin Satie Komati
Membro Externo

gov.br
Documento assinado digitalmente
MARIELLA BERGER ANDRADE
Data: 27/09/2024 14:00:44-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Mariella Berger Andrade
Membro Externo

(assinar utilizando **obrigatoriamente** o gov.br)

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória/ES, 24 de setembro de 2024

DEDICATÓRIA

À minha amada esposa Teresa Cristina e amados filhos Leonardo e Luciana,

ao meu saudoso pai Dirceu e querida mãe Leda.

AGRADECIMENTOS

Agradeço a Deus por todos os meios e caminhos que se abriram à minha frente ao longo da vida. À minha amada família, pelo apoio e compreensão, que nunca faltaram. À minha querida esposa Teresa Cristina Janes Carneiro, pelo suporte emocional e experiência acadêmica, que muito contribuíram para o sucesso dessa jornada. Ao meu professor orientador Dr. Alberto Ferreira de Souza, por ser uma inspiração constante, com brilhantismo e paixão contagiante pela Ciência e Tecnologia, aliada ao senso humanitário e coletivo. Aos demais professores e colegas do Laboratório de Computação de Alto Desempenho (LCAD) da Universidade Federal do Espírito Santo (UFES), especialmente aos meus companheiros de curso de doutorado: Rânik Guidolini, Pedro Azevedo e Vinicius Cardoso, pela atuação conjunta em projetos relevantes, cujos frutos se estenderam para muito além das paredes da Academia. À minha assistente Ludmila Dias, por sua laboriosa cooperação. Por fim, meu agradecimento ao Ministério da Educação (MEC) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo suporte financeiro fornecido por meio da bolsa de doutorado.

RESUMO

Este trabalho propõe o uso de redes neurais profundas (RNP) para resolver o problema de inferir a localização de vias trafegáveis e as suas propriedades relevantes, como os direitos de mudança de pista, ainda que as marcações de linha estejam pouco visíveis ou ausentes. Este problema é relevante para a operação de carros autônomos que demandam mapas e caminhos precisos. Nossa abordagem para o problema é o uso de RNP para segmentação semântica de mapas de grade de remissão de laser, gerando mapas de grade de vias trafegáveis. Ambos os mapas de grade, os de remissão de laser e os de vias trafegáveis, são matrizes quadradas nas quais cada célula representa características de uma pequena região 2D quadrada do mundo real (p. ex., 20cm x 20cm). Uma célula de um mapa de grade de remissão de laser contém as informações sobre a intensidade média da remissão de laser na superfície daquele local específico. Uma célula de mapa de grade de vias trafegáveis contém as informações semânticas sobre o pertencimento daquela área a uma via trafegável ou a uma marcação de linha na pista ou a uma área não trafegável. Os códigos semânticos associados às células dos mapas de vias trafegáveis contêm informações necessárias para a construção de uma rede de caminhos válidos, necessários para os carros autônomos construírem os seus planos de caminho. A técnica aqui pesquisada é inovadora para a construção automática de planos de caminho viáveis para carros autônomos. Em nossos experimentos, usamos o carro autônomo da UFES, a IARA (*Intelligent Autonomous Robotic Automobile*). Bases de dados anotadas manualmente são usadas para treinar e validar RNP de segmentação semântica para gerar mapas de grade de vias trafegáveis a partir de mapas de grade de remissão de laser. Os resultados obtidos atingiram uma precisão de segmentação de 94,7% nos casos de interesse. Os planos de caminho gerados automaticamente a partir dos mapas de vias trafegáveis inferidos foram testados no mundo real usando a IARA e demonstraram desempenho equivalente aos dos planos de caminho gerados manualmente.

Palavras-chaves: Carros Autônomos; Remissão de Laser; Mapas de Grade; Redes Neurais Profundas; Detecção de Vias; Planejamento de Caminho.

ABSTRACT

This work proposes the use of deep neural networks (DNN) for solving the problem of inferring the location of drivable lanes of roadways and their relevant properties such as the lane change right-of-way, even if the line markings are poor or absent. This problem is relevant to the operation of self-driving cars which requires precise maps and precise path plans. Our approach to the problem is the use of a DNN for semantic segmentation of LiDAR remission grid maps into road grid maps. Both LiDAR remission grid maps and road grid maps are square matrices in which each cell represents features of a small 2D-squared region of the real world (e.g., 20cm × 20cm). A LiDAR remission grid map cell contains the information about the average intensity of laser reflection remission on the surface of that particular place. A road grid map cell contains the semantic information about whether it belongs to either a drivable lane or a line marking or a non-drivable area. The semantic codes associated with the road map cells contain all information required for building a network of valid paths, which are required for self-driving cars to build their path plans. Our proposal is a novel technique for the automatic building of viable path plans for self-driving cars. In our experiments we use the self-driving car of UFES, IARA (Intelligent Autonomous Robotic Automobile). We built datasets of manually marked road lanes and use them to train and validate the DNNs used for the semantic segmentation and the generation of road grid maps from laser remission grid maps. The results achieved an average segmentation accuracy of 94.7% in cases of interest. The path plans automatically generated from the inferred road grid maps were tested in the real world using IARA and has shown performance equivalent to that of manually generated path plans.

Keywords: Self-Driving Cars, Laser Remission, Grid Maps, Deep Neural Networks, Lane Detection, Path Planning.

LIST OF FIGURES

LIST OF TABLES

SUMMARY

1. INTRODUCTION

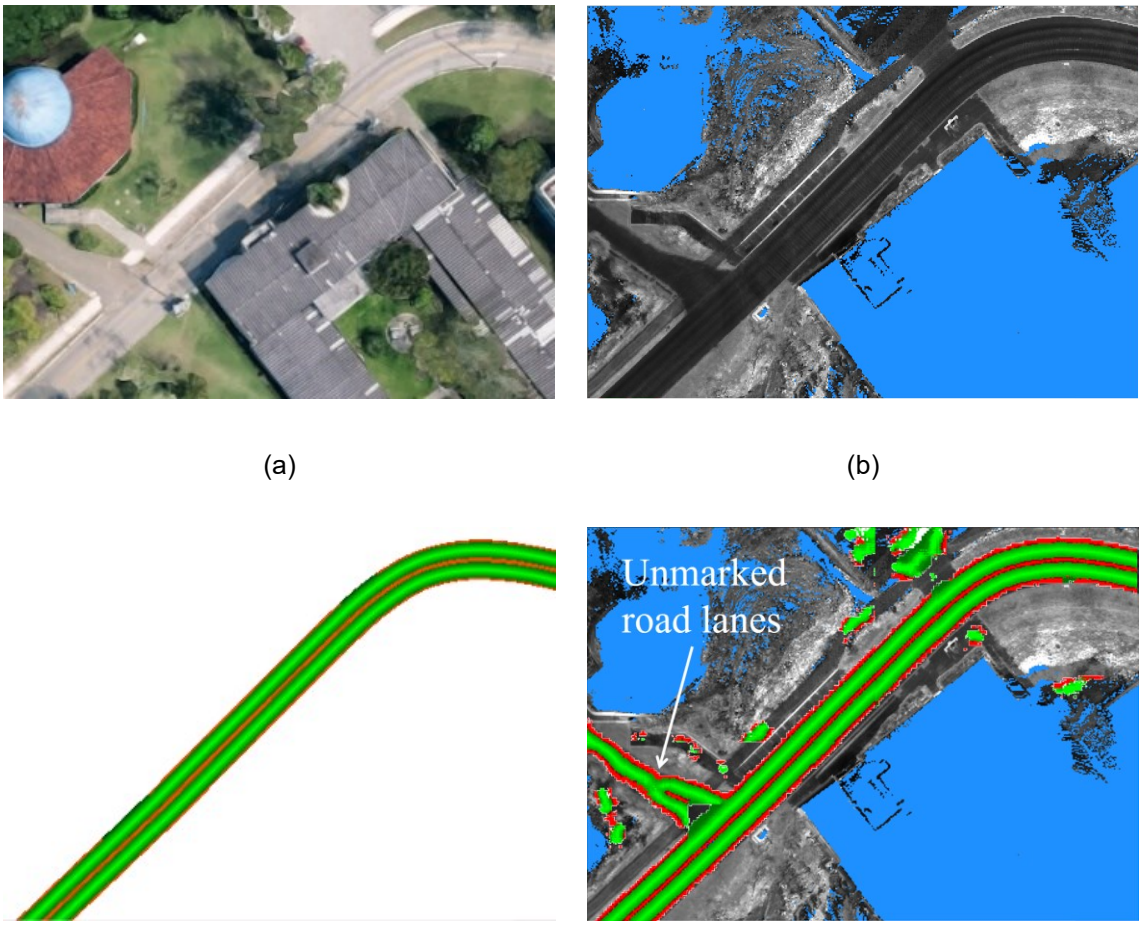
To enable the effective operation of self-driving cars, an internal model of the external environment is crucial. On roadways, a self-driving car must remain within its lane to accommodate other vehicles. This necessitates an internal map of road lanes. Human drivers use horizontal road markings to stay within lanes, and numerous studies have addressed the automatic detection of these markings for Advanced Driver Assistance Systems (ADAS) and autonomous vehicle systems [1]–[8]. However, horizontal markings are often degraded or absent. While this may complicate human driving, it does not make it impossible, as humans can infer lane positions based on contextual information. In contrast, the absence of these markings can render autonomous driving impractical if the vehicle's system relies solely on them.

To overcome this limitation, we propose leveraging deep neural networks (DNNs) to infer the position and characteristics of lanes on roads where horizontal markings are degraded or absent. Our method employs semantic segmentation to process LiDAR (Light Detection and Ranging) remission grid maps (Fig. 1 (b)), which capture the intensity of the reflected laser pulses, converting them into detailed road grid maps (Fig. 1 (c)) that depict the inferred lane structure.

To train the DNNs for this task, we created a dataset covering tens of kilometers of manually annotated road lanes[1]. After training, the model achieved an average segmentation accuracy of 94.7%, reflecting a high proportion of correctly classified cells in the remission grid maps. While this accuracy is not absolute, it underscores the DNN's capability to accurately infer lanes, even when they are missing in the ground truth (Fig. 1 (d)).

[1] Available at https://bit.ly/road_ds.

We tested this approach in the real world using the Intelligent Autonomous Robotic Automobile (IARA, Fig. 3), an autonomous vehicle developed at the Universidade Federal do Espírito Santo (UFES), Brazil. IARA is equipped with the necessary hardware and software for autonomous driving (detailed in Section 3), but until now, it lacked an automatic mechanism for lane inference.



(a) (b)

(c) (d)

Fig. 1. (a) Satellite image of a region of interest. (b) Remission grid map of the same region; areas never scanned by LiDAR beams are shown in blue. (c) Ground truth road map of the same region. (d) DNN-generated road map; some inferred lanes were unmarked in the ground truth.

Previously, IARA relied on Route Data Definition Files (RDDF [9]), with precisely annotated waypoint sequences, to navigate autonomously. With the integration of our

DNN-based module, IARA can now autonomously generate RDDFs from DNN-inferred road maps (Fig. 2).

This system was tested on a 3.7 km stretch of urban roads, and the results were comparable to those produced by manually generated RDDFs, demonstrating the effectiveness of our approach in real-world scenarios.

1.1. Hypothesis

The central hypothesis of this research is that viable path plans for self-driving cars can be automatically generated using LiDAR remission data, pre-processed into remission grid maps, and a pre-trained semantic segmentation DNN capable of transforming these remission grid maps into semantically coded road maps (Fig. 2). This approach enables the DNN to interpret the road structure from the remission data, facilitating the creation of accurate and reliable path plans for autonomous navigation.

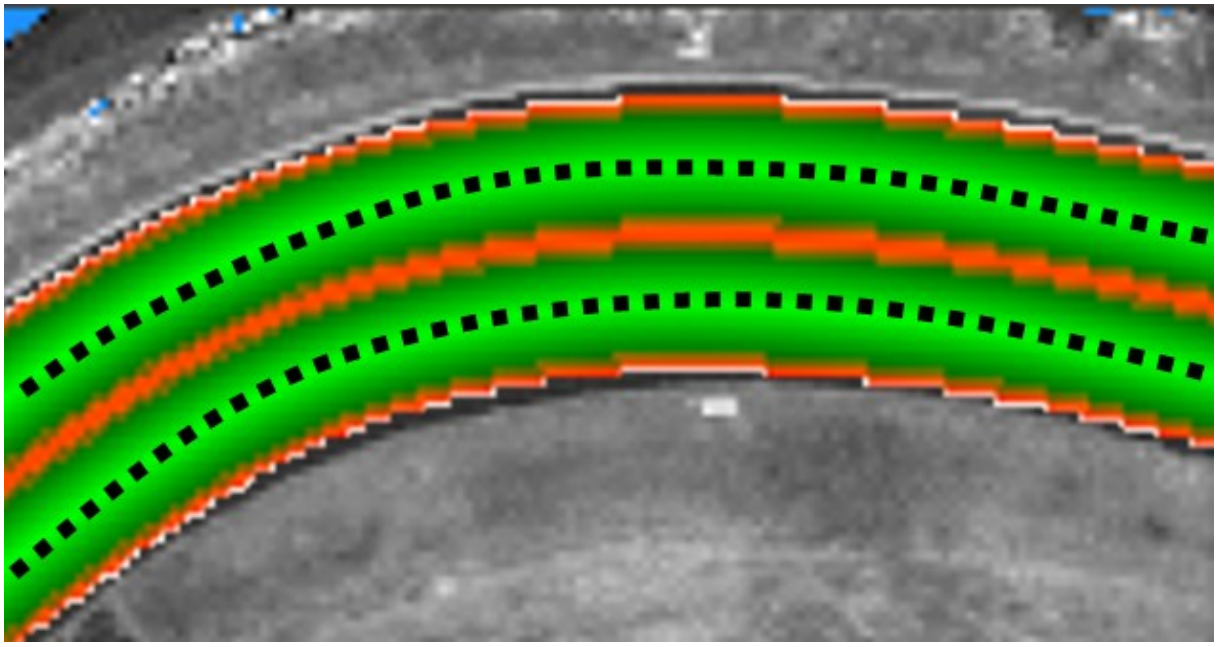


Fig. 2. Example of a semantically coded road grid map, represented in shades of green and red. The black dots correspond to path plans (RDDF waypoints) automatically extracted from the road grid map.

1.2. Objective

The primary objective of this research is to design, develop, and evaluate a novel approach for automatically generating road grid maps and extracting viable path plans from these maps.

To achieve this, we outline the following specific objectives:

- Develop a grid-mapping technique that accurately represents road characteristics.
- Design a method for automatically generating road grid maps from LiDAR remission grid maps.
- Create a technique to automatically extract viable path plans from the generated road grid maps.
- Assess the accuracy of the proposed techniques.
- Test and validate these techniques in real-world scenarios using a self-driving car.

1.3. Contributions

This work presents a series of contributions to the field of autonomous driving, particularly in the areas of road mapping and path planning using LiDAR remission data and deep neural networks (DNNs). The key innovations and advances made in this thesis are as follows:

- **Development of a Road Mapping Methodology from LiDAR Data:** We introduce a novel approach for generating detailed road grid maps from LiDAR remission grid maps. This methodology leverages the intensity of reflected LiDAR pulses to construct accurate semantic representations of road environments. These coding pattern representations capture both the drivable

lanes and non-drivable areas, providing a robust foundation for autonomous vehicle navigation.

- **Semantic Segmentation Using DNNs:** One of the main contributions of this research is the implementation of advanced DNN architectures, such as U-Net, ENet, Swin-Unet, and ST-UNet, for semantic segmentation of LiDAR data. These models are trained to transform LiDAR remission data into semantically coded road grid maps, effectively distinguishing between lanes, road boundaries, and other key road features, even in environments with poor or missing horizontal signalization.

- **Creation of a Comprehensive Dataset:** A substantial dataset comprising tens of kilometers of manually annotated road lanes was developed and made publicly available to the research community. This dataset serves as a benchmark for future research in the field of autonomous driving and road mapping, encouraging the exploration of new models and techniques.

- **Automatic Generation of Route Data Definition Files (RDDFs):** This work provides a mechanism for extracting path plans (RDDFs) directly from the DNN-generated road grid maps. The ability to autonomously generate RDDFs is a critical advancement, as it enables the system to navigate without relying on pre-defined, manually created path plans. This reduces the need for extensive manual input and enhances the flexibility of autonomous systems.

- **Real-World Validation with the IARA Autonomous Car:** The techniques proposed in this thesis were rigorously tested in real-world environments using the IARA (Intelligent Autonomous Robotic Automobile). The experiments demonstrated that the DNN-based road mapping and path planning subsystems performed at a level comparable to manually generated RDDFs. These real-world tests underline the practical viability of the approach and its potential for deployment in real autonomous driving systems.

- **Improved Generalization in Unseen Environments:** Compared to previous work [10], this research addresses the limitations of model generalization to

new and unseen regions. The DNN architectures developed here exhibit improved accuracy and reliability when generating road maps in areas that were not included in the training dataset, making the system more adaptable to diverse driving scenarios.

- **Performance and Scalability:** In addition to enhancing accuracy, this work also focuses on the computational efficiency of the proposed methods. By implementing optimized versions of DNN architectures, the system can operate in real-time environments with lower processing power requirements, making it suitable for deployment in both urban and rural settings.

The contributions outlined in this thesis pave the way for more robust, flexible, and accurate autonomous driving systems, particularly in challenging environments where road markings are either degraded or absent. These advancements represent a significant step forward in reducing the dependency on manual annotations and increasing the autonomy of self-driving vehicles.

The following papers were published during my doctoral program:

- **Raphael V. Carneiro**, Rafael C. Nascimento, Ranik Guidolini, Vinicius B. Cardoso, Thiago Oliveira-Santos, Claudine Badue, Alberto F. De Souza. “Mapping Road Lanes Using Laser Remission and Deep Neural Networks”. In 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE. 2018.

- Ranik Guidolini, **Raphael V. Carneiro**, Claudine Badue, Thiago Oliveira-Santos, Alberto F. De Souza. “Removing Movable Objects from Grid Maps of Self-Driving Cars Using Deep Neural Networks”. In 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE. 2019.

- Claudine Badue, Ranik Guidolini, **Raphael V. Carneiro**, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixão, Filipe Mutz, Lucas Veronese, Thiago Oliveira-Santos, Alberto F. De

Souza. “Self-Driving Cars: A Survey”. Expert Systems with Applications, v. 165, p. 113816, 2021.

- Sabrina S. Panceri, Filipe Mutz, Vinicius B. Cardoso, **Raphael V. Carneiro**, Thiago Oliveira-Santos, Claudine Badue, Alberto F. De Souza. “Detecting Cancerous Tissue in Mammograms Using Deep Neural Networks”. In 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE. 2021.

- João P. A. Andrade, Leonardo S. Paulucio, Thiago M. Paixão, Rodrigo F. Berriel, Teresa C. J. Carneiro, **Raphael V. Carneiro**, Alberto F. De Souza, Claudine Badue, Thiago Oliveira-Santos. “A Machine Learning-based System for Financial Fraud Detection”. In XVIII Encontro Nacional de Inteligência Artificial e Computacional, pp. 165-176. SBC. 2021.

- Pedro Azevedo, **Raphael V. Carneiro**, Alberto F. De Souza, Thiago Oliveira-Santos, Claudine Badue. “A Planning Pipeline for Software Systems of Autonomous Vehicles”. SSRN preprint, 2023. doi: 10.2139/ssrn.4690921.

- **Raphael V. Carneiro**, Ludmila Dias, Thiago Oliveira-Santos, Claudine Badue, Alberto F. De Souza. “Generating Road Grid Maps for Self-Driving Vehicles Using LiDAR Data and Swin Transformers”. SSRN preprint, 2024. doi: 10.2139/ssrn.4996339.

Lastly, in 2019, during my doctoral program, a group of researchers from LCAD Lab at UFES University, Brazil, including myself, created a startup company named Lume Robotics. The founding members were researchers who were actively involved in the Intelligent Autonomous Robotic Automobile (IARA) project, developed at LCAD. We successfully completed numerous proofs of concept and secured two rounds of investment. Additionally, we are currently working on six implementation contracts, in demanding environments such as mining operations, port operations, railway operations, and industrial plants.

1.4. Outline

The work is organized as follows:

- **Section 2: Related Work** – This section reviews previous research on road lane detection and classification methods, covering traditional approaches and AI-based solutions using camera images, LiDAR, and other sensors.

- **Section 3: The IARA Self-Driving Framework** – This section provides a comprehensive description of the IARA system, detailing its hardware and software architectures. Special attention is given to the route and path planning subsystems, which are essential for the scope of this research.

- **Section 4: Proposed Semantic Segmentation of Road Grid Maps** – This section introduces our methodology for generating road grid maps from LiDAR remission data using semantic segmentation. The transformation process from remission grid maps to road grid maps is explained in detail.

- **Section 5: DNNs for Semantic Segmentation of Road Grid Maps** – This section evaluates the deep neural network architectures used in this research. Different models such as U-Net, ENet, Swin-UNet, and ST-UNet are discussed, with design comparisons.

- **Section 6: Applications of Road Grid Maps in Autonomous Vehicle Navigation** – This section focuses on practical applications of road grid maps in real-world autonomous driving scenarios. We discuss enhancements to the IARA system that allow for dynamic path inference and the generation of Route Data Definition Files (RDDFs) from the road grid maps using the new developed algorithms.

- **Section 7: Experimental Methodology** – This section outlines the methodology used to evaluate the proposed techniques. We describe the datasets, experimental setup, and ground-truth marking procedures employed in our validation process.

- **Section 8: Experimental Results** – This section presents and analyzes the results of the experiments, including those conducted with the IARA in real-world environments. Both accuracy metrics and real-time performance evaluations are discussed, highlighting the system's ability to perform autonomous navigation tasks.

- **Section 9: Discussion** – This section delves into the interpretation of the experimental results, providing a critical analysis of their significance and limitations.

- **Section 10: Conclusion** – The final section summarizes the main contributions of this thesis, discusses the implications of the results, and proposes potential directions for future research.

2. RELATED WORK

Numerous solutions have been proposed for detecting and classifying road lanes in images. Recent systematic reviews of lane detection methods can be found in [11] and [12]. Traditional approaches mentioned in these reviews typically include image preprocessing, feature extraction, lane model fitting, and tracking of line markings in camera images. More advanced methods leverage AI-based systems, such as machine learning techniques, deep neural networks, and hybrid approaches. While most methods depend on input data from cameras, other equipment, including radar, GPS, and LiDAR, is also being employed to gather datasets. Cameras, though cost-effective, are sensitive to lighting conditions, whereas LiDAR can detect whether a beam has intercepted asphalt or a road painting, regardless of light levels.

In our review, we classified relevant studies into three primary categories: (i) methods based on lane markings; (ii) methods using image segmentation; and (iii) methods utilizing LiDAR data.

2.1. Methods Based on Lane Markings

In [13], Jung et al. introduce a method that utilizes aligned spatiotemporal images, generated by accumulating the pixels along a scanline over time. The aligned spatiotemporal image is binarized, and two dominant parallel straight lines, resulting from the temporal consistency of lane width on a given scanline, are detected using a Hough transform. The system outputs a cubic spline for both the right and left lanes (in the car's track), besides, each lane class. Similarly, Berriel et al. [3] propose a system that processes a temporal sequence of images, applying inverse perspective mapping and a particle filter for each image to track the lane over time. This system also outputs a cubic spline for both the right and left lanes (in the car's track), and the lane class.

Sultana et al. [14] propose an edge detection method that incorporates a comprehensive intensity threshold range, and a two-step lane verification algorithm based on geometric constraints (angle and length) to validate the characteristics of

lane markings. A lane tracking method predicts the lane position in the next frame by defining a range of horizontal lane positions. Sang and Norris [15] present a self-tuned algorithm that integrates fuzzy logic-based adaptive functions with edge identification and line detection modules, allowing the system to adjust the image in response to challenging weather conditions. The tracking function utilizes previous detection results to refine the selected range of interest.

Bharadwaj et al. [16] employ image calibration and perspective transformation techniques, converting perspective view into a bird's-eye view of the road, and then mathematical procedures to detect and monitor lane lines. Yu et al. [17] propose a lane detection method suitable for diverse road conditions, which preprocesses camera images through grayscale conversion using a weighted average and binarization with Otsu's algorithm. The system uses the Canny operator for edge extraction and applies the Hough transform, and the least squares method to fit straight lines and curved lanes. Alamsyah et al. [18] present a method that uses Canny edge detection and Hough transforms, with clustering via spatio-temporal incremental clustering

Banerjee et al. [19] and Padmaraja et al. [20] focus on noise removal and edge detection for lane detection using the Canny operator and Hough transform. Garg et al. [21] propose using gamma correction to enhance contrast and improve lane recognition at night. Bin Abdul Razak et al. [22] suggest a combination of Gaussian blur, masking, Canny edge detection, and Hough transform for lane detection.

More recently, Convolutional Neural Networks (CNNs) have surpassed traditional image processing methods in accuracy. Lee et al. [23] developed a dataset with 20,000 images with lanes labeled and trained a CNN to detect and track lanes and road markings at a pixel segmentation level, producing a confidence map for lane detection and classification. Yalabaka et al. [24] propose a CNN model that extracts necessary features from camera images to identify lanes, while Luo et al. [25] utilize a transformer-based approach for 3D-lane detection from front-view monocular images. Yao et al. [26] introduce a CNN with a transformer encoder to capture global context by extracting local anchor features from camera images.

Li et al. [27] propose a multi-lane detection method by fusing factorized and depth-wise convolution, increasing the network's receptive field. The approach employs location-based instance detection to flexibly distinguish lanes using the lane start locations at the image boundaries. Alzubaidi et al. [28] explore spatial variance using a hyperbolic space technique combined with a neural network and bidirectional Long Short-Term Memory (LSTM) model for lane detection. Islam et al. [29] present a CNN for recognizing lane and road border markings after applying Canny edge detection and Sobel operators. Rath et al. [30] propose a CNN-based lane detection method that classifies camera image pixels as lane marking or background, followed by post-processing with Canny edge detection, Hough transform, clustering and curve fitting.

Moraes et al. [31] present a real-time image-based path planner using CNN and a path model to infer paths from camera images, while Tabelini et al. [32] leverage temporal information by integrating lane marking detection into a deep neural network model that extracts features from multiple video frames.

However, these methods are limited to the presence of lane markings, do not generate a full road map, and have not been tested on real autonomous vehicles.

2.2. Methods Based on Image Segmentation

A second category of solutions focuses on detecting roads through pixel-level segmentation. Wang et al. [33] propose a road detection algorithm that generates a pixel-level confidence map using likelihood estimation. Laddha et al. [34] introduce an automatic dataset generation method, for pixel-level road image annotations, leveraging OpenStreetMap data, vehicle pose, and camera parameters to train a road detection CNN. Teichmann et al. [35] present an end-to-end approach that integrates classification, detection, and semantic segmentation within a unified architecture, where a shared encoder extracts rich abstract features. The model includes three decoders: one classifies the image as highway or minor road, the second detects vehicles, and the third segments road pixels.

Wang et al. [36] propose a Siamese fully convolutional network (FCN) for road segmentation, which takes an RGB image and a semantic contour map (generated via fast contour detection) as input to output pixel-level road regions. Zhang et al. [37] present an instance segmentation method that uses split attention to improve the feature representation for slender and sparse annotations, such as lane markings. The method incorporates self-attention distillation to enhance the network's ability to represent features.

Zhou et al. [38] introduce a lane detection technique that represents lanes as a series of line segment endpoints. A specially designed loss function detects semantic lanes using heatmaps, and an unsupervised embedding vector groups the endpoints into lanes. A non-maximum suppression algorithm is applied to merge the heatmap and embedding vectors to produce final lane predictions.

Suresh et al. [39] propose an entropy-based fusion model that improves lane and object detection results through CNN-based segmentation. A clustering method groups detected points, and a best-fit line is applied in the mean square sense to detect lanes. Zhang et al. [40] use a DNN with a feature shift aggregator between encoder and decoder to enhance prediction accuracy, employing data augmentation and a filter estimator to address the problem of glitches during lane changes.

Gautam et al. [41] use a DNN for semantic segmentation to classify 14 different object classes, including the background, from camera images. Long et al. [42] propose a geometric attention-aware network for lane detection, employing a multi-task branch architecture with attention information propagation between branches. This method combines semantic segmentation with geometric distance embedding to improve lane detection, particularly in challenging conditions.

However, these methods do not generate a complete road map and have not been tested on real autonomous vehicles.

2.3. Methods Based on LiDAR Data

Several solutions have been proposed that rely primarily on LiDAR data for road detection. Han et al. [43] present a method that fuses LiDAR and image data. Their approach begins by projecting LiDAR point clouds onto monocular images using cross-calibration and a joint bilateral filter, resulting in high-resolution height data. Next, pixels classification is performed using scores computed from an Adaboost classifier, leveraging features such as unary potential, pixel height, color information, and pairwise potential. This system detects roads on both images and LiDAR point cloud.

Caltagirone et al. [44] propose a deep learning-based road detection approach using only LiDAR point clouds. Their method employs a fully convolutional network (FCN) that processes top-view LiDAR images, encoding basic statistics such as mean elevation and density. This approach reduces the problem to a single-scale problem. Their FCN is designed for pixel-wise semantic segmentation, combining a large receptive field with high-resolution feature maps. This system classifies LiDAR points belonging to the road. In a subsequent work [45], Caltagirone et al. integrate LiDAR point clouds, GPS-IMU data, and Google driving directions to generate driving paths. This system uses an FCN to jointly learn perception and path generation from real-world driving sequences, outputting classifications of LiDAR points that belong to the path. Despite de advancements, these methods do not focus on lane detection or generate comprehensive road maps, and they were not evaluated on real autonomous vehicles.

Hernández et al. [46] develop a system based on 2D LiDAR reflectivity to detect roads and lanes, though it does not produce a road map and was not tested in autonomous driving conditions. Similarly, Kim et al. [47] employ 2D LiDAR reflectivity for lane detection and map generation. Joshi and James [6] combine 3D LiDAR data with OpenStreetMap information to apply a particle filter-based method, estimating lane centerlines and creating road maps, but without classifying road lanes. Gwon et al. [8] use 3D LiDAR intensity and GPS to extract lane markings, estimate road geometry, and produce road map representations with polynomial curves, which were tested on a real autonomous car. Reddy et al. [48] implement lane marking detection

algorithms using LiDAR point clouds and trajectory data, while Huang et al. [49] propose a real-time lane marking detection method by classifying road and curb points from LiDAR data.

However, these approaches rely on classical machine learning methods with human-defined feature extractors, which often underperform compared to DNNs in visual recognition challenges such as Pascal VOC and ImageNet.

3. THE IARA SELF-DRIVING FRAMEWORK

This section provides a description of the Intelligent Autonomous Robotic Automobile (IARA)'s hardware and software framework. IARA is an autonomous vehicle that has been developed since 2009 for scientific purposes by researchers of the High-Performance Computing Lab (LCAD) at the Universidade Federal do Espírito Santo (UFES). IARA's architecture is also described in [50].

IARA's software architecture is based on CARMEN [51], an acronym for Carnegie Mellon Navigation Toolkit, developed by researchers of the Carnegie Mellon University, located in Pittsburgh, USA. CARMEN is an open-source collection of modular software for mobile robot control, designed to provide basic navigation functionalities such as sensor control, logging, mapping, localization, path planning and obstacle avoidance. This framework has been reengineered into the CARMEN-LCAD System, with free access for the public[2], and comprising the modules presented in Fig. 6. Since its inception in 2010, the CARMEN-LCAD system has provided autonomous capabilities to the IARA and other autonomous robots as well.

The path planning based on road grid maps presented in this thesis was integrated into CARMEN-LCAD and evaluated in the real world using the IARA. The remainder of this section is organized as follows. The hardware and software architectures of IARA are described in Sections 3.1 and 3.2, respectively. The IARA's route planning and path planning are described in Section 3.3. The non-automated techniques for creating routes for IARA are described in Section 3.4.

[2] Available at https://github.com/LCAD-UFES/carmen_lcad.

3.1. The IARA's Hardware Architecture

IARA is built upon a standard mass-produced passenger car, model Ford Escape Hybrid 2011. The low-level automation was implemented by Torc Robotics[3] to enable the vehicle to receive driving commands from a computer, such as the steering wheel movements, the throttle, and braking actions. The research in the LCAD/UFES improved the automation, adding new hardware modules.

An embedded workstation, model Dell Precision R5500 with 2 Xeon X5690 six-core 3.4GHz processors and one NVIDIA GeForce GTX-1030, powered by the 330-Volt battery of the hybrid car, runs the CARMEN-LCAD self-driving system software.

IARA's sensors comprise wheel's odometers, two multi-layer LiDARs (Light Detection and Ranging, models Velodyne HDL-32E and SICK LD-MRS), stereo cameras, IMU (Inertial Measurement Unit, model Xsens MTi), and a dual RTK GPS (Real-Time Kinematic Global Positioning System, based on model Trimble BD982 receiver) as illustrated in Fig. 3. For conciseness, these sensors will be referred just by odometer, LiDAR, camera, IMU, and GPS, from now on.

The odometer reads the car's linear velocity and the steering wheel angle.

The LiDAR located on the car top (Velodyne HDL-32E) has 32 laser beams assembled one above the other, vertically aligned. Each laser beam measures the distance (range) to the surface hit by the beam, and the intensity of the reflection (reflectivity) of the beam on that surface. The set of 32 range and reflectivity readings are assumed to be obtained at the same time and it is referred as a shot. A rotating motor turns the 32 laser beams around the vertical axis. For each shot, the sensor also returns the horizontal angle of the rotating motor. The set of all shots making a complete 360° round is referred as a point cloud. By turning round and round, the point

[3] https://torc.ai

clouds can provide a 3D-view of the environment around the vehicle. Fig. 4 presents examples of point clouds captured by IARA's LiDAR on the car top.

Fig. 3. The Intelligent Autonomous Robotic Automobile (IARA) of LCAD/UFES. IARA uses several sensors to build an internal representation of the external world. A video of IARA's autonomous operation is available[4].

The LiDAR located at the car front (SICK LD-MRS) has 4 laser beams. It is used for long-range obstacle avoidance in a narrow field of view. The LiDAR on the car top is also used for mid-range obstacle avoidance in a 360° horizontal field of view. So, for that purpose, both LiDARs work in a redundant way in a certain range and field of view.

The GPS reads the position in the world coordinate system. If the car velocity is above a defined minimum value, the car orientation (x and y axes) is estimated using two consecutive GPS readings.

[4] Available at https://bit.ly/iara_mpp.

The IMU reads the acceleration and angular velocity in the directions of the x, y, and z axes, as well as the orientation in the world coordinate system using a magnetometer.

The odometer, the GPS, the IMU, and the LiDAR located on the car top are the sensors used for the Simultaneous Localization and Mapping (SLAM) algorithms, considering the fusion of all the readings.

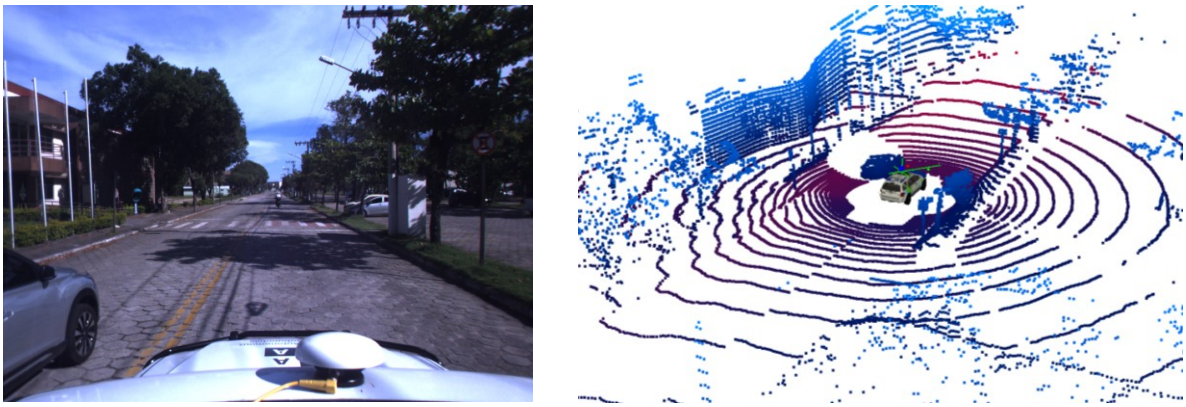


Fig. 4. Examples of camera images and LiDAR point clouds captured by IARA's sensors.

The camera images are used for the computer vision algorithms, using deep neural networks for detection, classification and tracking of objects around the autonomous vehicle, such as pedestrians, semaphores, and other vehicles. The cameras may capture images in various resolutions: VGA (640x480 pixels), HD (1366x720 pixels) and Full HD (1920x1080 pixels).

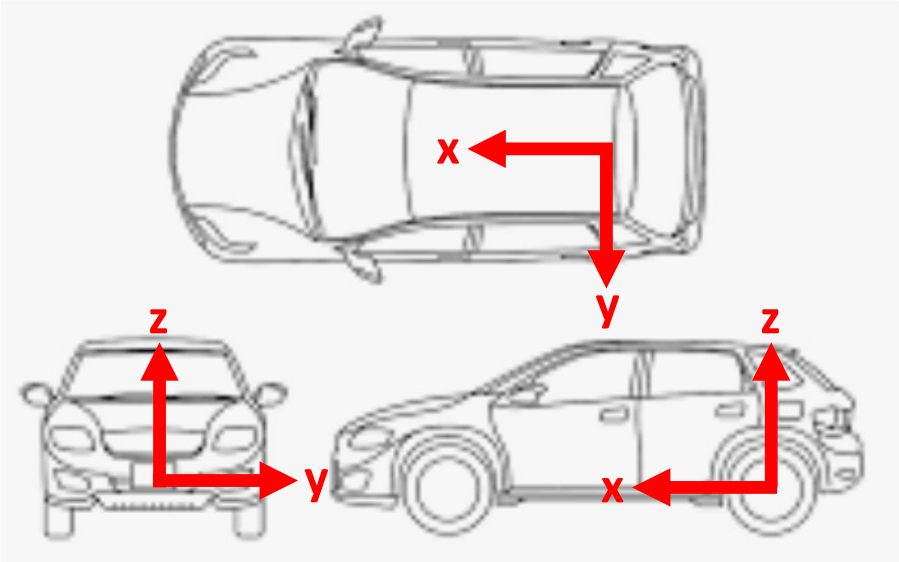


Fig. 5. The IARA's coordinate system.

The IARA's coordinate system's origin is defined to be located at the middle of the car's rear axle. The IARA's x axis points forward, the y axis points to the left, and the z axis points upwards (Fig. 5). Each sensor has its own coordinate system's origin and axis orientations. So, the relative poses of each sensor are manually measured, in relation to the IARA's coordinate system, and the proper translation and rotation operations are performed to integrate all readings.

3.2. The IARA's Software Architecture

By the time the IARA project started in 2009, the Carnegie Mellon Navigation Toolkit (CARMEN [51]) was chosen as the development framework because it was more stable and simpler than emerging platforms. The project extended it and created a new version named CARMEN-LCAD[5] that was used to develop the IARA's autonomous system. When in autonomous mode, the IARA system is currently

[5] Available at https://github.com/LCAD-UFES/carmen_lcad.

composed of more than 25 modules (programs) that work together thanks to the Inter-Process Communication (IPC [52]) library. The IPC implements the publish-subscribe paradigm.

We briefly describe here below the IARA's software modules that are relevant for the comprehension of the current study (Fig. 6).

The Mapper module [53] functions in two different modes: offline and online. While in offline mode, the Mapper module usually takes as input the sensor data previously logged when the vehicle was manually operated, and then generate as output the grid maps of the surrounding locations where the sensor data was captured. These grid maps, referred to as offline maps, aim to depict the static features of the environment as accurately as possible, and are stored in hard disks for later use. The most common types of grid maps generated by the Mapper module are the occupancy grid map, in which each map cell contains the information about the probability of that small region of the world being occupied by some obstacle (e.g. a barrier, a pole, a curb, a deep hole, et cetera); and the remission grid map, in which each map cell contains the information about the intensity of the reflectivity of the laser beams that hit the surface of that small region of the world. The remission grid maps are computed using a technique like that of Levinson and Thrun [54].

The Localizer module [55] is responsible for estimating the autonomous vehicle's current pose, including the current coordinates and the orientation to which the vehicle is heading (orientation equal zero means North, ½ π means West, -½ π means East, -π means South), in relation to the current map. At system startup, the Localizer module usually does not have a current map to which relate, so it performs a global localization [56] using GPS data only and publishes it to the modules that need this information (subscriber modules). Afterwards, whether the autonomous vehicle is moving or not, the Localizer module estimates the current pose using a probabilistic approach, by performing position tracking using a particle filter based on offline occupancy grid maps and LiDAR point clouds [55] and publishes it to the subscriber modules. It performs the pose estimation in a cyclic manner, according to the LiDAR point cloud publishing frequency (usually at 20 Hz).

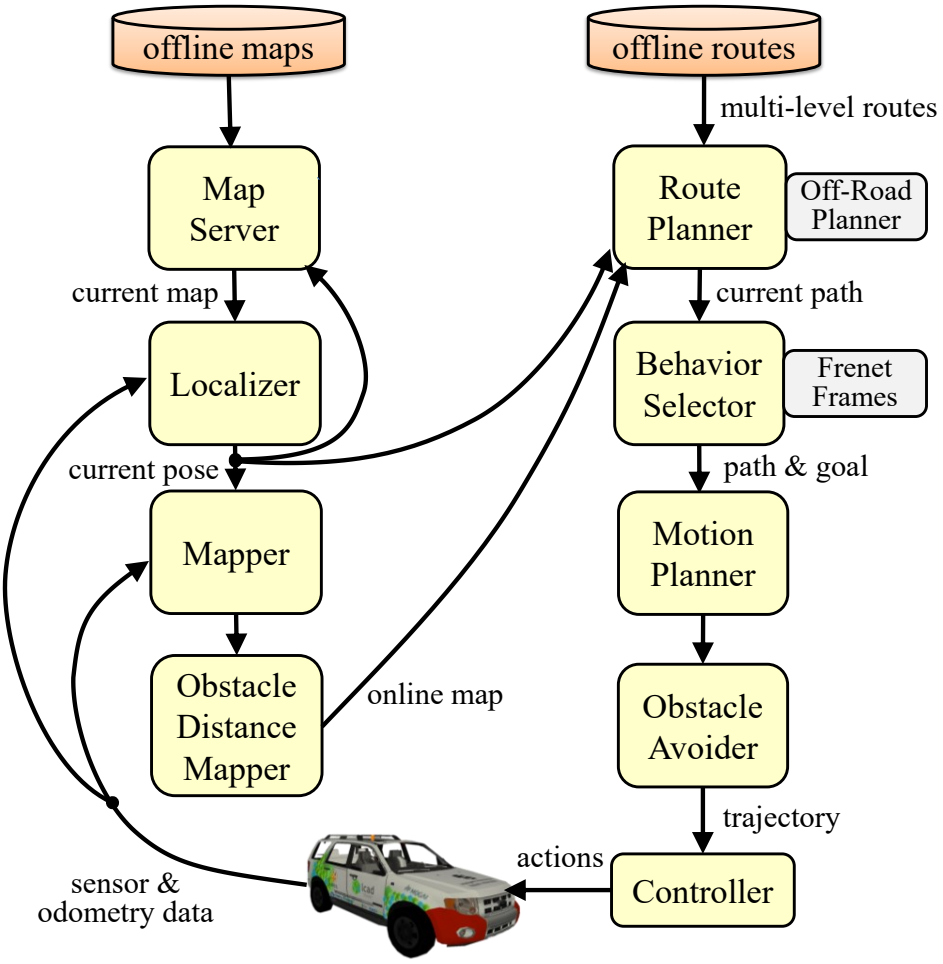


Fig. 6. IARA's software simplified block diagram.

The Map Server module uses the initial pose published by the Localizer module to retrieve the offline maps available from the disk: the occupancy grid map and the remission grid map corresponding to the pose coordinates and publishes them to the modules that need this information (subscriber modules). Each published map represents a world region of dimensions 210 m × 210 m, and combines nine stored submaps of dimensions 70 m × 70 m. The Map Server module fetches the submaps from disk in such way as to have the autonomous vehicle in the central submap. As the vehicle moves, the Localizer module keeps publishing its updated current pose, and at the same time the Map Server module may fetch new submaps, so to keep the autonomous vehicle always within the central submap (Fig. 7 – see also the example

video[6]). Every time the autonomous vehicle crosses the limits of the central submap, new submaps are fetched from disk and a new grid map is built and published. Other types of grid maps that have the same dimensions and coordinates as of the occupancy grid maps and the remission grid maps, might be available from disk; if so, all of them will be published together by the Map Server module.

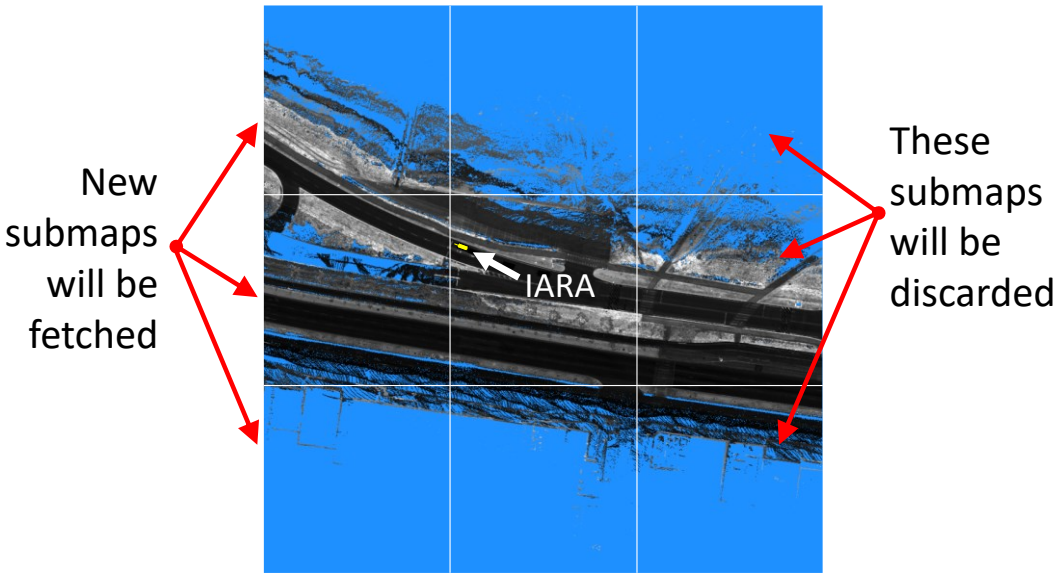


Fig. 7. IARA's Map Server module fetching and discarding submaps as needed. The vehicle is depicted as a small rectangle, moving in the direction indicated by the white arrow.

The Mapper module switches to online mode during autonomous operation. In this mode, real-time sensor data is used to create online grid maps representing the characteristics of the areas scanned by the sensors embedded in the autonomous vehicle, including the presence of both static and moving obstacles, such as pedestrians and nearby vehicles. These online maps are then combined with the offline maps to generate merged maps that incorporate both real-time and pre-existing information about the environment. The merged grid maps provide details about both static and dynamic elements [53].

[6] Available at https://youtu.be/dkYKzvSZWVQ.

The Obstacle Distance Mapper module processes the online merged grid maps to generate a different type of online map, where each map cell contains the distance of the nearest obstacle relative to that cell. These obstacle distance grid maps are then published, allowing dynamic obstacles to be considered by modules responsible for the decision-making and planning tasks, such as the Behavior Selector, Motion Planner, and Obstacle Avoider modules.

The Route Planner module is detailed in the following section. In summary, the Route Planner module retrieves all available routes from disk. These routes were previously created using non-automated techniques such as hand-drawing graphical tools or pose-estimation based on logged sensors and vehicle odometry data. Using the available routes, the module employs the A* search algorithm to determine the optimal path from the current pose to the specified destination. Routes may exist at multiple levels, but the final output must be expressed at the lowest level. This output is a path consisting of a sequence of waypoints from the origin to the destination, with an average distance of 0.5 meters between each consecutive waypoint.

If either the origin pose or the specified destination is not close enough to any waypoint of the main path computed by the Route Plan module, the Off-Road Planner module computes connecting paths, using obstacle distance grid maps and advanced search algorithms [57]. This ensures seamless transitions and accurate path planning, even in off-road areas. These connecting paths are then combined with the main path, generating a complete path from the origin to the destination.

The Behavior Selector module subscribes to the current path provided by the Route Planner module and the obstacle distance grid maps, which considers both static and moving obstacles. It also gathers other relevant information, such as annotations placed along the route (e.g., speed limits, speed bumps, barriers), traffic signal status, pedestrian crossing status, and more. Using this information, the Behavior Selector module decides the next driving behavior, such as lane keeping, overtaking, or stopping before a traffic sign or an obstacle. The Frenet Frames Path Planner [58] is used to provide alternative paths to avoid obstacles, offering options a short distance to the left and right of the current path. Finaly, the Behavior Selector

module publishes the selected path among the alternatives and a goal consisting of a pose to be reached within a few seconds (usually five seconds ahead) according to the selected path plan, along with the desired velocity for the autonomous vehicle by the time it reaches that pose.

The Motion Planner module [59] subscribes to the path and goal published by the Behavior Selector module and plans the vehicle's movements from its current state (pose, velocity, and steering angle) to a state as close as possible to the goal. The motion plan consists of a trajectory made up of a sequence of control commands, including the desired linear velocity, steering wheel angle, and execution time. It employs a model-predictive approach that follows the path provided by the Behavior Selector module while satisfying the vehicle's kinematic and dynamic constraints. The Motion Planner module computes and publishes the optimum motion plan cyclically (typically 20 times per second) to keep the autonomous vehicle as close as possible to the path plan while maintaining a safe distance from both static and dynamic obstacles.

The Obstacle Avoider module [60] subscribes to these motion plans and updates them as needed to avoid trajectories that could lead to collisions. It simulates the execution of the trajectory along the obstacle distance map. If the trajectory intercepts an obstacle, the module adjusts by decreasing the linear velocity of the commands to prevent a collision.

The Controller module [61] subscribes to the trajectories updated by the Obstacle Avoider module and computes action commands for the steering wheel, throttle, and brake actuators, which are sent directly to the vehicle. It employs a Proportional Integral Derivative (PID) approach. Using the planned trajectory and the vehicle's odometry readings, the Controller module estimates an error between the vehicle's actual steering wheel angle and velocity and those specified in the trajectory. New action commands are then computed to minimize this error.

3.3. The IARA's Route Planning and Path Planning

Route planning and path planning both refer to the processes of determining the most efficient way from a starting point to a destination. This involves considering various factors such as distance, travel time, road conditions, traffic patterns, and specific requirements like avoiding certain areas. Route planning is essential for logistics, navigation systems, and autonomous vehicles to optimize travel routes, reduce travel time, and improve overall efficiency. Path planning is crucial in robotics, navigation systems, and autonomous vehicles to ensure safe, efficient, and reliable movement through complex environments.

While route planning and path planning share common goals and methodologies, they differ primarily in their applications and focus:

- Scope and application: Route planning is typically used in larger-scale navigation, such as determining the best route for vehicles on road networks. It focuses on finding the optimal path over a network of predefined routes considering traffic, road conditions, and other factors. Path planning is commonly used in robotics and autonomous systems to navigate through a specific environment. It focuses on navigating through spaces with obstacles and dynamic elements, requiring real-time adjustments.

- Granularity: Route planning operates on a macro level, dealing with entire road networks and infrastructure. Path planning operates on a micro level, dealing with precise movements and navigation through detailed and complex environments.

- Techniques and algorithms: Route planning uses algorithms like Dijkstra's, A*, and various heuristic methods to find the shortest or fastest route on a map. Path planning uses techniques such as Rapidly-exploring Random Trees (RRT), Potential Fields, and occupancy grid mapping to navigate around obstacles and plan feasible paths in real-time.

- Environmental interaction: Route planning is primarily concerned with static road networks and may include real-time traffic updates. Path Planning continuously interacts with dynamic environments, requiring frequent updates and re-planning to avoid obstacles and adapt to changes.

- Output: Route planning provides a sequence of roads and turns to follow, often used in GPS navigation systems. Path planning provides a detailed trajectory or series of waypoints that must be followed to reach its destination safely.

The primary function of the IARA's Route Planner module is to perform route planning at a macro level, handling the entire static road network. Additionally, it executes the initial stage of the path planning pipeline [58] at a micro level, selecting a series of detailed waypoints to follow. Each function uses a specific level of the offline routes, which were previously created using non-automated techniques and stored in disk files as a road network. The macro-level planning operates at the lane level of the road network (Fig. 8), while the micro-level planning operates at the waypoint level of the road network (Fig. 9).

The IARA's road network is structured as a weighted directed graph. Each vertex represents a valid waypoint, and each edge connecting two consecutive vertices represents a valid path segment. The weight of and edge indicates the cost of traversing that specific path segment. Currently, the implemented cost function is the edge's length, which is the physical distance between its two vertices. However, the system architecture allows for future implementation of more sophisticated cost functions, such as the average time spent traversing each edge, or the average expenditure for traversing each edge (including factors like fuel consumption, toll fees, vehicle maintenance and depreciation).

Fig. 8 illustrates an example of the lane-level representation of a road network, denoted as $R_L = \{\mathrm{v}_1, \ldots, \mathrm{v}_j, \ldots, \mathrm{v}_{|R|}\}$, where each v_j is a vertex of the lane-level graph, represented by a tuple $\mathrm{v}_j = (x, y, \theta, d)$, where x and y are the map coordinates, θ is the orientation angle in relation to North, and d is the direction of movement (forward

or backward). In this example, there are two starting vertices (v_1 and v_6), three ending vertices (v_4, v_5, and v_8), three forking vertices (v_2, v_3, and v_7), and two merging vertices (v_5 and v_7). When searching for routes using this graph, the forking and merging vertices might act as either starting or ending vertices.

Each edge is represented by a tuple $E_j = (v_i, v_f, c)$, where v_i and v_f are the initial and final vertices, respectively, and c denotes the cost of traversing the road segment represented by that edge. In a lane-level graph like this one, each edge represents a road segment that may include curves and ramps, but no detours.

The system assumes there is only one possible sequence of waypoints that connects the vertices v_i and v_f of any edge. The detailed sequence of waypoints corresponding to each lane-level edge is defined in the waypoint-level graph (Fig. 9).

Fig. 9 illustrates an example of the waypoint-level representation of a road network, denoted as $R_W = \{\mathrm{w}_1, \ldots, \mathrm{w}_j, \ldots, \mathrm{w}_{|R|}\}$, where each w_j is a vertex in the waypoint-level graph. In this example, each one of the eight vertices numbered v_1 to v_8 in the lane-level graph (Fig. 8) corresponds to a vertex w_j with the same pose but different incoming and outgoing edges.

The lane-level graph shown in Fig. 8 consists of only eight vertices and eight edges, with each edge representing a road segment ranging from a few tens to a few hundred meters in length. In contrast, the waypoint-level graph depicted in Fig. 9 contains several hundred vertices and edges, with each edge representing a road segment averaging 0.5 meters in length.

For improved performance, the Route Planner module searches for the routes using the lane-level road network graph, which has significantly fewer vertices and edges compared to the waypoint-level graph.

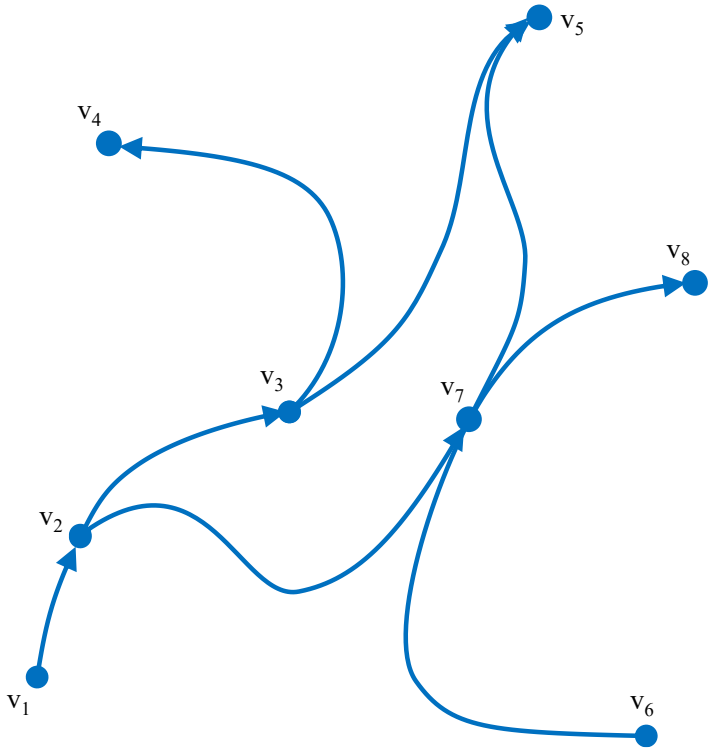


Fig. 8. Example of the lane level of a road network. Each solid arrow (edge) is a single lane.

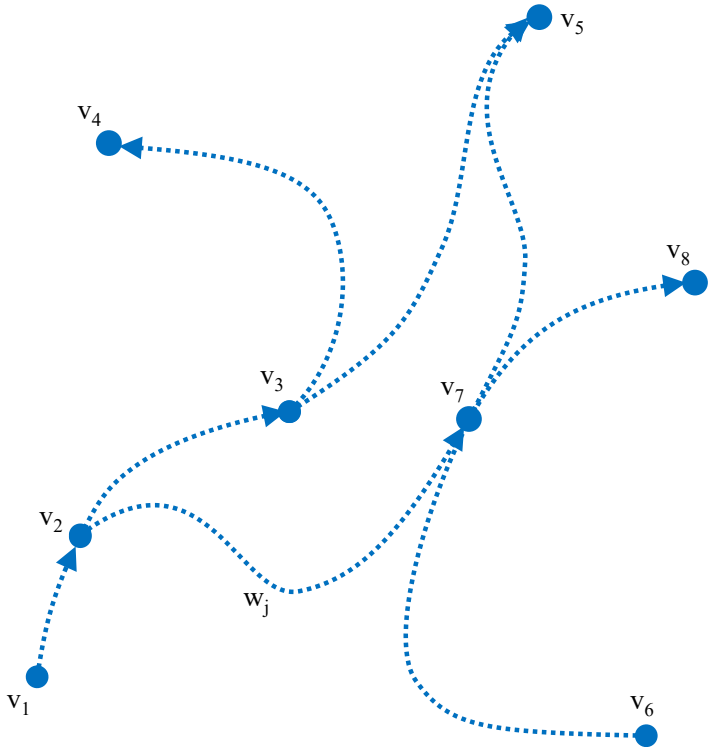


Fig. 9. Example of the waypoint level of a road network. Each small dot w_j is a single waypoint.

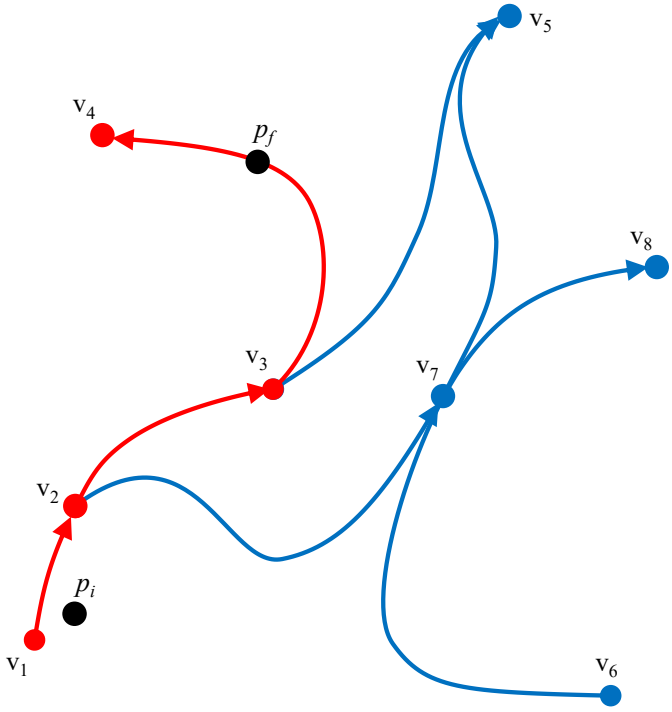


Fig. 10. Example of the route planning process using the lane-level road network graph. The red lines represent the graph edges that make the optimal route from point p_i to p_f.

In IARA's route and path planning processes, the initial and final poses are always predefined for every mission. Fig. 10 illustrates an example of route planning, given an initial pose p_i and a final pose p_f. The algorithm identifies that the nearest edge to the initial pose is the one connecting vertex v_1 to v_2, thus selecting v_1 as the starting vertex. Similarly, it determines that the nearest edge to the final pose is the one connecting vertex v_3 to v_4, choosing v_4 as the ending vertex. The A* search algorithm is then executed, resulting in the optimal lane-level route $R_{1,4} = \{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \mathrm{v}_4\}$ shown in red.

Once route planning is completed, path planning is carried out in three distinct steps, each executed by a separate module.

The first step of path planning is performed by the Route Planner module using the waypoint-level road network graph. Fig. 11 illustrates an example of this process. The Route Planner module retrieves the sequences of waypoints corresponding to each edge in the optimal route. The algorithm then identifies that the nearest waypoint to the initial pose is w_i, which belongs to the sequence of waypoints corresponding to

the initial edge. Similarly, it determines that the nearest waypoint to the final pose is w_f, which belongs to the sequence of waypoints corresponding to the final edge.

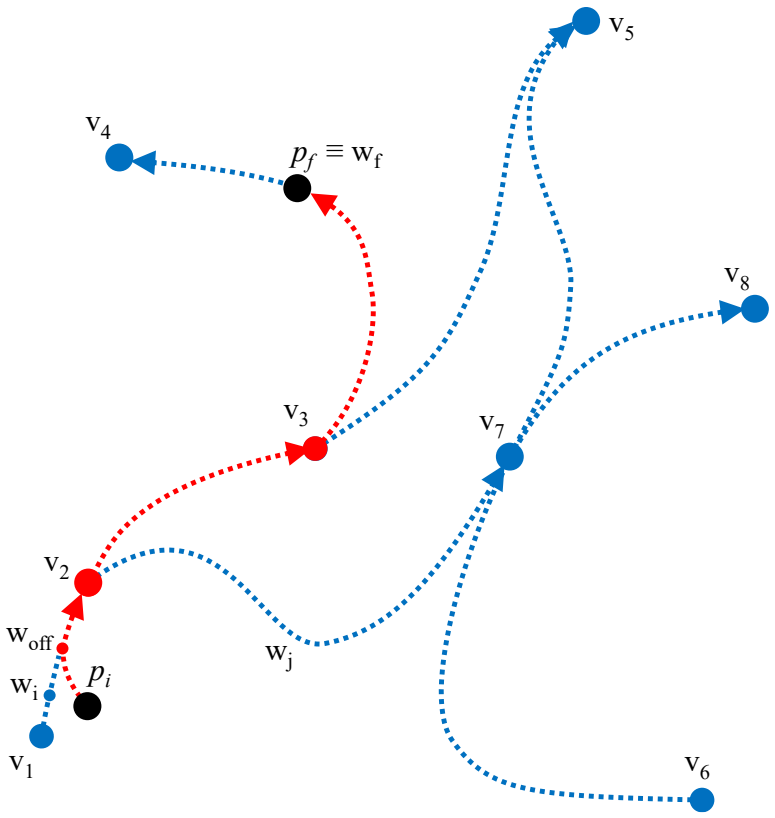


Fig. 11. Example of the path planning process using the waypoint-level rode network graph. The red dots represent the optimal path from point p_i to p_f.

The second step of path planning is performed by the Off-Road Planner module, as shown in Fig. 11. The Route Planner module identifies that the distance between waypoint w_i and initial pose p_i exceeds a predefined maximum limit. Consequently, it selects an alternative waypoint from the main path, w_{off}, based on the coordinates and orientation angle θ of p_i and the vehicle's kinematics. The Off-Road Planner module is then invoked to compute a connecting path from p_i to w_{off}. This path is designed to merge smoothly with the main route, using available grid maps and advanced algorithms [57]. A similar approach is applied to the waypoint w_f and the final pose p_f. In this example, w_f and p_f coincide, so no additional connecting path is needed. The connecting path is then combined with the main path, generating a complete path plan from the origin p_i to the destination p_f, shown in red.

The Router Planner module receives the complete path plan from p_i to p_f and stores it in memory for reference throughout the travel. As the autonomous vehicle follows this path, the Route Planner module continuously publishes a cropped sequence of 150 waypoints ahead of the current pose and 50 waypoints behind (if available). The Behavior Selector module subscribes to this current path plan and uses it for the third step of path planning and the decision-making process.

Fig. 12 and Fig. 13 illustrate examples of the third step of path planning, performed by the Behavior Selector module, using obstacle distance grid maps and the Frenet Frames Path Planner. They display images generated by the Navigator Graphical User Interface module, captured 10 seconds apart. These images feature an occupancy grid map as background, where each white cell represents obstacle-free ground, and dark cells indicate obstacles with a minimum height above or depth below the ground. The blue rectangle represents the vehicle's current pose, as published by the Localizer module, while the yellow rectangle indicates the goal pose, as published by the Behavior Selector module. The blue long line represents the current path, as published by the Route Planner module, and the orange long line shows the available nearby offline routes.

The numbered elements (e.g., 14, 15, 16, 17) represent dynamic obstacles that were not present in the offline maps but are now detected near the autonomous vehicle. To navigate around these dynamic obstacles, the Frenet Frames Path Planner is invoked. This planner computes a set of alternative paths ahead of the vehicle's current pose, both to the left and to the right, using the main route as a reference.

The red and purple lines in Fig. 12 represent the Frenet Frames alternative paths that put a high risk of collision with the dynamic obstacles numbered 15 and 16. In contrast, the two green lines on the far-right side of the Frenet Frames indicate alternative paths with negligible collision risk. The Behavior Selector module chooses the green alternative path that is closest to the main route. This constitutes the third step of path planning.

Fig. 13, captured 10 seconds later, shows that the autonomous vehicle (represented by the blue rectangle) had successfully avoided the dynamic obstacles numbered 15 and 16. Subsequently, the Frenet Frames Path Planner computed a new set of alternative paths ahead of the vehicle's current pose, aiming to merge back with the main route. As before, the red and purple lines represent the alternative paths with high risk of collision with the dynamic obstacle numbered 17. The Behavior Selector module then selects the green alternative path that is closest to the main route at that moment.

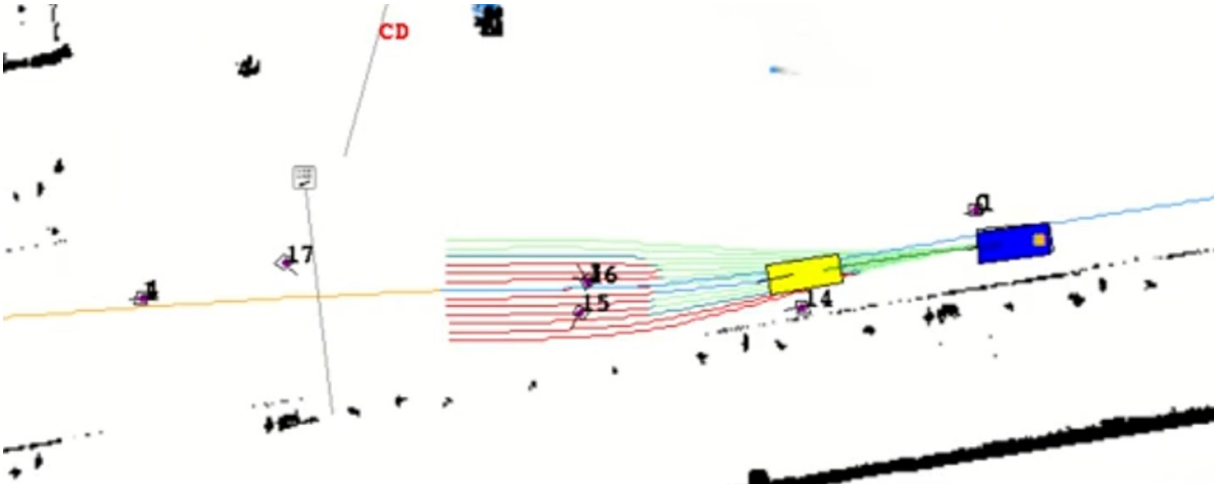


Fig. 12. Example of the path planning process using the Frenet Frames at time t_1.

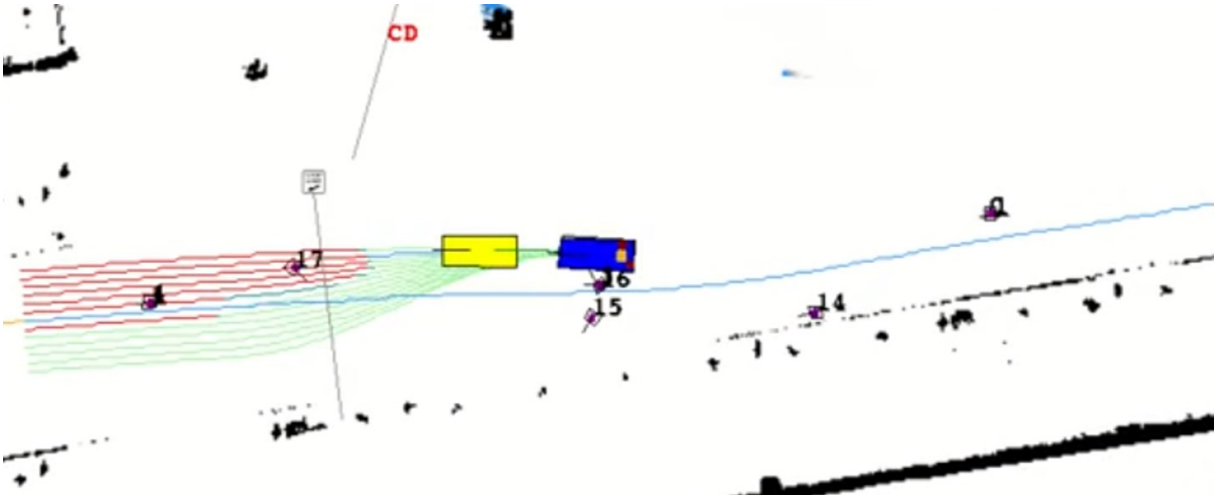


Fig. 13. Example of the path planning process using the Frenet Frames at time t_2.

As depicted in this section, IARA's route planning and path planning follow a top-down approach. It begins with the highest-level available information: lane-level offline routes of a road network previously generated using non-automated techniques. Next, it utilizes waypoint-level offline routes previously created. Finally, it employs obstacle distance grid maps to compute either off-road paths or sets of alternative paths for avoiding dynamic obstacles.

3.4. The Creation of Routes using Non-Automated Techniques

This section describes the process of building static routes from grid maps using non-automated techniques. These methods demand significant human effort, either for sensors data logging or graphical editing. This underscores the motivation for the present study, which aims to develop artificial intelligent techniques to reduce the effort required for low-level route creation.

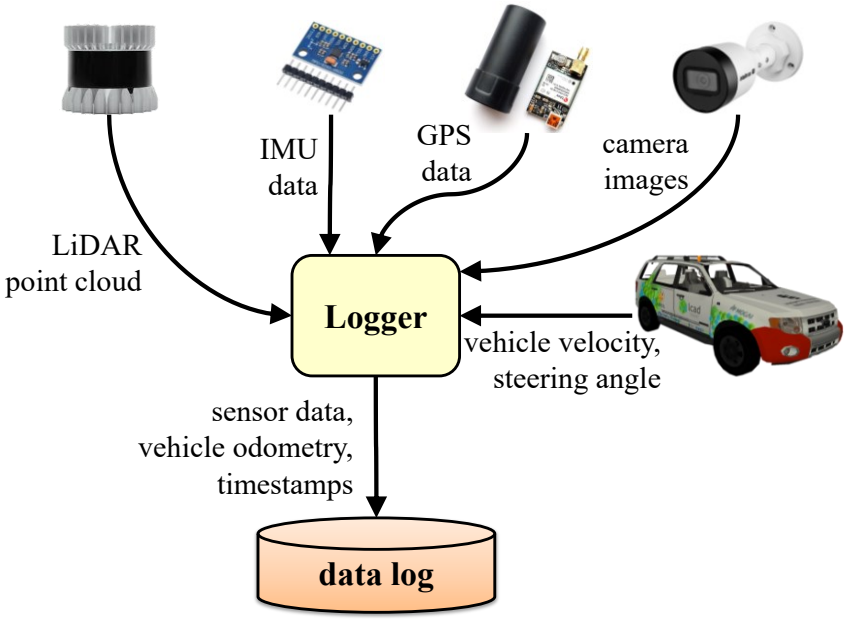


Fig. 14. Logging sensor data: LiDAR, IMU, GPS, camera, and vehicle odometry data.

Creating a route or path requires an existing grid map of the region the pathway traverses. For the IARA System, the standard procedure for generating grid maps begins with logging sensors and vehicle odometry data (Fig. 14). This involves running the sensor drives and the Logger module while manually driving the vehicle through

the entire area that needs to be mapped. Because the Controller module is not active during this procedure and the vehicle is manually operated, the logging process is considered an “offline” process.

Subsequent offline processes utilize the logged data, employing algorithms inspired by Graph SLAM [62] [63], to apply optimization techniques to the simultaneous localization and mapping (SLAM) problem. Fig. 15 depicts these processes.

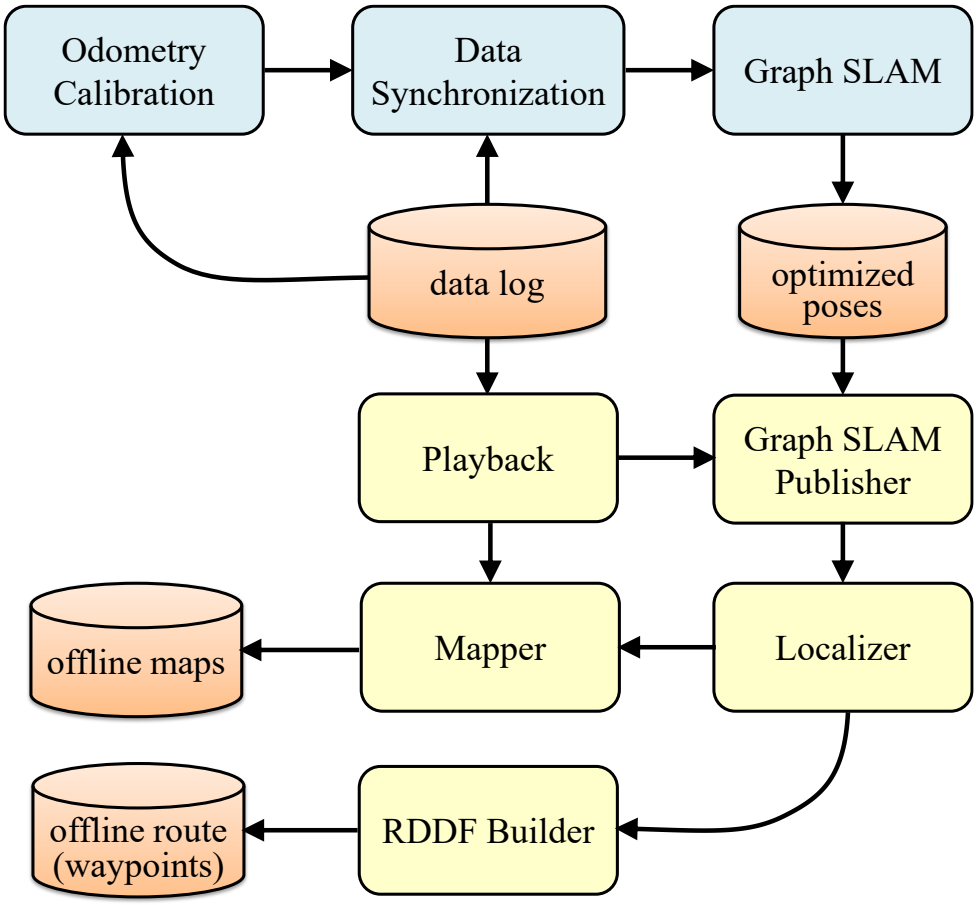


Fig. 15. IARA’s offline processes simplified block diagram.

Once the data log is available, the first pre-processing step for the Graph SLAM is the Odometry Calibration module. This module processes the entire data log to compute calibrated odometry poses, along with the multiplier and bias factors for vehicle velocity and steering angle measurements.

The Data Synchronization module processes the entire data log, applying the calibrated multiplier and bias factors to generate synchronized data that integrates all sensors readings.

The Graph SLAM module processes the synchronized data, applying the optimization algorithm [62] to produce a set of optimized poses that accurately represent the vehicle's trajectory during the logging process.

Once the optimized poses are available, the mapping process begins by running the Playback module, which reads the data log and publishes all the sensors and vehicle odometry data to the subscriber modules.

The Graph SLAM Publisher module reads the set of optimized poses generated by the Graph SLAM module and uses this information to provide corrected fused odometry to the Localizer module. This localization reference is then published to the Mapper module for generating the offline maps and to the RDDF Builder module for creating a single waypoint-level route.

The overall procedure described in this section can be executed multiple times across adjacent regions and pathways, allowing for the creation of multiple offline maps and routes.

The Waypoints Editor [58] provides a graphical tool for manually editing the shape and coordinates of one or multiple waypoint-level routes. This process is particularly important for ensuring that two routes intersect correctly.

Fig. 16 shows two images of the Waypoints Editor. On the left side, three lines in distinct colors – magenta, orange, and green – represent different waypoint-level routes. Each gray dot on the magenta line corresponds to an individual waypoint that can be manually edited. In this instance, the magenta and the green lines do not intersect correctly, which would cause the route planning process to fail in creating a composed route connecting them. On the right side, the Waypoints Editor displays the magenta route after the editing process, where it now correctly intersects with the

green route. Following this adjustment, it becomes possible to create a coherent road network.

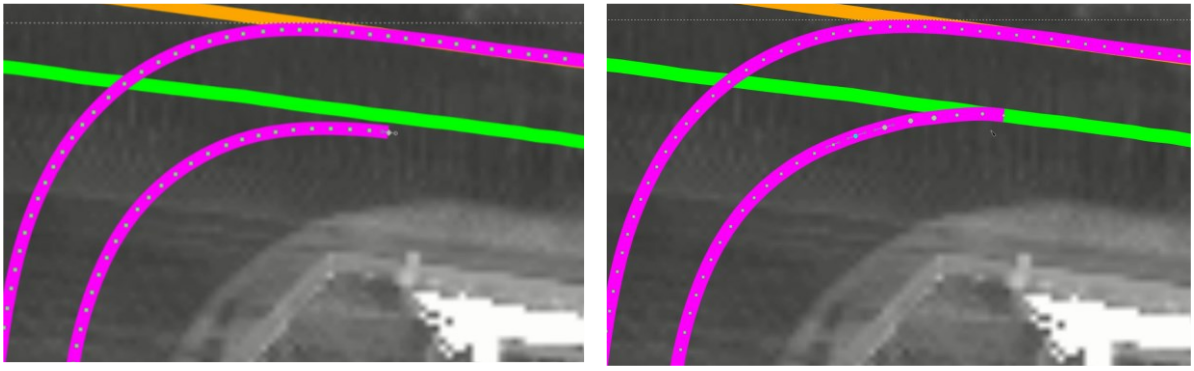


Fig. 16. The Waypoints Editor is used to adjust route intersections.

The Road Network Generator module processes the entire set of waypoint-level routes available for a region and creates a directed graph, as shown in Fig. 11. In this road network graph, each waypoint functions as a vertex, and each edge connecting two vertices considers the presence of merging and forking intersections, where applicable. Once the waypoint-level graph is created, the Road Network Generator module further processes it to produce the lane-level graph, as depicted in Fig. 10.

As outlined in this section, the creation of routes and paths for the IARA System follows a bottom-up approach. It starts with the waypoint-level routes, which are generated either though pose estimation based on logged sensor data or by manually editing graphical tools – both of which are time-intensive processes. Once the low-level graph is available, it is then used to compute a higher-level graph.

This process involves several offline stages, beginning with the logging of sensor and vehicle odometry data and progressing through various stages, requiring substantial human intervention, particularly in the manual adjustment of waypoints and the editing of route intersections.

The challenges inherent in these traditional methods underscore the importance of developing more advanced, automated approaches to route creation. The significant time and effort required for generating and refining low-level and high-level graphs

reveal a critical need for innovative solutions that can streamline these processes. By leveraging artificial intelligence (AI) techniques, it is possible to reduce the time and effort required to create and optimize routes.

The present study aims to address these challenges by exploring and developing AI-based techniques that can automate the low-level route creation process. While the current non-automated methods for creating static routes from grid maps are effective, the development of AI techniques to automate these processes holds significant potential to enhance the efficiency and accuracy of route creation, thereby improving the overall performance of autonomous systems like IARA. The shift towards AI-driven methods represents a crucial step forward in the evolution of autonomous navigation, envisioning more reliable route generation in dynamic and complex environments.

4. PROPOSED SEMANTIC SEGMENTATION OF ROAD GRID MAPS FROM REMISSION GRID MAPS

In this work, we represent roads using what we call *road grid maps*. The road grid maps we propose are square matrices, $C = \{c_{0,0}, \dots, c_{i,j}, \dots, c_{s-1,s-1}\}$, where: $s \times s$ is the size of C in cells; each element $c_{i,j}$ represents a small square region of real-world space; and the value of each $c_{i,j}$ is a code associated with the semantics of the road grid map.

In the IARA System, remission grid maps are square matrices, $R = \{r_{0,0}, \dots, r_{i,j}, \dots, r_{s-1,s-1}\}$, where each cell $r_{i,j}$ represents a small square region of real-world space with dimensions 0.2 m × 0.2 m. The value of each $r_{i,j}$ corresponds to the average LiDAR remission (the intensity of the returned laser pulse) for that region as observed during mapping. If a grid map cell was never scanned by any LiDAR beam, its corresponding value is set to –1, and this cell is depicted in blue in all figures of remission grid maps.

The dimensions of the road grid maps are identical to those of IARA's remission grid maps, with each map having a size of s =1,050. The physical area of real-world space represented by each cell $c_{i,j}$ in the grid corresponds to 0.2 m × 0.2 m. Therefore, any single road grid map or remission grid map covers a total area of 210 m × 210 m.

The coding pattern proposed for the road grid map cells was defined to differentiate between cells representing potential pathways (lanes) and those that fall outside drivable areas (off lane). Additionally, it distinguishes cells located near the center of a lane from those near the edges. Another important characteristic encoded in the map is the type of lane border: whether it is a broken line, permitting vehicle overtaking, or a solid line, which restricts it.

We defined and tested two distinct sets of coding patterns.

4.1. Coding Pattern #1

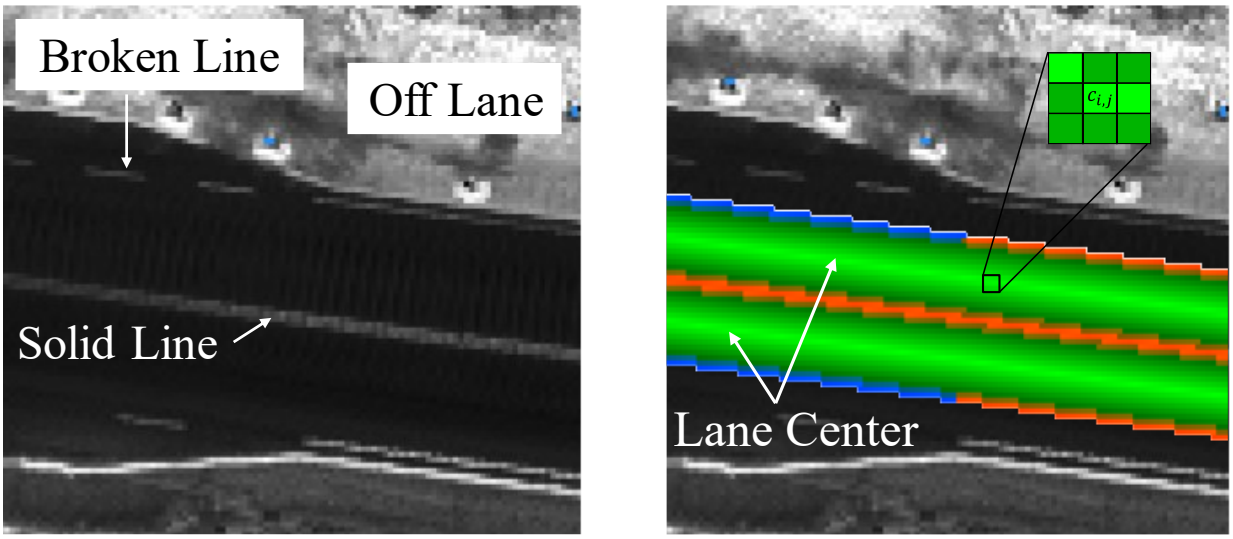
Set #1 consists of 17 classes, with the possible values of $c_{i,j}$ defined in TABLE 1. Refer to Fig. 17 for a visual representation of the coding pattern.

TABLE 1. CODING PATTERN #1 FOR ROAD GRID MAP CELLS

Class	Road Feature	Distance from Lane Center
0	Off Lane	N/A
1	Solid Line Marking	N/A
2	Broken Line Marking	N/A
3	Partial Solid Line Marking	N/A
4	Partial Broken Line Marking	N/A
5	Within Lane	0 (Center)
6	Within Lane	1/22 of lane width
7	Within Lane	2/22 of lane width
8	Within Lane	3/22 of lane width
9	Within Lane	4/22 of lane width
10	Within Lane	5/22 of lane width
11	Within Lane	6/22 of lane width
12	Within Lane	7/22 of lane width
13	Within Lane	8/22 of lane width
14	Within Lane	9/22 of lane width
15	Within Lane	10/22 of lane width
16	Within Lane	11/22 or 1/2 lane width

In this coding pattern, we assumed a fixed lane width of 3.2 m.

In summary, the table classifies each map cell based on its position relative to the lane, indicating whether it is inside or outside the lane, its proximity to the lane center, and its alignment with road markings. This classification aids in interpretation of the map in relation to the road infrastructure, providing valuable use for autonomous driving and road analysis tasks.



(a) (b)

Fig. 17. (a) A cropped section of a remission grid map. (b) Same cropped section overlaid with the ground truth road grid map. The zoomed-in view highlights nine map cells $c_{i,j}$ arranged in a 3 × 3 grid.

The value of each cell, $c_{i,j}$, in the road grid map ground truth corresponds to different feature classes: $c_{i,j}$ = 0 represents an Off Lane area, indicating a non-drivable region, as shown in Fig. 17 (a); $c_{i,j}$ = 1 indicates a Solid Line marking near the lane boundary, as depicted in (a) and color-coded in red in (b); $c_{i,j}$ = 2 denotes a Broken Line marking near the lane boundary, also shown in (a) and color-coded in blue in (b); values of $c_{i,j}$ ranging from 5 to 16 represent varying distances from the map cell to the center of the lane, with different shades of green used for color coding in (b).

4.2. Coding Pattern #2

Set #2 consists of 6 classes, with the possible values of $c_{i,j}$ defined in TABLE 2.

Coding pattern #1 is more granular, providing finer detail in classifying a map cell's position within the lane, whereas pattern #2 simplifies this by consolidating multiple positions into broader categories. Pattern #1 is more complex and detailed, which

makes it suitable for applications requiring high precision, while pattern #2 is simpler and might be more practical for applications where such granularity is unnecessary.

Both sets encode the most relevant road features. The purpose of testing these two distinct approaches is to assess their impact on the performance of training and the accuracy of the selected DNNs.

TABLE 2. CODING PATTERN #2 FOR ROAD GRID MAP CELLS

Class	Road Feature	Distance from Lane Center
0	Off Lane	N/A
1	Solid Line Marking	N/A
2	Broken Line Marking	N/A
3	Within Lane	Up to 1/22 of lane width
4	Within Lane	From 2/22 to 6/22 of lane width
5	Within Lane	From 7/22 to 11/22 or 1/2 lane width

As with Set #1, in this coding pattern we assume a fixed lane width of 3.2 m.

5. DNNs FOR SEMANTIC SEGMENTATION OF ROAD GRID MAPS

We selected various DNNs to evaluate the feasibility of generating semantic segmentation of road grid maps based on remission grid maps. The selection criteria included:

(i) Availability for multi-purpose tasks: Once a DNN has demonstrated success across a variety of segmentation use cases, it is more likely to effectively handle new tasks, including the one proposed in this work.

(ii) Open licensing: DNNs with free licensing promote collaboration within the scientific community, encouraging further research, advancing applications, and facilitating meaningful comparisons.

(iii) Low resource requirements for training and testing the DNNs: Given that the primary goal of this work is to develop embedded solutions for autonomous vehicles, minimizing resource consumption is a critical consideration.

5.1. U-Net – U-shaped Encoder-Decoder

The encoder-decoder architecture was first introduced in [64] for Recurrent Neural Networks (RNNs) and has since become a foundational approach in a wide range of deep learning tasks, particularly in natural language processing, image processing, and semantic segmentation.

U-Net [65] is an encoder-decoder convolutional neural network (CNN) architecture specifically developed for biomedical image segmentation. Its design allows for effective performance even with limited training data, making it highly valuable in the medical imaging field, where annotated datasets are often scarce. U-Net's versatility, however, extends beyond biomedical applications; it has proven effective in other domains that demand precise segmentation, such as satellite imagery [66], agricultural

analysis [67], and autonomous driving [68]. This widespread utility has made U-Net one of the most popular and influential architectures in the field of image segmentation.

The U-Net architecture is distinguished by its U-shaped encoder-decoder structure, which consists of a contracting path and an expansive path. This design enables U-Net to effectively capture both high-level contextual information and fine-grained details, which are essential for accurate segmentation tasks. In Fig. 18, the U-Net architecture is illustrated, where each blue box represents a multi-channel feature map. The number of channels is indicated above each box, while the spatial dimensions are noted below. White boxes depict copied feature maps, and the arrows illustrate the various operations performed through the network.

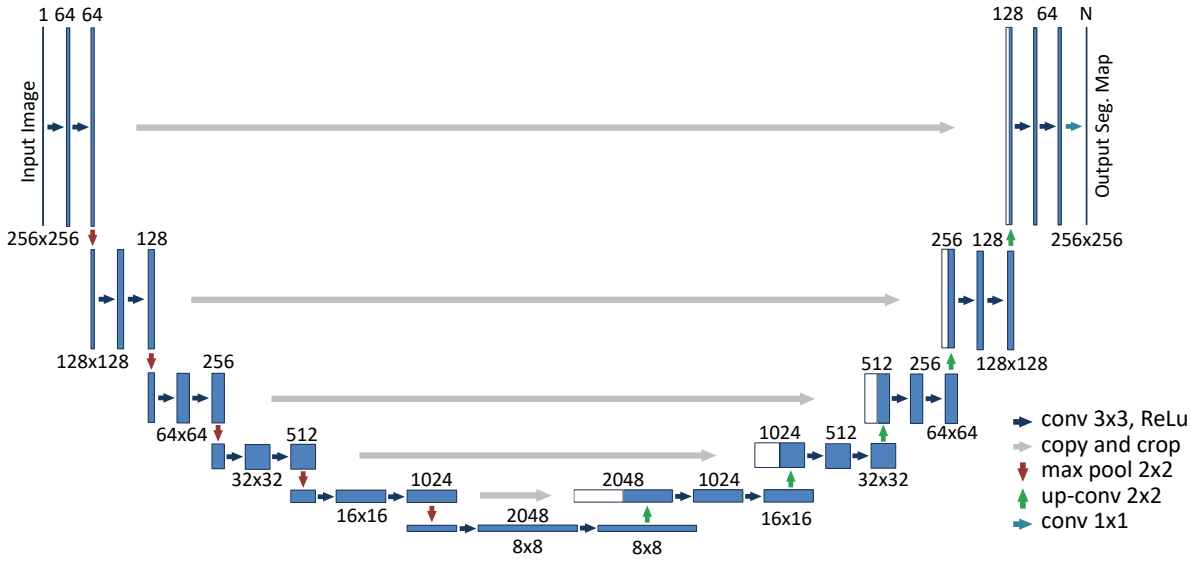


Fig. 18. The U-Net U-shaped encoder-decoder architecture.

The contracting path, or the encoder, is responsible for capturing the context of the input image. It consists of repeated applications of two 3x3 convolutional layers, each followed by a rectified linear unit (ReLU) activation function and a 2x2 max pooling operation with a stride of 2. This downsampling process reduces the spatial dimensions of the feature maps while increasing the number of feature channels. The reduction in spatial resolution allows the network to focus on larger regions of the image, capturing more contextual information, which is crucial for segmentation tasks.

As the network goes deeper, it captures more abstract features that are essential for distinguishing between different structures in an image.

Between the contracting and expansive paths, there is a bottleneck that represents the lowest resolution of the feature maps. This bottleneck consists of two convolutional layers with ReLU activation, followed by a max pooling layer. The bottleneck serves as the bridge between the encoder and decoder, capturing the most abstract and high-level features of the image.

The expansive path, or the decoder, is symmetric to the contracting path. It is responsible for upsampling the feature maps back to the original image resolution, allowing for precise localization of the segmented regions. The upsampling is achieved through a 2x2 transposed convolution (also known as up-convolution or deconvolution) that halves the number of feature channels. Each upsampling step is followed by the concatenation of the corresponding feature map from the contracting path. This concatenation process is critical because it provides the decoder with high-resolution features that were previously captured during downsampling. After concatenation, two 3x3 convolutions with ReLU activation are applied, refining the features and enhancing the segmentation accuracy.

U-Net employs skip connections between the encoder and decoder at each corresponding level. These skip connections transfer the fine-grained information from the encoder directly to the decoder, allowing the network to retain spatial details that might otherwise be lost during downsampling. This strategy ensures that the network can generate detailed and accurate segmentation maps, even for complex and small structures.

The final layer of U-Net is a 1x1 convolution that maps the 64-component feature vector at each pixel to the desired number of output classes (N). U-Net typically uses a softmax activation function in its output layer, when applied to multi-class image segmentation tasks. The result is a pixel-wise classification map, where each pixel is assigned a class label, effectively segmenting the image.

U-Net was designed to be trained with limited annotated data, which is common in biomedical imaging. To achieve high performance despite the small dataset size, the authors of U-Net introduced several key training strategies:

- Data augmentation is critical in the training process to artificially expand the training dataset and teach the network invariances. U-Net utilizes various augmentation techniques such as elastic deformations, rotations, shifts, and intensity variations. Elastic deformations are effective in biomedical imaging because they simulate realistic variations in tissue structures, making the network robust to such changes. This approach enables U-Net to generalize well even when trained on a small number of annotated images.

- U-Net uses a pixel-wise softmax combined with a cross-entropy loss function to evaluate the accuracy of the segmentation. The softmax function assigns a probability to each class at every pixel, while the cross-entropy loss penalizes the network for incorrect predictions. Additionally, the authors introduced a weighted loss function to address the challenge of segmenting touching objects of the same class. This is achieved by assigning higher weights to the background labels between touching objects, encouraging the network to learn the separation boundaries more effectively.

- Proper initialization of the network weights is essential to ensure stable and effective training. The weights in U-Net are initialized using a Gaussian distribution with a standard deviation that is scaled according to the number of incoming connections in each layer. This initialization helps prevent issues like vanishing or exploding gradients. The network is then trained using stochastic gradient descent (SGD) with a high momentum, which allows the network to make use of past gradients to accelerate convergence.

5.2. ENet – Efficient Neural Network

ENet (Efficient Neural Network) [69] is a deep learning model designed for real-time semantic segmentation, specifically tailored for applications requiring high efficiency and speed, such as autonomous driving, robotics, and augmented reality. ENet addresses the challenge of deploying neural networks in environments where computational resources are limited, and real-time performance is critical.

ENet is built on the concept of efficiency, balancing model complexity with performance. ENet has an encoder-decoder DNN architecture [64] [70] inspired by the well-known ResNet [71] [72] model but introducing several modifications to optimize for speed and memory usage. Unlike the typical U-Net [65], ENet uses an asymmetric encoder-decoder structure where the encoder is designed to be lightweight and the decoder even more so, relying heavily on the encoder's ability to retain critical spatial information (Fig. 19).

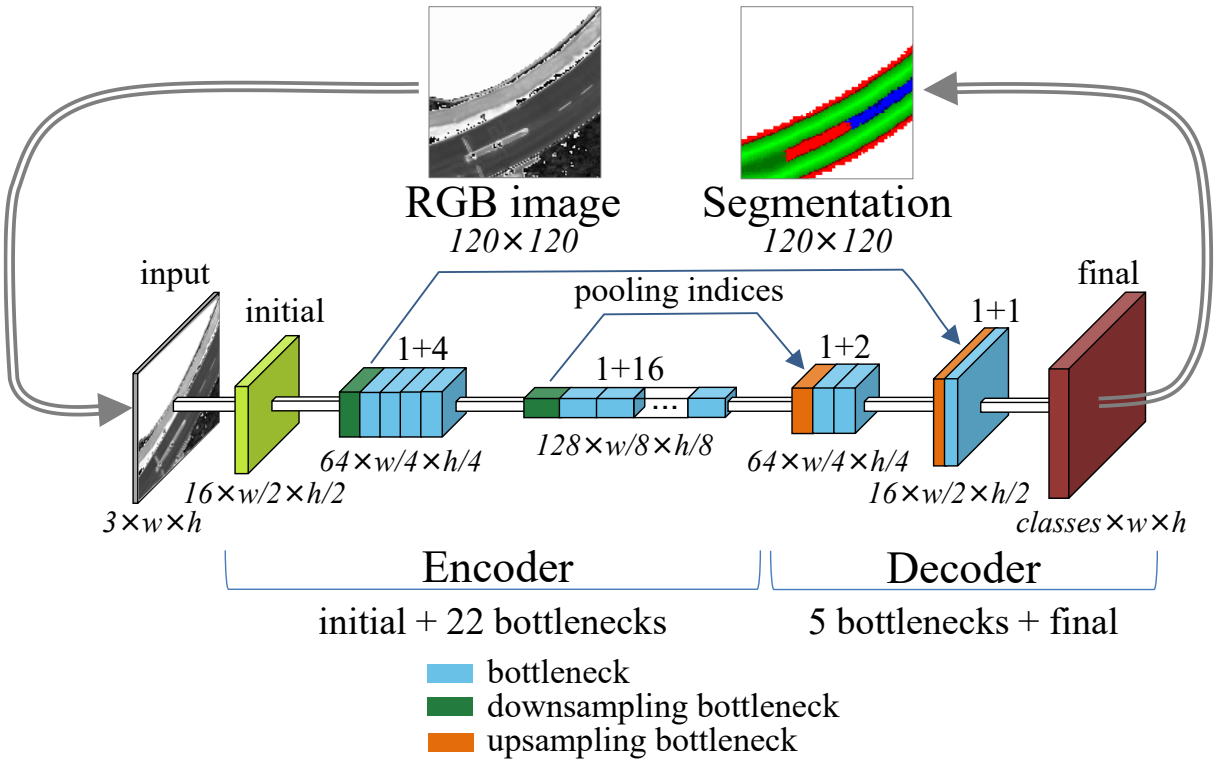


Fig. 19. The ENet encoder-decoder asymmetric architecture.

The encoder of ENet is designed to capture spatial features from the input image. It consists of multiple stages of convolutional layers, each followed by batch normalization and parametric ReLU (PReLU) activation functions. The initial layers downsample the image, reducing its spatial resolution and increasing the number of feature channels. This downsampling is critical to maintaining efficiency, as it reduces the computational load for subsequent layers. ENet uses dilated convolutions in later stages of the encoder to maintain a larger receptive field without increasing the computational burden.

The decoder in ENet is significantly smaller than the encoder and is responsible for upsampling the feature maps back to the original image resolution. The upsampling is performed using learned deconvolution (also known as transposed convolution) layers. However, unlike typical segmentation networks, ENet's decoder is minimalistic, relying on the encoded features to carry the bulk of the segmentation information. The idea is to achieve sufficient segmentation accuracy while keeping the network lightweight and fast.

To mitigate the loss of spatial information due to downsampling in the encoder, ENet incorporates skip connections between the encoder and the decoder. These connections allow feature maps from earlier layers of the encoder to be directly fed into corresponding layers of the decoder, helping to preserve fine-grained details in the segmentation output.

ENet is similar to SegNet [73] [74], but with a shorter decoder and several other modifications to allow faster operation without significant degradation of segmentation performance. ENet extensively uses bottleneck modules, which consist of a sequence of convolutions designed to reduce the number of parameters and operations. A typical bottleneck module includes a 1x1 convolution to reduce the number of channels, followed by a 3x3 convolution, and then another 1x1 convolution to restore the channel dimension. The use of these modules significantly reduces the model's complexity and enhances its efficiency.

ENet is trained in two stages. In the first stage, only the encoder is trained (receiving as input, in our case, a crop of a remission grid map, and as target the associated crop of the road grid map ground truth). Since the encoder reduces dimensionality by a factor of 8, it is necessary to perform an upsample of its output to compare to the ground truth. So, a temporary single deconvolution layer is connected at the output allowing the calculation of a softmax classifiers matrix with the size of the target ground truth (120 × 120 cells in our case). In the second stage, this temporary layer is removed and the decoder is connected at the output of the encoder (the output of the decoder is also a matrix of softmax classifiers with the size of the target ground truth). Then, the full network is trained.

During the test phase, the ENet receives as input a crop of the remission grid map and outputs the inferred road grid map in the form of a matrix of softmax classifiers: the class most active of each softmax classifier will be the value inferred for the corresponding $c_{i,j}$ of the road grid map.

One of the primary applications of ENet is in autonomous driving, where the ability to perform real-time segmentation of road scenes is crucial. ENet's efficient design allows it to process high-resolution images at high frame rates, enabling vehicles to quickly interpret and react to their surroundings. Additionally, ENet has been used in robotics for tasks like obstacle detection and navigation, where real-time environmental understanding is necessary. ENet’s lightweight architecture allows it to run on devices with limited processing power, such as mobile devices and embedded systems. ENet has been shown to achieve reasonable accuracy on datasets like Cityscapes, which is widely used for evaluating segmentation models in urban driving environments.

While ENet offers a good balance between efficiency and accuracy, it is not without limitations. The aggressive downsampling in the encoder can lead to a loss of fine-grained details, which might result in lower accuracy compared to more complex models, especially in scenarios requiring precise segmentation boundaries. Additionally, ENet's performance may degrade when dealing with highly complex scenes or tasks that require very high-resolution outputs.

5.3. Swin-Unet – Vision Transformer using Shifted Windows

Transformer [75] is a type of DNN architecture that has revolutionized the field of artificial intelligence, particularly in natural language processing (NLP) and computer vision. The core idea of a Transformer is the self-attention mechanism, which allows the model to focus on different parts of the input sequence when making predictions. For example, in language processing, the word "it" might refer to a previously mentioned noun, and self-attention helps the model identify these relationships by assigning higher importance to relevant words. In computer vision tasks, transformers can capture complex relationships between different parts of an image. Vision Transformers (ViTs) [76] and other variants have shown impressive results in image classification, object detection, and segmentation.

Transformers consist of multiple layers of self-attention and feedforward neural networks. The layers are stacked together, and the output of one layer becomes the input to the next. The standard Transformer architecture has an encoder and a decoder, each consisting of several layers. Unlike Recurrent Neural Networks (RNNs), which process sequences one step at a time, Transformers can process entire sequences in parallel. This parallelization makes them much faster and more efficient, especially when dealing with long sequences.

An efficient hierarchical Vision Transformer, called Swin Transformer [77], was proposed as a vision backbone. The name "Swin" comes from “Shifted Windows”, which is a key innovation introduced in this model. Unlike traditional ViTs, which apply self-attention globally across the entire image, the Swin Transformer applies self-attention within local, non-overlapping windows. These windows are then shifted in subsequent layers to allow cross-window connections. This approach helps to reduce computational complexity while maintaining the ability to model long-range dependencies (Fig. 20).

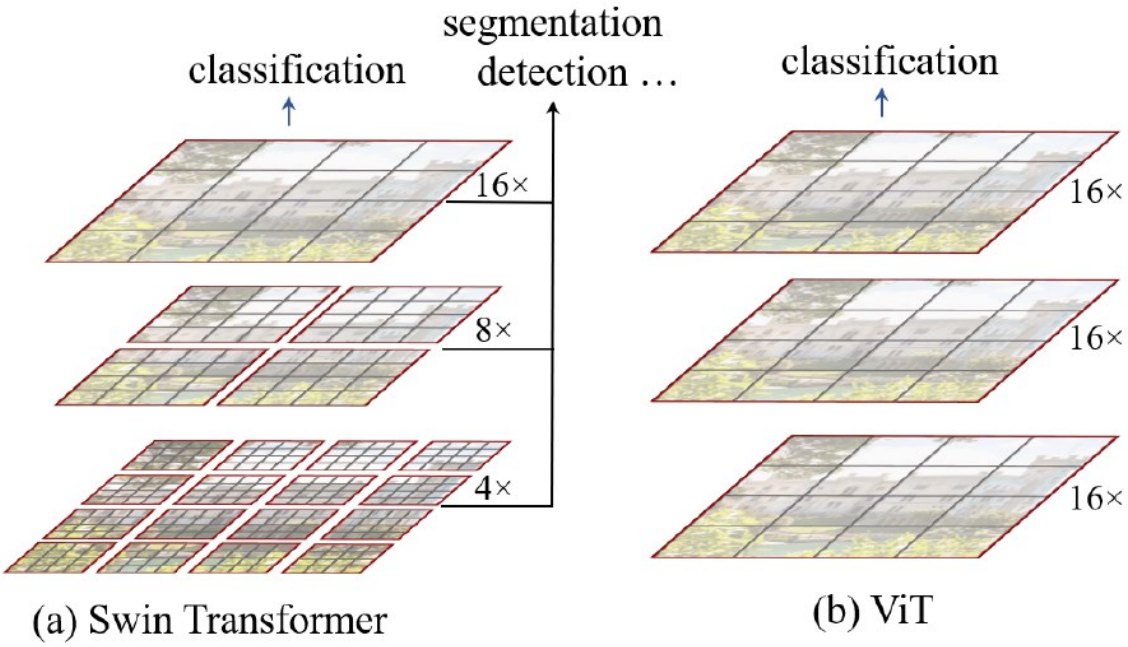


Fig. 20. Swin Transformer hierarchical approach in contrast to traditional Vision Transformers.

Swin Transformers introduce a hierarchical architecture that processes images at multiple scales, similar to how Convolutional Neural Networks (CNNs) work. The image is progressively downsampled, and feature maps of increasing size are produced at each stage. This hierarchical approach allows the model to capture both fine-grained details and global context.

Fig. 20 (a) shows the Swin Transformer building hierarchical feature maps by merging image patches in deeper layers. It has linear computation complexity to input image size due to computation of self-attention only within each local window. Fig. 20 (b) shows in contrast, previous ViTs building feature maps of a single low resolution. It has quadratic computation complexity to input image size due to computation of self-attention globally.

The Swin Transformer has been shown to perform well across various vision tasks, including image classification, object detection, and semantic segmentation. Its design allows it to be used as a general-purpose backbone for many vision-related applications.

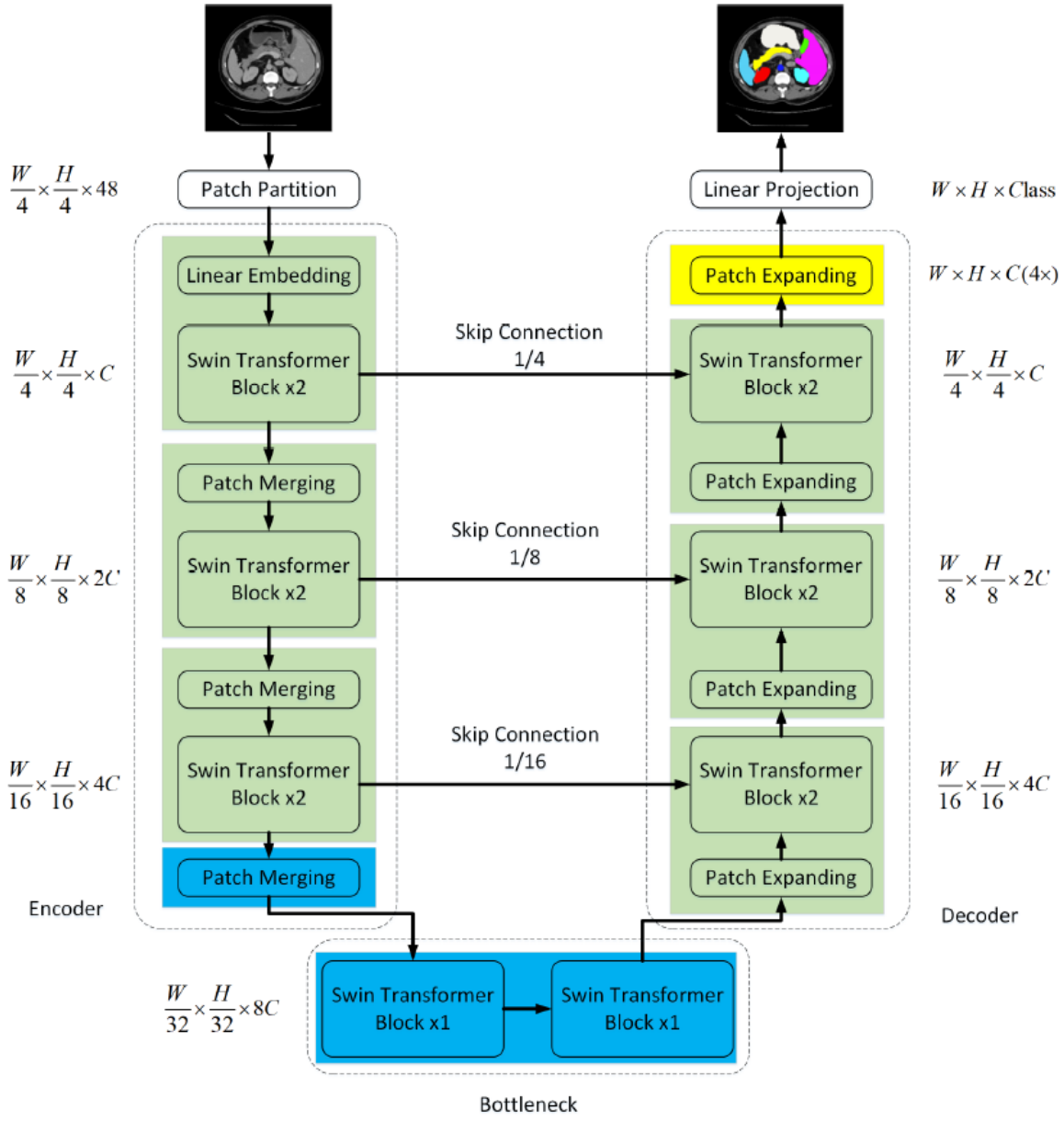


Fig. 21. The Swin-Unet architecture.

The Swin-Unet model [78] combines the power of Swin Transformers with the traditional U-Net architecture, which is typically based on convolutional operations. In recent years, convolutional neural networks (CNNs) with U-shaped architectures, such as U-Net [65], have been widely used. However, the inherent limitation of CNNs in capturing global and long-range semantic information due to the locality of convolution operations has driven the exploration of alternative approaches that can overcome these challenges.

The Swin-Unet follows the classic encoder-decoder structure of U-Net but with significant differences. Fig. 21 shows the architecture of Swin-Unet, which is composed of encoder, bottleneck, decoder and skip connections. Encoder, bottleneck, and decoder are all constructed based on Swin Transformer block.

The input image is divided into small patches, which are then treated as tokens and fed into an encoder based on Swin Transformers. These encoders are responsible for extracting contextual features from the image, which are then upsampled by a symmetric decoder to restore spatial resolution. The multi-scale features extracted by the encoder are fused with the features recovered in the decoder through skip connections, which are essential for maintaining spatial accuracy.

In the encoder, Swin-Unet divides the images into non-overlapping 4x4 patches. These patches are then processed by several Swin Transformer blocks and patch merging layers, which reduce the resolution and increase the dimensionality of the features. The bottleneck of the model consists of only two Swin Transformer blocks, reflecting the controlled depth of the model to avoid convergence issues. In the decoder, patch expanding layers perform the upsampling of the features, restoring the image's original resolution.

5.4. ST-UNet – Swin Transformer Embedding UNet

Traditional CNN-based methods, while effective, struggle with capturing global context due to the locality of convolution operations. ST-UNet [79] addresses this issue by embedding the Swin Transformer [77], which is known for its global modeling capabilities, into the U-Net [65] architecture. The ST-UNet is a hybrid model that combines the strengths of CNNs and transformers by using a dual encoder structure. The CNN-based encoder captures local features, while the Swin Transformer-based encoder models global dependencies.

The architecture includes:

- Spatial Interaction Module (SIM) to improve global feature correlation, focusing on pixel-level correlations, mitigating the impact of occlusions, and improving the segmentation of objects that are partially hidden.
- Feature Compression Module (FCM) to enhance small-scale feature representation, reducing information loss during downsampling process by using dilated convolutions and soft-pooling techniques, particularly for small-scale objects, thereby improving segmentation accuracy.
- Relational Aggregation Module (RAM) to combine global and local features effectively, integrating global dependencies captured by the Swin Transformer into the CNN features, enhancing the model's ability to differentiate between similar objects.

The Swin Transformer block is a key component of ST-UNet, offering a window-based self-attention mechanism that scales efficiently with image size. The block uses a shifted window approach to improve global context modeling while maintaining computational efficiency.

Fig. 22 illustrates the overall architecture of the ST-UNet, a hybrid model that combines the strengths of the Swin Transformer and the U-Net for semantic segmentation of images.

The left side of the architecture represents the main encoder, which is based on a CNN, specifically a ResNet-50 [80] with half-compressed channels. This encoder is responsible for extracting deep features from the input image. The features extracted by each layer of the CNN are denoted as A_n, where n indicates the stage or level of the encoder.

Parallel to the main encoder, the auxiliary encoder is composed of Swin Transformer blocks. These blocks are responsible for capturing global context information and long-range dependencies in the image. The output features of the Swin Transformer at each stage are denoted as S_n.

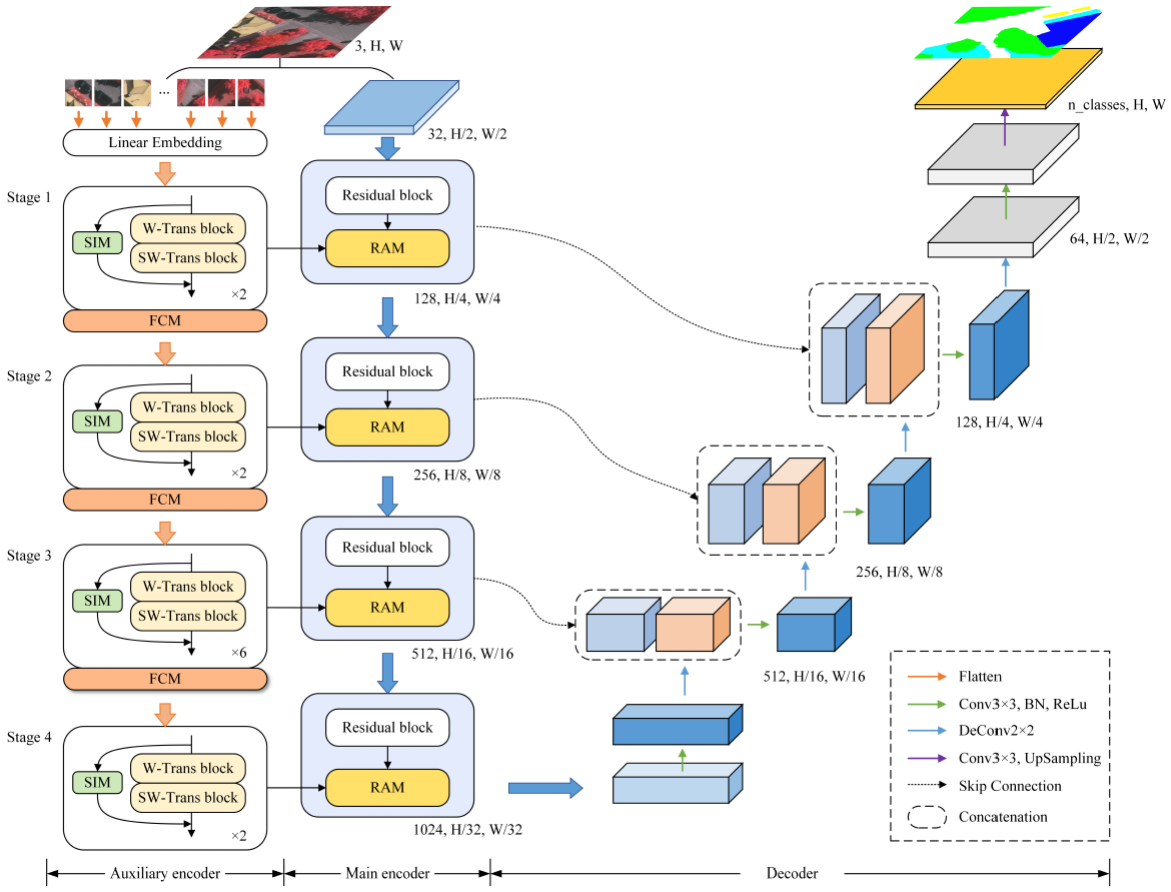


Fig. 22. The ST-UNet architecture.

Positioned within the auxiliary encoder, the SIM module is designed to enhance the information exchange across different spatial dimensions, compensating for the limitations of the window-based self-attention mechanism in the Swin Transformer. This module introduces pixel-level attention, which helps in resolving ambiguities caused by occlusions in the image. The SIM module works by focusing on the relationships between pixels rather than just patch tokens, making the transformer more effective for image segmentation tasks.

The FCM module is also integrated into the auxiliary encoder. Its role is to address the potential loss of fine details that can occur during the patch token downsampling process in the Swin Transformer. The FCM ensures that small-scale features are preserved, which is crucial for accurately segmenting small objects in the images. It uses a combination of dilated convolutions and soft-pooling to gather more detailed information and maintain structural integrity.

The RAM module serves as a bridge between the two encoders. It takes the output features from both the main encoder (A_n) and the auxiliary encoder (S_n) and integrates them to form more discriminative feature representations. The RAM does this by leveraging deformable convolutions and channel attention mechanisms, which help in refining the features based on the global context information provided by the Swin Transformer.

The decoder in ST-UNet follows the traditional U-Net structure, where it progressively upsamples the feature maps to reconstruct the image's spatial resolution. The decoder uses skip connections to concatenate the high-level semantic features from the encoder stages with the corresponding decoder stages. These skip connections are critical for preserving spatial information and improving segmentation accuracy, especially in the finer details of the image.

The flow of information in ST-UNet starts with an input image being fed into both the CNN-based main encoder and the Swin Transformer-based auxiliary encoder. For the auxiliary encoder, the input image is first divided into small patches. Each patch is a small section of the image, usually of a fixed size (e.g., 8x8 pixels). After these patches are extracted, the flatten operation converts each 2D patch into a 1D vector. For example, an 8x8 patch with 3 channels (RGB) would be flattened into a vector of size 192 (8x8x3). This flattened vector can then be fed into a linear embedding layer, where it is projected into a lower-dimensional space or prepared for further processing. In the case of the Swin Transformer, this flattened vector is used to create a sequence of "tokens" that the transformer can process.

The CNN extracts local features through its residual blocks, while the Swin Transformer captures global context features. At each stage, the RAM fuses the features from both encoders, enriching the local CNN features with global context from the Swin Transformer. This fused feature set is then passed to the next stage of the main encoder. The fused features are decoded through the U-Net's decoder structure, which progressively increases the resolution of the feature maps until the original image resolution is restored. The final segmentation map is generated as the output.

The final output of the decoder is a high-resolution segmentation map, where each pixel is assigned a semantic label. This output is obtained by applying a 3x3 convolutional layer followed by a linear interpolation upsampling, which ensures that the segmentation map matches the resolution of the input image.

6. APPLICATIONS OF ROAD GRID MAPS IN AUTONOMOUS VEHICLE NAVIGATION

The road grid map we have developed has numerous important applications in the field of autonomous vehicle. For instance, it can estimate the distance between the autonomous vehicle and the center of the nearest lane or indicate whether the vehicle is permitted to overtake another vehicle based on the lane markings (e.g., broken or solid lines). Additionally, it can be utilized to construct a detailed graph of the mapped road network, among other uses.

In this study, we propose the use of road grid maps to build low-level path plans, called RDDFs (Route Data Definition Files), which are essential for the autonomous operation of IARA. In the IARA System, RDDFs are structured in a simple text format, consisting of a sequence of waypoints spaced approximately 0.5 meters apart. Each waypoint is defined by coordinates in the UTM system and includes a recommended vehicle orientation (yaw angle). Additionally, waypoints may be enriched with road annotations, such as speed limits, speed bumps, crosswalks, and other relevant features, providing a detailed and precise representation of the road environment.

TABLE 3. EXAMPLE OF RDDF FORMAT USED IN THE IARA SYSTEM – WAYPOINTS FILE

X Coord (m)	Y Coord (m)	Yaw (rad)	Velocity (m/s)	Steering (rad)	Timestamp (s)
7757673.76	-363604.52	0.687	2.81	0.010953	1502316065.290736
7757674.19	-363604.22	0.689	3.08	0.013366	1502316065.490572
7757674.71	-363603.80	0.690	3.30	0.007510	1502316065.690336

TABLE 3 shows an example of the RDDF format used in the IARA System. The Waypoints file consists of a sequence of waypoints, each defined by three primary values: X and Y coordinates in meters, and the vehicle's orientation angle (yaw) in radians. The last three columns, which are optional, include information about the

logged velocity, steering angle, and timestamp. These are present only if the RDDF was generated by the RDDF Builder module, as described in Section 3.4.

In the IARA System, the X and Y coordinates are derived from the UTM (Universal Transverse Mercator) coordinate system, using the following conversion formulas:

$$IARA\_X = UTM\_Y \quad (1)$$

$$IARA\_Y = -UTM\_X \quad (2)$$

In TABLE 4, the Road Annotations file details the relevant features of the road environment, which are defined by: the type and subtype (code) of the annotation, the applicable orientation angle (yaw) in radians, the X and Y coordinates in meters, and an additional parameter required for specific types of annotations. The applicable yaw angle is particularly important because traffic signs often apply only to lanes in one direction, not to those in the opposite direction. In this example, the extra parameter for the pedestrian track annotation is 9.00, indicating that the system must monitor a circular area with a radius of 9.00 meters around those coordinates to detect the presence of pedestrians. For the traffic light annotation, the extra parameter is 3.32, indicating that the traffic light is located at those coordinates, 3.32 meters above the ground.

TABLE 4. EXAMPLE OF RDDF FORMAT USED IN THE IARA SYSTEM – ROAD ANNOTATIONS FILE

Description	Type	Code	Yaw (rad)	X Coord (m)	Y Coord (m)	Param
SPEED_LIMIT	7	8	-2.015	7753405.40	-364951.60	0.00
PEDESTRIAN_TRACK	3	0	0.103	7757355.40	-364102.40	9.00
TRAFFIC_LIGHT	8	0	0.451	7757077.60	-363696.00	3.32

6.1. Proposed Enhancements in the IARA System

With the adoption of the proposed road grid maps and the implementation of appropriate inference processes for generating low-level path plans (RDDFs), the IARA System can benefit from enhanced capabilities. Fig. 23 presents a simplified block diagram of the new proposed IARA's software architecture, building on the diagram previously shown in Fig. 6.

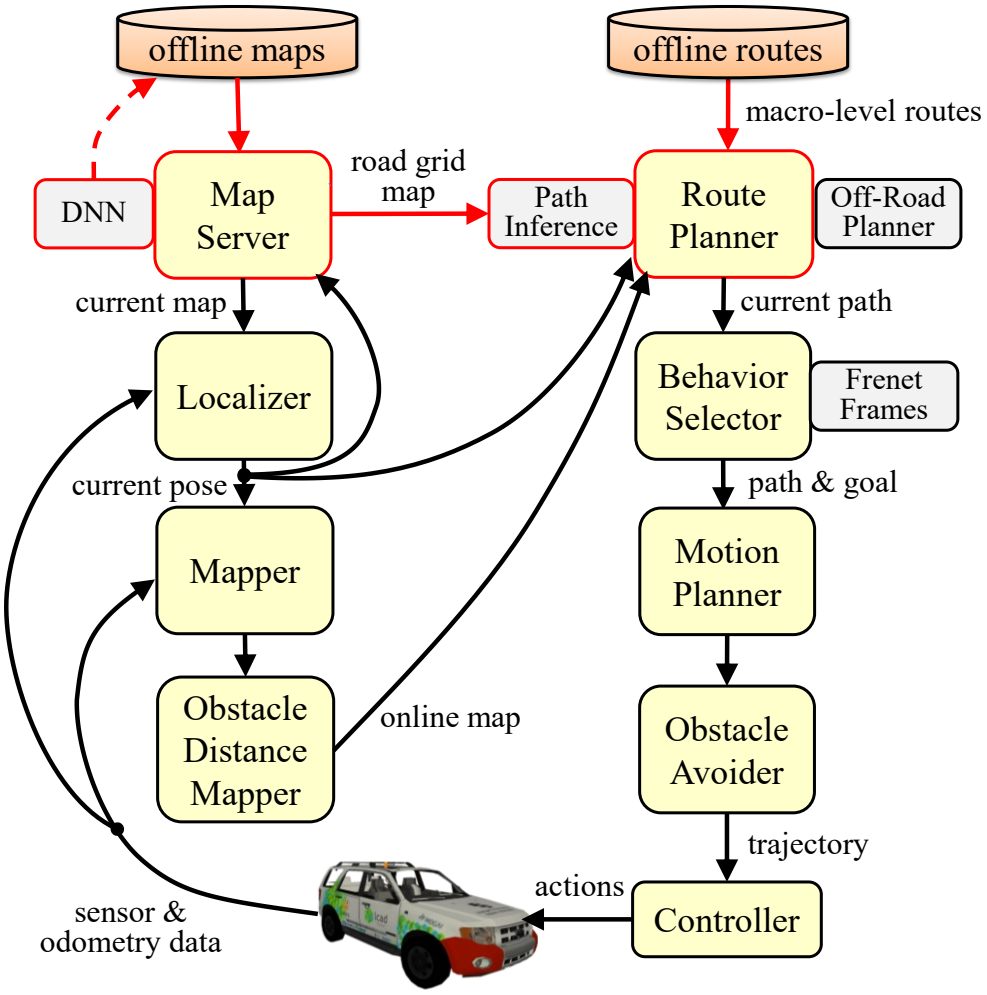


Fig. 23. New proposed IARA System simplified block diagram. The red lines indicate the enhanced modules and data flows.

The primary enhancements proposed for the IARA System focus on the Map Server and the Route Planner modules. A new Deep Neural Network (DNN) module has been introduced to perform semantic segmentation of road grid maps derived from

offline remission grid maps. This DNN module, tailored and trained based on one of the architectures detailed in Section 5, can operate in two distinct modes: (a) as an offline process that reads all the offline remission grid maps and generates corresponding road grid maps, which are stored on disk for later use; or (b) as an online process invoked by the Map Server module to dynamically process offline remission grid maps and publish online road grid maps to the subscriber modules in real time.

The major advantage of method (a) is its reduced resource demand during the online cycle of the IARA System, as all intensive DNN processing is completed offline. The IARA System's cycle, which consists of sensing – localizing – mapping – path planning – behavior decision-making – motion planning – controlling – sensing (in a continuous loop), must be completed in no more than 50 milliseconds to maintain a frequency of 20 Hertz. Therefore, each online module's task should take significantly less than 50 milliseconds to avoid degrading the overall system performance.

In contrast, method (b) requires substantially more online computational resources but offers key benefits such as reduced disk storage, fewer disk I/O operations, and greater flexibility to handle dynamic online remission grid maps, rather than relying on pre-processed offline versions. This approach would involve enhancing the Mapper module to integrate real-time LiDAR readings with offline remission grid maps.

In the experimental tests described in the following sections, method (a) was utilized due to resource constraints in the embedded computing devices of the IARA System.

Using either method (a) or (b), a road grid map is published for use by the subscriber Route Planner module. The Route Planner then invokes a new Path Inference process, responsible for computing a low-level path plan directly from a road grid map (refer to Section 6.2). This new approach replaces the initial step of path planning previously managed by the Route Planner, as detailed in Section 3.3. In this new proposed structure, there is no longer a need to store low-level routes (waypoint-level graph) on disk, as they are dynamically inferred by the Path Inference module. The Route Planner now only requires macro-level routes (lane-level graph) to

carry out the route planning process that precedes the path planning steps. This new approach offers several benefits, including reduced disk storage requirements, fewer disk I/O operations, and the flexibility to use the outputs of dynamic online mapping.

6.2. Computing Path Plans from Road Grid Maps

To compute the waypoints of an RDDF from a road grid map, we use Algorithm 1. When RDDFs are generated in this manner, the IARA System employs the Path Inference module depicted in Fig. 23.

Algorithm 1: compute_rddf_from_road_grid_map()

Input: *pose*, *road_grid_map*

Output: *RDDF_crop*

1: *rddf.wps_ahead* ← get_waypoints_ahead(*pose*, *road_grid_map, 150*)
2: *rddf.wps_behind* ← get_waypoints_behind(*pose*, *road_grid_map, 50*)
3: *RDDF_crop* ← smooth_rddf_using_conjugate_gradient(*rddf*)
4: **return** (*RDDF_crop*)

Algorithm 2: get_waypoints_ahead()

Input: *pose*, *road_grid_map, num_waypoints*

Output: *wps_ahead*

1: *next_pose* ← *pose*
2: **for** *i = 0* **to** *num_waypoints – 1*
3: *wps_ahead[i]* ← get_lane_central_pose(*next_pose*, *road_grid_map*)
4: *next_pose* ← add_distance_to_pose(*wps_ahead[i]*, *0.5*)
5: *next_pose.yaw* ← *atan2*(*next_pose.y – wps_ahead[i].y*, *next_pose.x – wps_ahead[i].x*)
6: **return** (*wps_ahead*)

In Algorithm 1, lines 1 and 2 compute 150 waypoints ahead and 50 waypoints behind the vehicle's current pose using the functions **`get_waypoints_ahead()`** and **`get_waypoints_behind()`**, respectively. These waypoints cover a distance of 75 meters ahead and 25 meters behind the vehicle. In line 3, the combined set of waypoints are smoothed using the **`smooth_rddf_using_conjugate_gradient()`** function. Smoothing is essential because the waypoints computed in lines 1 and 2 are

centered within map cells, and the cell dimensions (0.2 m × 0.2 m) are close to the average separation between waypoints (0.5 m). Without smoothing, this can result in a serrated path.

Algorithm 2 implements the **`get_waypoints_ahead()`** function. In line 3, the next waypoint in the sequence is computed using the **`get_lane_central_pose()`** function. This function takes as input **`next_pose = {x, y, yaw}`** and returns the **`{x, y}`** coordinates of the road grid map cell closest to the lane center, based on an orthogonal line crossing through **`next_pose`** (illustrated by the yellow line in Fig. 24 (b)). Initially, **`next_pose`** is set to the vehicle's current pose (line 1), so **`get_lane_central_pose()`** finds the map cell nearest to the lane center at the vehicle's current location. For the remainder of the loop, this function generates waypoints along the lane center, spaced 0.5 meters apart, due to the incremental distance added to **`next_pose`** in line 4.

Algorithm 3: get_waypoints_behind()

Input: *pose*, *road_grid_map, num_waypoints*

Output: *wps_behind*

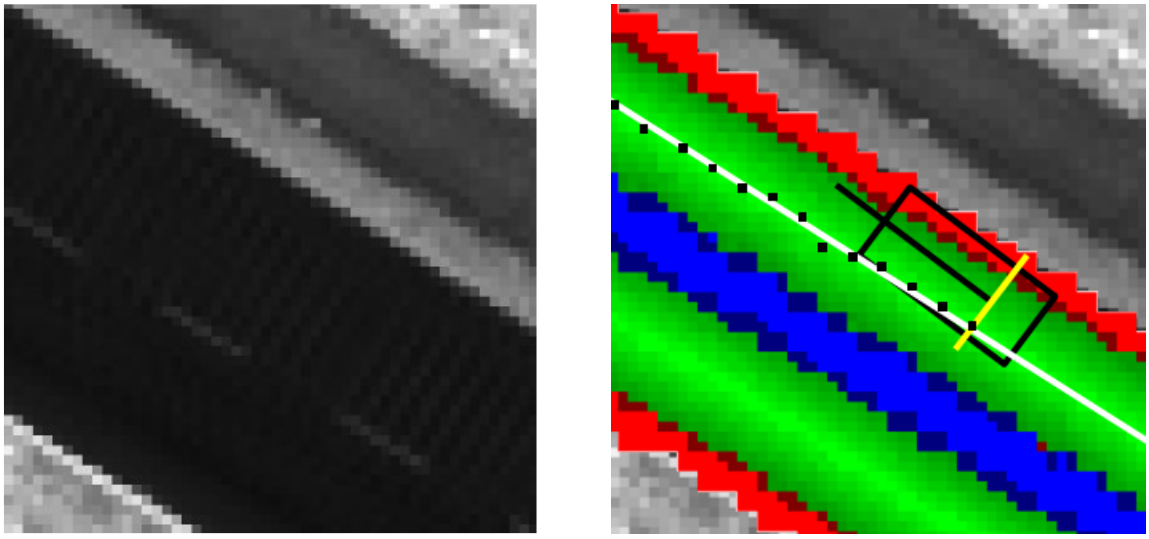
1: *previous_pose* ← *pose*
2: **for** *i = 0* **to** *num_waypoints – 1*
3: *wps_behind[i]* ← get_lane_central_pose(*previous_pose*, *road_grid_map*)
4: *previous_pose* ← add_distance_opposite_to_pose(*wps_behind[i]*, *0.5*)
5: *previous_pose.yaw* ← *atan2*(*wps_behind[i].y – previous_pose.y*,
wps_behind[i].x – previous_pose.x)
6: **return** (*wps_behind*)

The **`get_waypoints_behind()`** function, implemented in Algorithm 3, operates similarly to **`get_waypoints_ahead()`** but locates waypoints in the opposite direction.

We implemented the **`smooth_rddf_using_conjugate_gradient()`** function by modeling the smoothing optimization problem using the same objective function as Dolgov et al. in [81], but focusing exclusively on the last term that quantifies the path's smoothness (equation 1 in [81]). Therefore, the objective function defined in our algorithm is as follows:

$$\sum_{i=1}^{N-2} (\Delta w_{i+1} - \Delta w_i)^2 \quad (3)$$

- N: Number of waypoints.
- $w_i = \{x_i, y_i\}$: Coordinates of the i-th waypoint.
- $\Delta w_i = w_i - w_{i-1}$: Distance between the i-th waypoint and the previous one.



(a) (b)

Fig. 24. (a) A cropped section of a remission grid map. (b) The same section with a superimposed road grid map and the inferred RDDF waypoints.

In Fig. 24 (b), the vehicle is depicted as a transparent rectangle with black outlines, and its current pose is represented by the origin of the black line segment inside the rectangle. The yellow line segment indicates the orthogonal line used to search for the lane center. The black dots represent the waypoints ahead (`wps_ahead`) found by **`get_lane_central_pose()`** function, while the white line illustrates the final `RDDF_crop`, a smoothed path derived from `wps_ahead` and `wps_behind` using the **`smooth_rddf_using_conjugate_gradient()`** function.

7. EXPERIMENTAL METHODOLOGY

To evaluate the real-world application of path plans inferred from road grid maps segmented by DNNs, we employed the IARA autonomous vehicle (Fig. 3). Both the hardware and software of IARA were developed by researchers at UFES. The vehicle's hardware is based on a Ford Escape Hybrid car, which has been modified to enable electronic control of the steering, throttle, and brakes, in addition to capturing vehicle odometry data and powering multiple high-performance computers and sensors. IARA is equipped with a Velodyne HDL 32-E LiDAR, a Trimble dual RTK GPS, an Xsens MTi IMU, a Point Grey Bumblebee camera, a Point Grey Bumblebee XB3 stereo camera, and two Dell Precision R5500 computers.

IARA's software architecture consists of multiple modules that communicate via messages using the publish-subscribe paradigm. These modules were developed within the UFES version of the Carnegie Mellon Robot Navigation Toolkit (CARMEN [51]). The entire source code of IARA, including the code developed for this study, is publicly available[7].

To train the DNNs for segmenting road grid maps from remission grid maps, we created datasets covering tens of kilometers of manually marked road lanes. These datasets are publicly available[8]: (i) the UFES dataset (Fig. 26), which covers 3.7 km of the ring road at Universidade Federal do Espírito Santo (UFES) in Vitória, Brazil; (ii) the Highway dataset (Fig. 27), covering 32.4 km along Rodovia do Sol, a highway connecting the cities of Vila Velha and Guarapari, Brazil (distant from UFES); and (iii) the Industrial Plant dataset (Fig. 29), covering a route of 0.65 km in an industrial plant located in the State of São Paulo, Brazil.

[7] Available at https://github.com/LCAD-UFES/carmen_lcad.

[8] Available at https://bit.ly/road_ds.

7.1. Ground-Truth Marking Procedure

Each dataset consists of multiple cropped remission grid map images paired with their corresponding road grid map ground truth images. To create the ground truth images, we utilized the Inkscape[9] vector graphics software. For that, we saved each remission grid map as a PNG image and followed the procedure we developed (Fig. 25), which involves the following steps:

1. Open the image: Load a remission map image file in Inkscape and draw a polyline all the way along the center of each road lane.

2. Extend polyline limits: Position both the start and finish points beyond the image boundaries to ensure a better fit for each polyline.

3. Align with driving direction: Ensure that the lanes' driving orientation aligns with the direction from the polyline's start point to the finish point.

4. Handle intersections: If two polylines intersect, merge, or fork, draw the primary polyline first, followed by the secondary one.

5. Smooth the polyline: Set each polyline point to auto-smooth mode, converting the polyline into a cubic Bézier curve.

6. Set lane width: Adjust the stroke width of each polyline to the standard lane width we defined (3.2 m, or 16 cells).

7. Apply color coding: Assign the appropriate stroke color to each line according to the color code shown in TABLE 5.

This process is time-consuming. To expand the dataset's size and diversity, we augmented each pair of remission grid map and road grid map ground truth crops by:

[9] http://inkscape.org

(i) rotating them at different angles; and (ii) translating them by various distances across the vehicle's orientation direction (yaw).

TABLE 5. COLOR CODING FOR LINE MARKINGS IN THE GROUND TRUTH MARKING PROCEDURE

Color	RGB Value	Coding Description
	(255, 000, 000) = #ff0000	Solid line marking on both sides
	(255, 000, 127) = #ff007f	Solid line marking on the right and broken line on the left side
	(127, 000, 255) = #7f00ff	Broken line marking on the right and solid line on the left side
	(000, 000, 255) = #0000ff	Broken line marking on both sides
	(000, 255, 000) = #00ff00	No line marking on both sides
	(255, 127, 000) = #ff7f00	Solid line marking on the right and no line on the left side
	(127, 255, 000) = #7fff00	No line marking on the right and solid line on the left side
	(000, 127, 255) = #007fff	Broken line marking on the right and no line on the left side
	(000, 255, 127) = #00ff7f	No line marking on the right and broken line on the left side

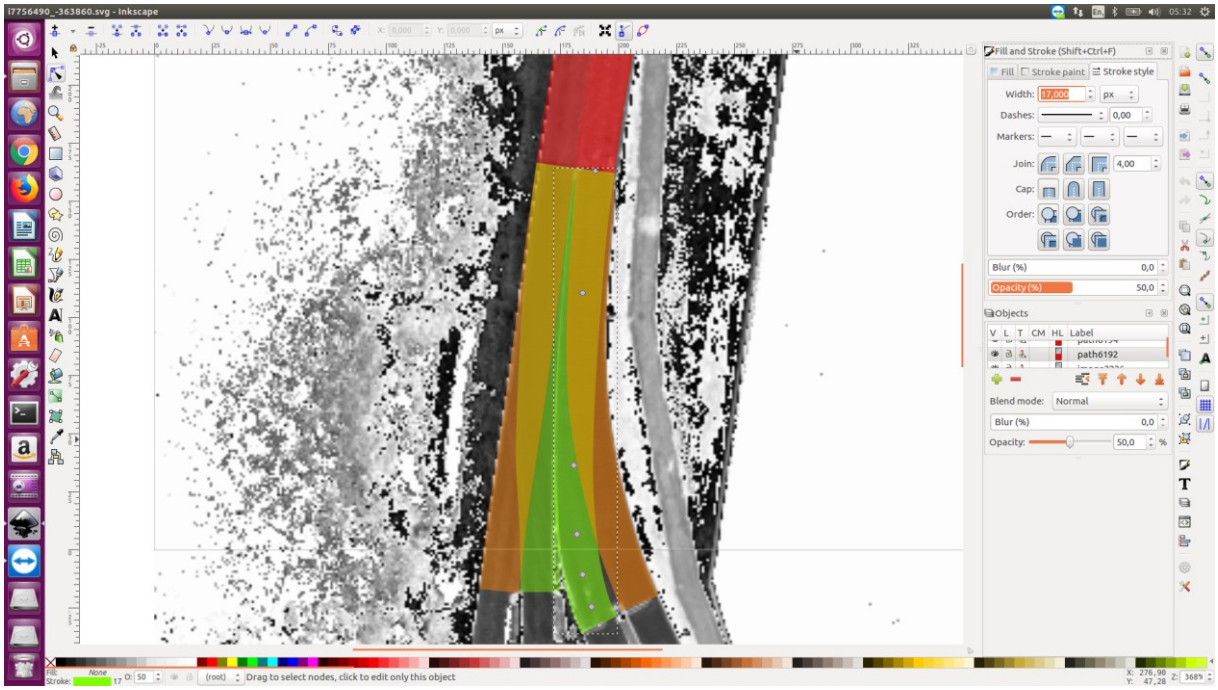


Fig. 25. The ground truth marking procedure using the Inkscape vector graphics software.

7.2. UFES Dataset

The UFES dataset consists of 658 cropped sections of 120 × 120 cells (covering an area of 24 m × 24 m) of remission grid maps, along with corresponding ground truth road grid maps. These were captured along the 3.7 km ring road of UFES (Fig. 26), with a spacing of approximately 5 meters between each section.

To augment the size and diversity of the dataset, we generated a total of 168 pairs for each original pair of remission grid map and corresponding road grid map ground truth crops. This was achieved by: (i) applying 24 different rotations (15 degrees apart), and (ii) translating them with 7 different shifts (–1.5 m, –1.0 m, –0.5 m, 0.0 m, 0.5 m, 1.0 m, 1.5 m) across the direction of the ring road at the location of the original pair of crops. The final UFES dataset comprises 110,544 (658 × 168) pairs of crops.

Fig. 26. The 3.7 km ring road of the Universidade Federal do Espírito Santo (UFES), Brazil, located at coordinates 20°16.4313'S 40°18.3653'W.

To mitigate potential overfitting issues, which we suspected were caused by excessive data augmentation, we defined a subset of the UFES dataset with a reduced number of transformations. This subset, named UFES-II, includes only 15,792 rotated image pairs (658 × 24), excluding all translated image pairs from this subset.

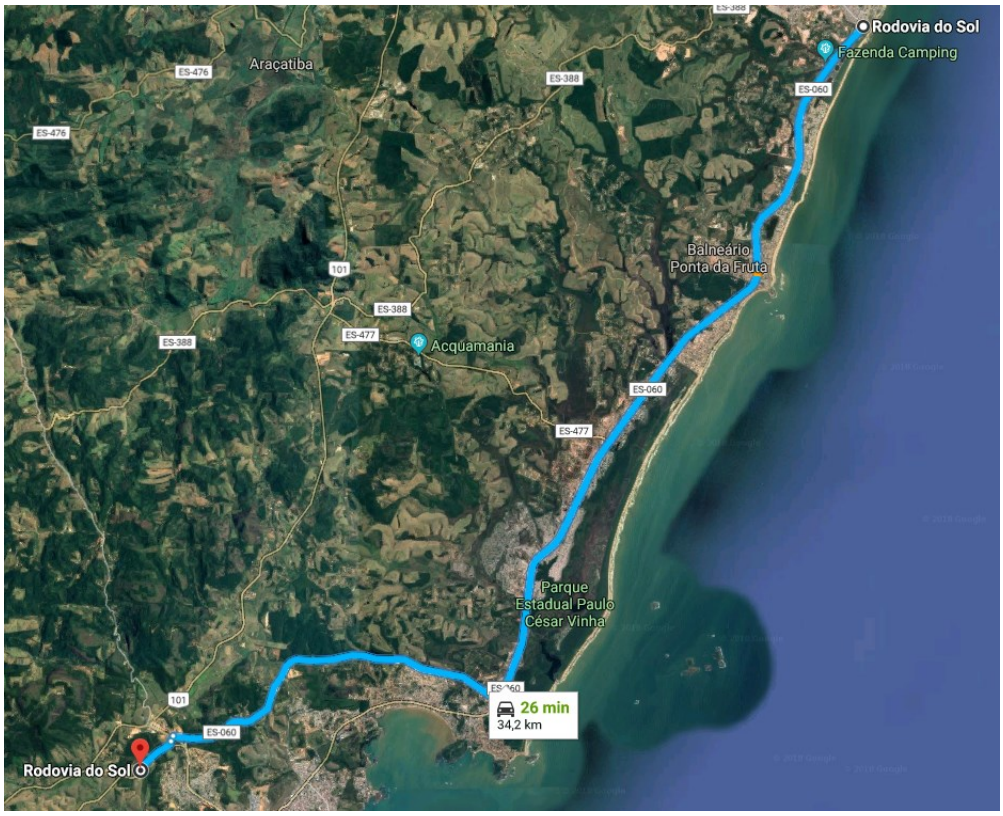


Fig. 27. The 32.4 km of the Rodovia do Sol highway in the State of Espírito Santo, Brazil, located at coordinates 20°27.5838'S 40°20.5988'W.

7.3. Highway Dataset

The Highway dataset consists of 3,556 cropped sections of 120 × 120 cells (covering an area of 24 m × 24 m) of remission grid maps, along with corresponding ground truth road grid maps, captured along 32.4 km of the Rodovia do Sol highway (Fig. 27), with a spacing of approximately 12 meters between each section. This dataset was used primarily for testing; therefore, no rotations or translations were applied. We used the Highway dataset to evaluate our model's ability to generalize in a different context that was not encountered during training of the DNNs.

7.4. Industrial Plant Dataset

The Industrial Plant dataset was generated following the standard procedure outlined in Section 3.4.

The odometry and sensor readings were collected using a Mercedes-Benz Atego 1730 truck, as shown in Fig. 28. The vehicle was equipped with three Ouster OS1-32 LiDARs: one mounted on top of the truck cabin, attached to a metal box, configured in below-horizon mode, with a vertical field of view ranging from the horizon down to 22.5 degrees below; and two units mounted on each side of the vehicle, just above the headlights, configured in uniform mode, with a vertical field of view spanning from 22.5 degrees above to 22.5 degrees below the horizon.

In autonomous mode, this truck operates using a hardware and software framework closely aligned with the one detailed in Section 3.

Fig. 28. Mercedes-Benz truck used for logging odometry and sensor readings at the industrial plant.

Fig. 29. The 0.65 km route of an industrial plant in the State of São Paulo, Brazil, located at coordinates 22°43.0933'S 46°47.4720'W.

The Industrial Plant dataset consists of 145 cropped sections of 120 × 120 cells (each covering an area of 24 m × 24 m) of remission grid maps and corresponding ground truth road grid maps, captured along a 0.65 km route of an industrial plant, as shown in Fig. 29, with a spacing of approximately 5 meters between each section.

To augment the dataset's size and diversity, we generated a total of 24 pairs for each original pair of crops by applying 24 different rotations, each 15 degrees apart. The final Industrial Plant dataset comprises 3,480 (145 × 24) pairs of crops.

7.5. U-Net DNN

We employed a slightly modified version of the U-Net architecture, available at a GitHub repository[10]. Our modifications include integrating dropout and batch normalization layers to reduce overfitting and improve the training process.

The model processes 256 × 256-pixel remission images as input, with 256 × 256-pixel road grid maps serving as the ground truth. To ensure compatibility with these image dimensions, padding is applied prior to feeding the images into the model and removed before calculating the evaluation metrics.

In the experiments detailed in Section 8, the model was trained using different training sets: (i) the UFES dataset alone; (ii) both the UFES and Industrial Plant datasets; and (iii) a combined set consisting of UFES, Industrial Plant, and Highway datasets.

The U-Net model was trained with a batch size of 8 and an initial learning rate of 0.001. The LION optimizer was employed with a weight decay of 0.001. The CrossEntropy loss function was used, with class weights computed based on the distribution of the training data.

To evaluate the model's performance, we used multiple metrics. Class average accuracy was calculated as the ratio of correctly predicted classes to the total number of evaluated map cells. F1 score was also employed to provide a harmonic mean of precision and recall, ensuring a balanced assessment of the model's accuracy. Lastly, the mean Intersection over Union (mean IoU) was used to measure the overlap between predicted and ground truth segments, offering a comprehensive evaluation of segmentation performance across all classes.

[10] https://github.com/xiaochengcike/pytorch-unet-1

7.6. ENet DNN

We utilized two versions derived from the original ENet architecture. The first version was modified to accept grayscale PNG images of 120 × 120 pixels representing remission grid maps as input, with corresponding 120 × 120-pixel road grid maps serving as ground truth. This version was assessed in [10] and is available at a GitHub repository[11]. This model was trained using the UFES dataset, and its performance was evaluated on both the UFES test dataset (20% of the images) and the entire Highway dataset.

The training process for the encoder utilized the following parameters: a batch size of 16, an initial learning rate of 0.005, reduced by a factor of 10 every epoch, and the ADAM optimizer, configured with a first momentum of 0.9, a second momentum of 0.999, and a weight decay of 0.0002. These same parameters were applied during the training of the full network (encoder and decoder).

To assess ENet's performance, we used class average accuracy as the primary metric. This was calculated as the ratio of correctly predicted classes to the total number of evaluated map cells, chosen for its simplicity and suitability for this segmentation task. Additionally, qualitative metrics related to the model's practical application in navigation were considered, specifically: (i) the ability to maintain the vehicle centered in the lane, and (ii) avoiding abrupt steering movements, which are critical for assessing real-world usability.

The second version of the ENet model, available at a GitHub repository[12], was adapted to accept 256 × 256-pixel remission images as input, with 256 × 256-pixel road grid maps as the ground truth. To handle varying image dimensions, padding was

[11] https://github.com/LCAD-UFES/carmen_lcad/tree/master/sharedlib/ENet

[12] https://github.com/iArunava/ENet-Real-Time-Semantic-Segmentation/tree/master

applied before feeding the images into the model and removed prior to calculating evaluation metrics.

For this version of ENet, the same training datasets were used as in the U-Net experiments, including the UFES, Industrial Plant, and Highway datasets. The training configuration also mirrored that of U-Net, with a batch size of 8, an initial learning rate of 0.001, the LION optimizer with a weight decay of 0.001, and the CrossEntropy loss function with class weights calculated based on the training data distribution.

The performance of this ENet model was evaluated using the same metrics as those applied to U-Net: class average accuracy, F1 score, and mean Intersection over Union (mean IoU), ensuring consistency in the assessment of both architectures.

7.7. Swin-Unet DNN

We used a slightly modified version of Swin-Unet, available at a GitHub repository[13]. The primary modification involved adapting the model to accept single-channel grayscale images as input.

The model was designed to process 256 × 256-pixel remission images as input, with 256 × 256-pixel road grid maps as the ground truth. To handle varying image dimensions, padding was applied before feeding the images into the model and removed prior to calculating evaluation metrics.

For this model, the same training datasets were used as in the U-Net experiments, including the UFES, Industrial Plant, and Highway datasets. The training configuration also mirrored that of U-Net, with a batch size of 8, an initial learning rate of 0.001, the

[13] https://github.com/Wzysaber/ST_Unet_pytorch_Semantic-segmentation/tree/main/model/SwinUnet

LION optimizer with a weight decay of 0.001, and the CrossEntropy loss function with class weights calculated based on the training data distribution.

The performance of this model was evaluated using the same metrics as those applied to U-Net: class average accuracy, F1 score, and mean Intersection over Union (mean IoU), ensuring consistency in the assessment of both architectures.

7.8. ST-UNet DNN

We used a slightly modified version of ST-UNet, available at a GitHub repository[14]. The modification involved configuring the model to accept single-channel (grayscale) images as input.

The model was designed to process 256 × 256-pixel remission images as input, with 256 × 256-pixel road grid maps as the ground truth. To handle varying image dimensions, padding was applied before feeding the images into the model and removed prior to calculating evaluation metrics.

For this model, the same training datasets were used as in the U-Net experiments, including the UFES, Industrial Plant, and Highway datasets. The training configuration also mirrored that of U-Net, with a batch size of 8, an initial learning rate of 0.001, the LION optimizer with a weight decay of 0.001, and the CrossEntropy loss function with class weights calculated based on the training data distribution.

The performance of this model was evaluated using the same metrics as those applied to U-Net: class average accuracy, F1 score, and mean Intersection over Union (mean IoU), ensuring consistency in the assessment of both architectures.

[14] https://github.com/Wzysaber/ST_Unet_pytorch_Semantic-segmentation/tree/main/model/ST_Unet

TABLE 6 summarizes the training of all DNNs.

TABLE 6. DNN TRAINING CONFIGURATION

Model	Batch Size	Computing Configuration	Training Time	Number of Parameters
U-Net	8	GPU: RTX 2080, 8GB CPU: AMD Ryzen Threadripper 3970X 32-Core Processor 62 GB RAM	02:56:50	31,042,694
ENet	8	GPU: NVIDIA T4, 15 GB CPU: Intel(R) Xeon(R) 2.20GHz 12.7 GB RAM	01:51:43	353,052
Swin-Unet	8	GPU: NVIDIA T4, 15 GB CPU: Intel(R) Xeon(R) 2.20GHz 12.7 GB RAM	02:13:22	12,306,058
ST-UNet	8	GPU: RTX 2080, 8 GB CPU: AMD Ryzen Threadripper 3970X 32-Core Processor, 62 GB RAM	03:26:41	105,321,958

8. EXPERIMENTAL RESULTS

A series of experiments were conducted to evaluate the performance of semantic segmentation using the proposed coding patterns, the selected DNNs, and different combinations of datasets for training, validation and testing.

Experiment #1 reported in [10] evaluated the ENet DNN with coding pattern #1, using UFES dataset for training, and the Highway dataset for testing. Experiment #2 evaluated the ENet DNN with coding pattern #2, comparing it to the results from the first experiment, which used coding pattern #1. Additionally, it compared the performance of four models (U-Net, ENet, Swin-Unet, ST-UNet) with coding pattern #2, using UFES dataset for training, and both the Highway and the Industrial Plant datasets for testing. Experiment #3 reduced the number of samples of UFES training dataset, by eliminating the original augmentation, and included the Industrial Plant dataset in the training data, using the Highway dataset for testing. Experiment #4 used the three datasets for training and testing (UFES, Industrial Plant, and Highway), but ensured that the samples included in the testing dataset were not used in the validation during training. Additional experiments were conducted with the IARA autonomous car to evaluate the performance of the proposed method in the real world.

8.1. Experiment #1

The first experiment, conducted in [10], evaluated the performance of ENet DNN (version 1) using coding pattern #1, a batch size of 16, and 3 epochs of training.

TABLE 7. EXPERIMENT #1 CONFIGURATION

Model	ENet (version 1)
Training Dataset	UFES (80%): 88,368 images in 16,569 batches
Validation Dataset	UFES (20%): 22,176 images
Testing Dataset 1	UFES (same used for validation)
Testing Dataset 2	Highway: 3,556 images

Fig. 30 presents the experimental results for training full ENet model (encoder and decoder). On the x-axis, the number of batches trained is shown, while the y-axis represents the model's accuracy. As illustrated, accuracy improves steadily with the number of training batches but plateaus around 2,000 batches. A second increase is observed when the learning rate is reduced to 0.0005, reaching another plateau at around 5,000 batches. Further reductions in the learning rate did not result in noticeable improvements, with the accuracy stabilizing at approximately 78% during the training.

Fig. 31 shows that ENet achieved an average accuracy of 83.7% on the UFES test dataset, exceeding the accuracy observed during training. This improvement is attributed to the use of SpatialDropout [82] during training, which enhances generalization. Since SpatialDropout is disabled during testing, the model's performance appears higher. When tested on the Highway dataset, ENet achieved an average accuracy of 64.1%.

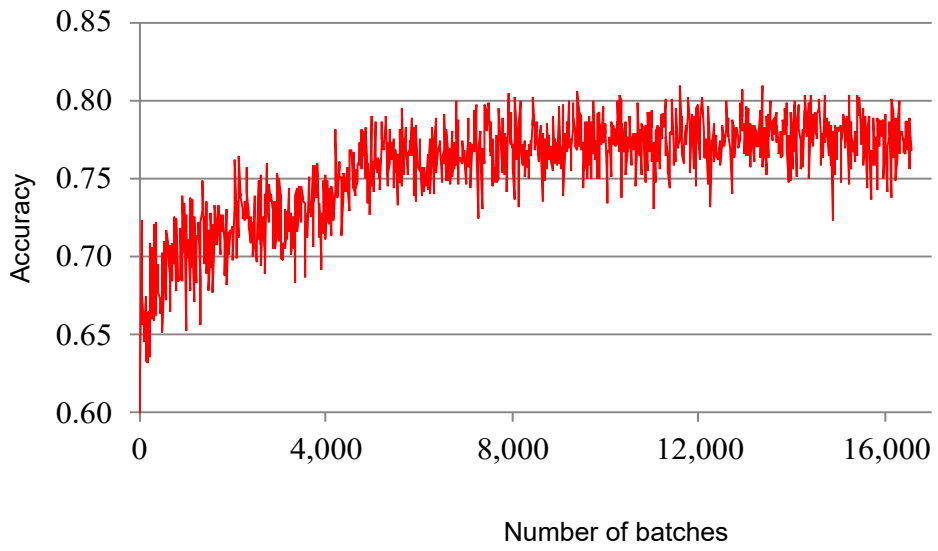


Fig. 30. Experiment #1: ENet encoder-decoder's accuracy achieved during training.

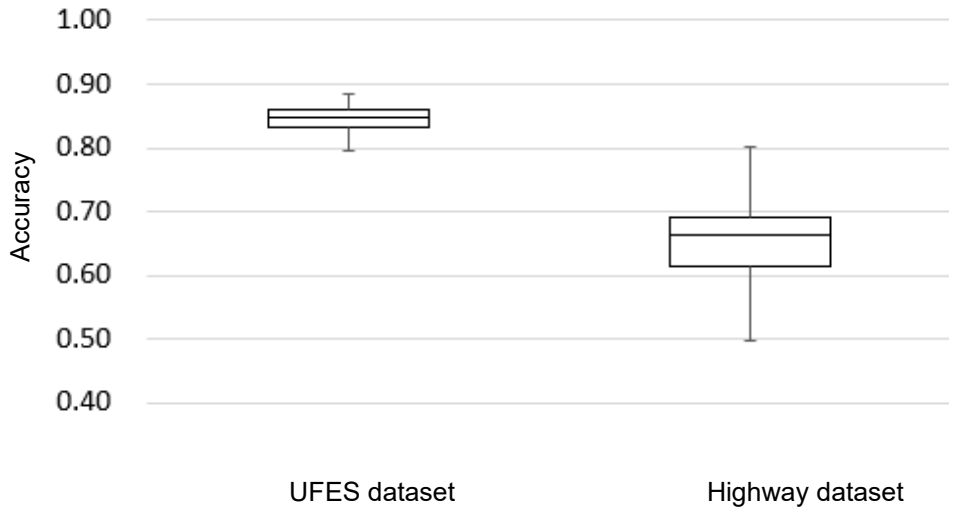
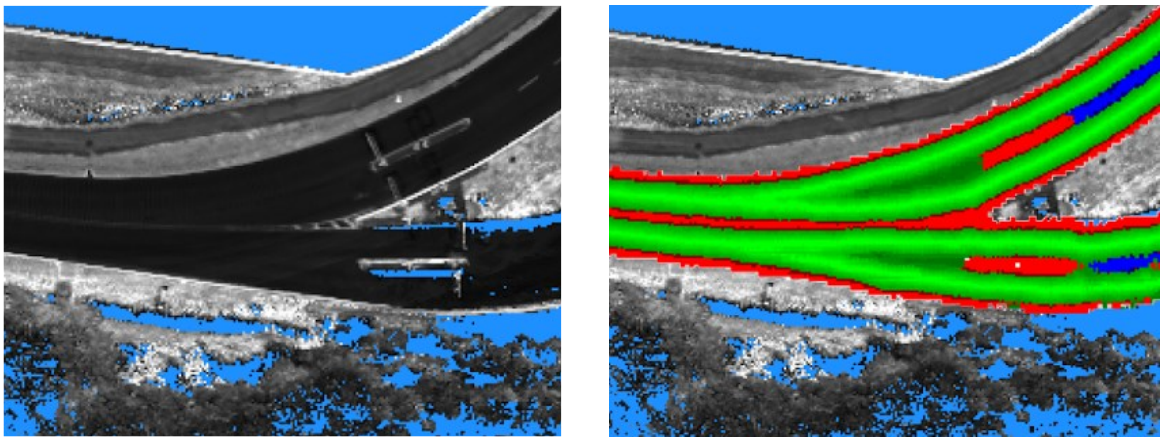
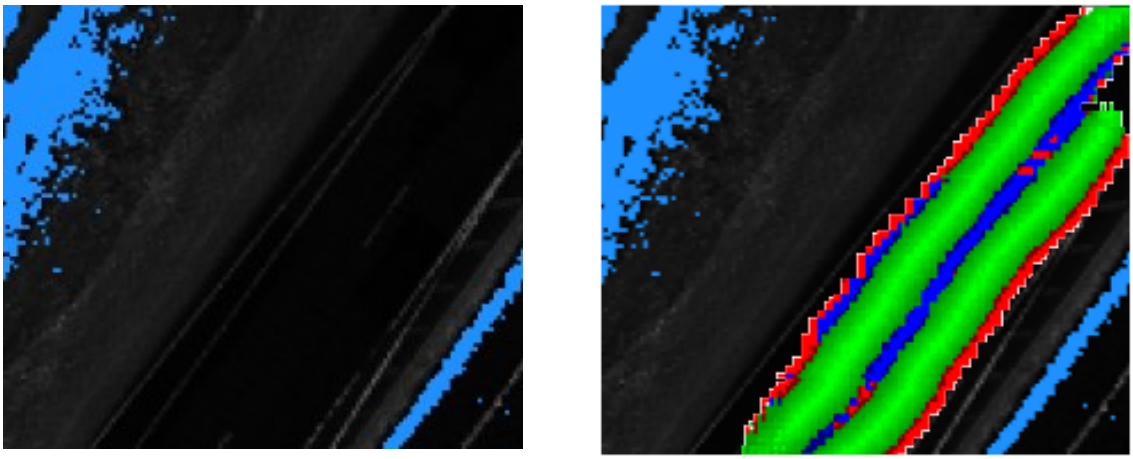


Fig. 31. Experiment #1: Box plot of ENet's accuracy achieved during testing.



(a) (b)

Fig. 32. Test of inference on the ring road of UFES. (a) Crop of a remission grid map. (b) Same as (a) with superimposed road grid map inferred by ENet with 83% of class average accuracy.



(a) (b)

Fig. 33. Test of inference on the Highway dataset. (a) Crop of a remission grid map. (b) Same as (a) with superimposed inferred road grid map with 64% of class average accuracy.

Fig. 32 presents an example of ENet inference on a segment of the UFES test dataset. In Fig. 32 (a), the input remission grid map is shown, while Fig. 32 (b) displays the same remission grid map with the inferred road grid map superimposed. Fig. 33 provides an example of ENet's performance on the Highway dataset. Despite the lower accuracy, the model was still able to reasonably extract the road grid map from the remission grid map, even in an unseen region with significantly different surroundings from those in the training data.

8.2. Experiment #2

The second experiment evaluated the performance of ENet DNN (version 2) using coding pattern #2, comparing it to the results from the first experiment. Additionally, it compared the performance of several models trained with coding pattern #2, using a batch size of 8 and 3 epochs of training.

TABLE 8. EXPERIMENT #2 CONFIGURATION

Models	U-Net, ENet (version 2), Swin-Unet, ST-UNet
Training Dataset	UFES (80%): 88,368 images
Validation Dataset	UFES (20%): 22,176 images
Testing Dataset 1	UFES (same used for validation)
Testing Dataset 2	Industrial Plant: 3,480 images
Testing Dataset 3	Highway: 3,556 images

The following tables and figures show the results of experiment #2. In the table, the legend (t) denotes that the testing dataset was not used for validation during training, while (v) indicates that it was included for validation.

TABLE 9. COMPARED PERFORMANCE OF ENET USING DIFFERENT CODING PATTERNS

Model	Coding Pattern	Testing Dataset	Average Accuracy	Median Accuracy	IQR Accuracy
ENet	Exp #1	UFES (v)	83.75%	84.78%	2.68%
ENet	Exp #2	UFES (v)	93.23%	94.51%	2.27%
ENet	Exp #1	Highway (t)	64.10%	66.41%	7.73%
ENet	Exp #2	Highway (t)	64.38%	65.50%	9.94%

TABLE 9 shows that coding pattern #2 allowed a significant improvement in the accuracy of the model using UFES dataset. In other words, the DNN was able to learn more precisely the dataset used during training. There was no significant variation in the accuracy using the Highway dataset, which was not seen during training. This means that the model's generalization ability was not improved, neither reduced.

Fig. 34 to Fig. 37 show compared performances of multiple models using coding pattern #2, on the three datasets. Using UFES dataset, ST-UNet achieved a performance slightly better than U-Net and ENet, and significantly better than Swin-Unet. Using the Industrial Plant dataset, all four models achieved a similar performance. Using the Highway dataset ST-UNet achieved the best performance. Using any of the three datasets. U-Net and ENet achieved very similar performances.

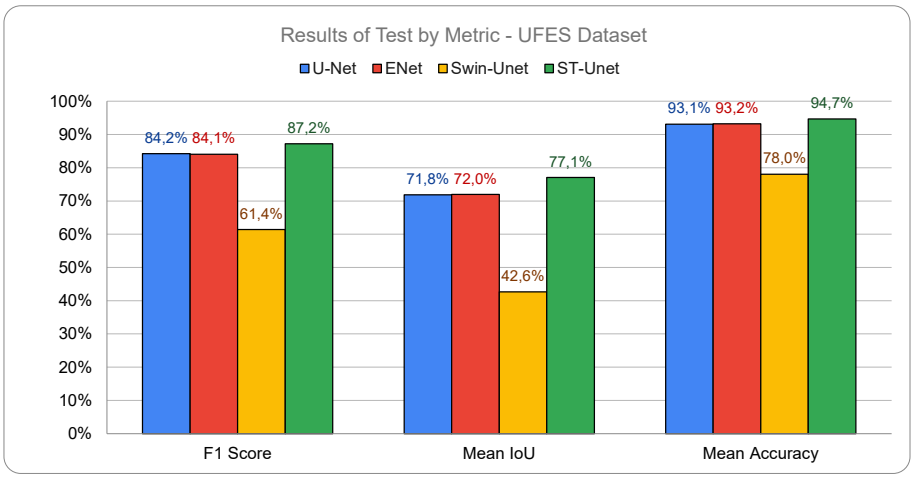


Fig. 34. Performance metrics of multiple DNNs using UFES dataset and coding pattern #2.

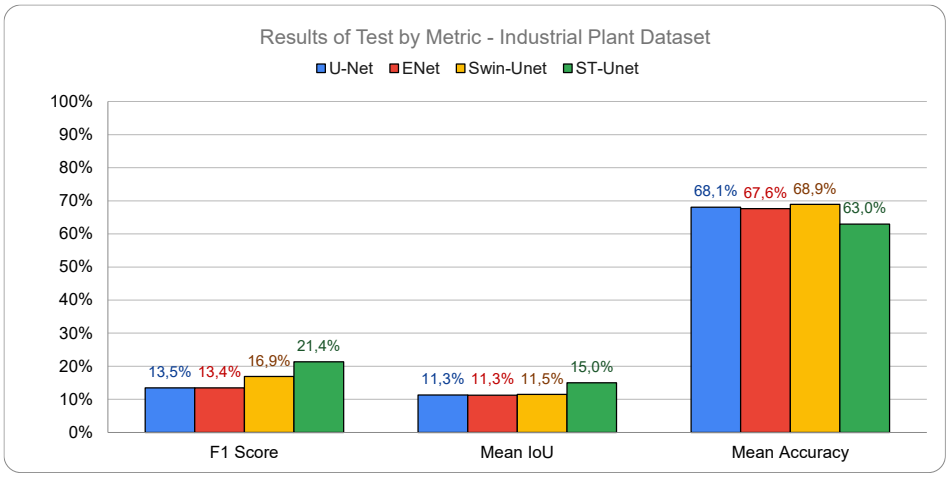


Fig. 35. Performance metrics of multiple DNNs using the Industrial Plant dataset and coding pattern #2.

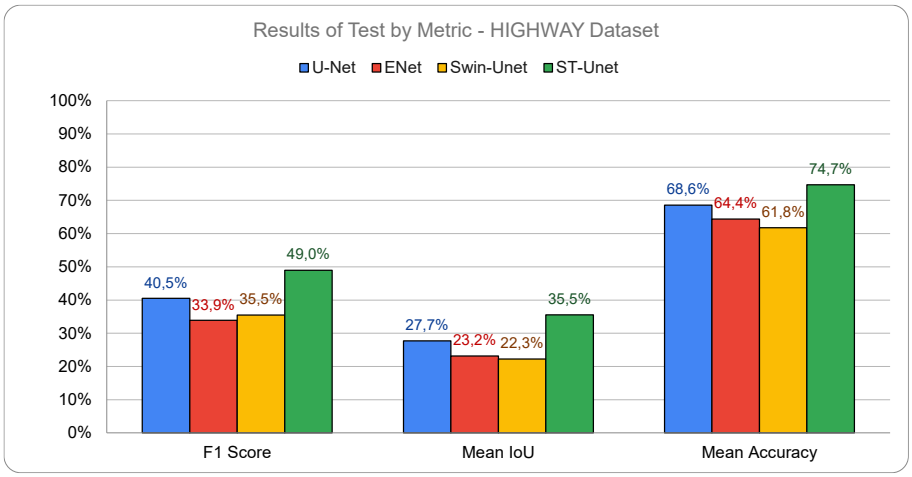


Fig. 36. Performance metrics of multiple DNNs using the Highway dataset and coding pattern #2.

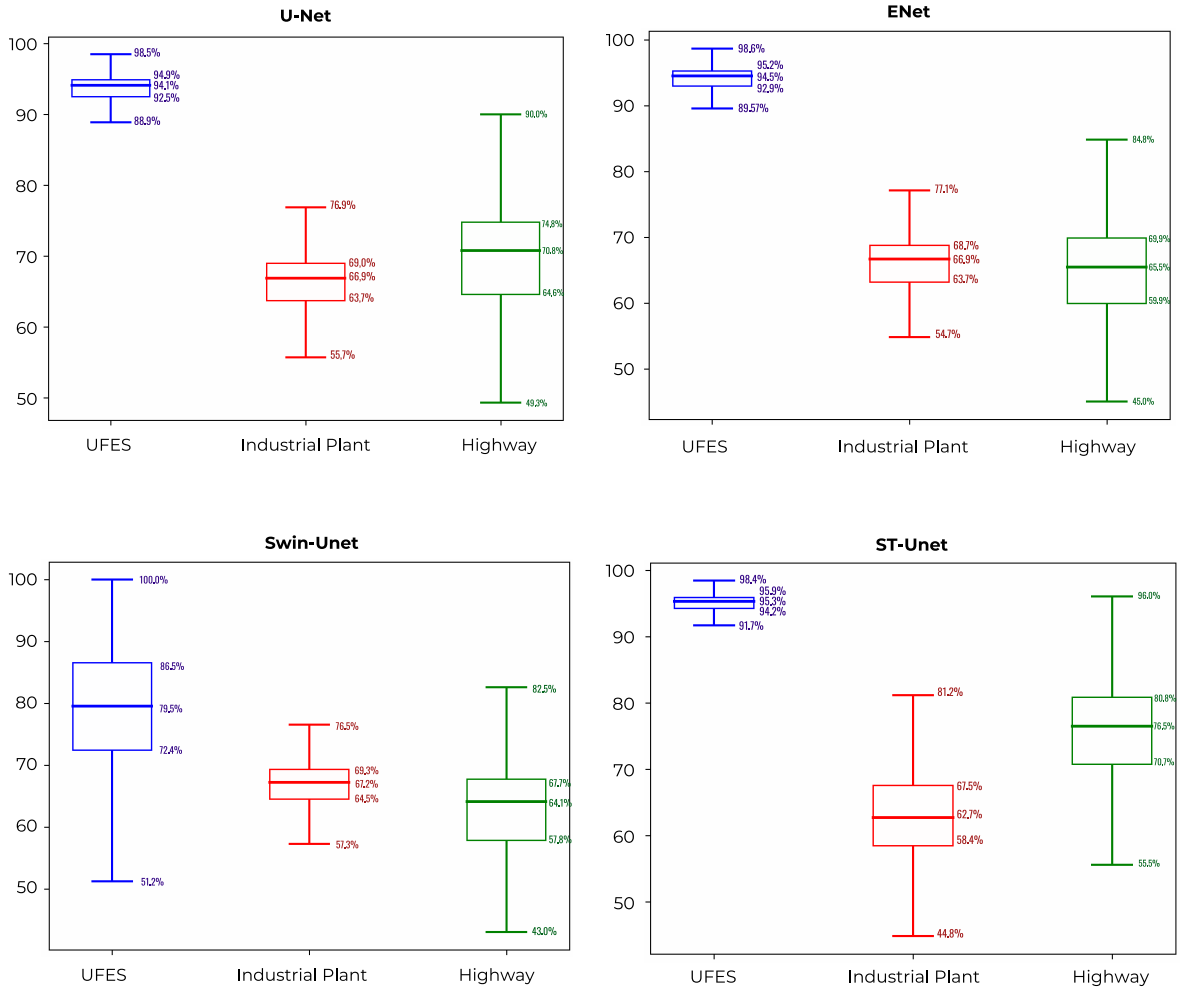


Fig. 37. Box plot of accuracy of multiple DNNs using the three datasets and coding pattern #2.

8.3. Experiment #3

In the third experiment, we reduced the number of samples of UFES training dataset, by eliminating the original augmentation made with 6 translations per image. Additionally, we included the Industrial Plant dataset in the training data.

TABLE 10. EXPERIMENT #3 CONFIGURATION

Model	U-Net
Training Dataset	UFES-II (80%): 12,624 images Industrial Plant (80%): 2,784 images
Validation Dataset	UFES-II (20%): 3,168 images Industrial Plant (20%): 696 images
Testing Dataset 1	UFES-II (same used for validation)
Testing Dataset 2	Industrial Plant: (same used for validation)
Testing Dataset 3	Highway: 3,556 images

TABLE 11 compares the performance of the U-Net DNN in the third experiment with the results from the second experiment. In the table, the legend (t) denotes that the testing dataset was not used for validation during training, while (v) indicates that it was included for validation.

TABLE 11. COMPARED PERFORMANCE OF U-NET USING DIFFERENT TRAINING DATASETS

Model	Training Dataset	Testing Dataset	Average Accuracy	Median Accuracy	IQR Accuracy
U-Net	Exp #2	UFES-II (v)	93.12%	94.13%	2.40%
U-Net	Exp #3	UFES-II (v)	93.87%	94.77%	2.11%
U-Net	Exp #2	Industrial Plant (t)	68.07%	66.90%	5.32%
U-Net	Exp #3	Industrial Plant (v)	87.41%	90.72%	6.93%
U-Net	Exp #2	Highway (t)	68.55%	70.86%	10.18%
U-Net	Exp #3	Highway (t)	69.02%	70.54%	14.45%

TABLE 11 and Fig. 38 show that the accuracy of the model using UFES dataset was not impacted by the shortening of the samples included in the training dataset of experiment #3. Presumably, the augmentation produced by merely translating the images was not helpful for accuracy. The model was able to learn much more precisely the Industrial Plant dataset, as it was used during training. There was no significant variation in the accuracy using the Highway dataset, which was not seen during

training. This means that the DNN's generalization ability was not improved, nor reduced.

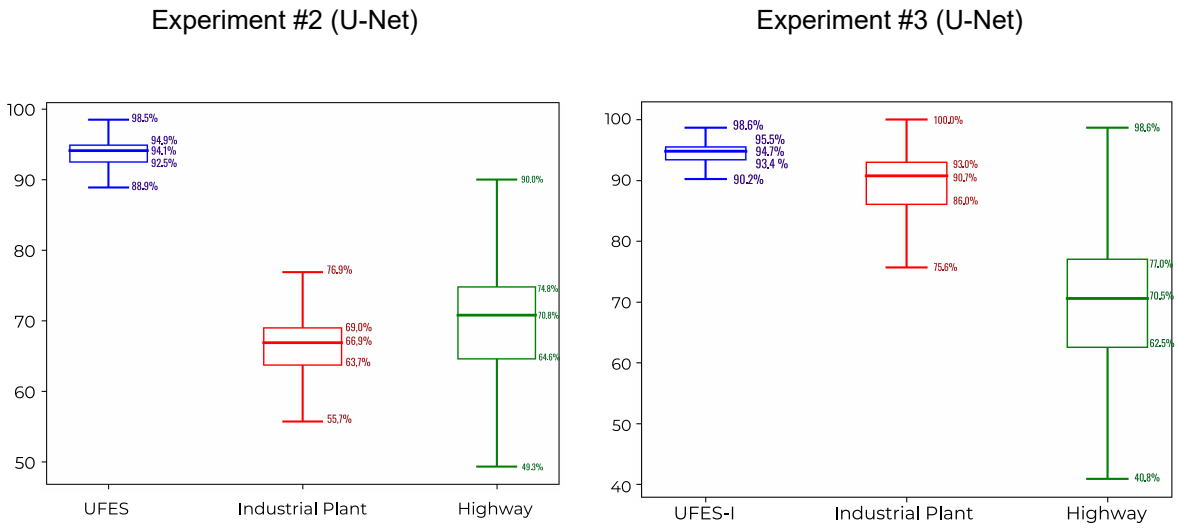


Fig. 38. Box plot of accuracy of U-Net DNN using the datasets from experiments #2 and #3.

8.4. Experiment #4

In the fourth experiment, we reduced the number of samples of UFES training dataset from 12,624 to 11,040, thus saving 1,584 samples for the testing dataset, which was not used for validation during training. Likewise, we reduced the number of samples of the Industrial Plant training dataset from 2,784 to 2,424, thus saving 360 samples for the testing dataset, which was not used for validation during training. Additionally, we included 2,488 samples of the Highway dataset in the training data, saving 356 samples for the testing dataset, which was not used for validation during training.

TABLE 12. EXPERIMENT #4 CONFIGURATION

Models	U-Net, ST-UNet
Training Dataset	UFES-II (70%): 11,040 images Industrial Plant (70%): 2,424 images Highway (70%): 2,488 images
Validation Dataset	UFES-II (20%): 3,168 images Industrial Plant (20%): 696 images Highway (20%): 712 images
Testing Dataset 1	UFES-II (10%): 1,584 images
Testing Dataset 2	Industrial Plant (10%): 360 images
Testing Dataset 3	Highway (10%): 356 images

TABLE 13 compares the performance of the U-Net DNN in the fourth experiment with the results from the third experiment. In the table, the legend (t) denotes that the testing dataset was not used for validation during training, while (v) indicates that it was included for validation.

TABLE 13. COMPARED PERFORMANCE OF U-NET USING DIFFERENT TRAINING DATASETS

Model	Training Dataset	Testing Dataset	Average Accuracy	Median Accuracy	IQR Accuracy
U-Net	Exp #3	UFES-II (v)	93.87%	94.77%	2.11%
U-Net	Exp #4	UFES-II (v)	92.74%	94.59%	2.87%
U-Net	Exp #4	UFES-II (t)	86.97%	90.21%	10.92%
U-Net	Exp #3	Industrial Plant (v)	87.41%	90.72%	6.93%
U-Net	Exp #4	Industrial Plant (v)	85.05%	90.38%	14.52%
U-Net	Exp #4	Industrial Plant (t)	74.58%	82.90%	27.34%
U-Net	Exp #3	Highway (t)	69.02%	70.54%	14.45%
U-Net	Exp #4	Highway (v)	82.76%	84.85%	10.83%
U-Net	Exp #4	Highway (t)	82.61%	84.61%	11.04%

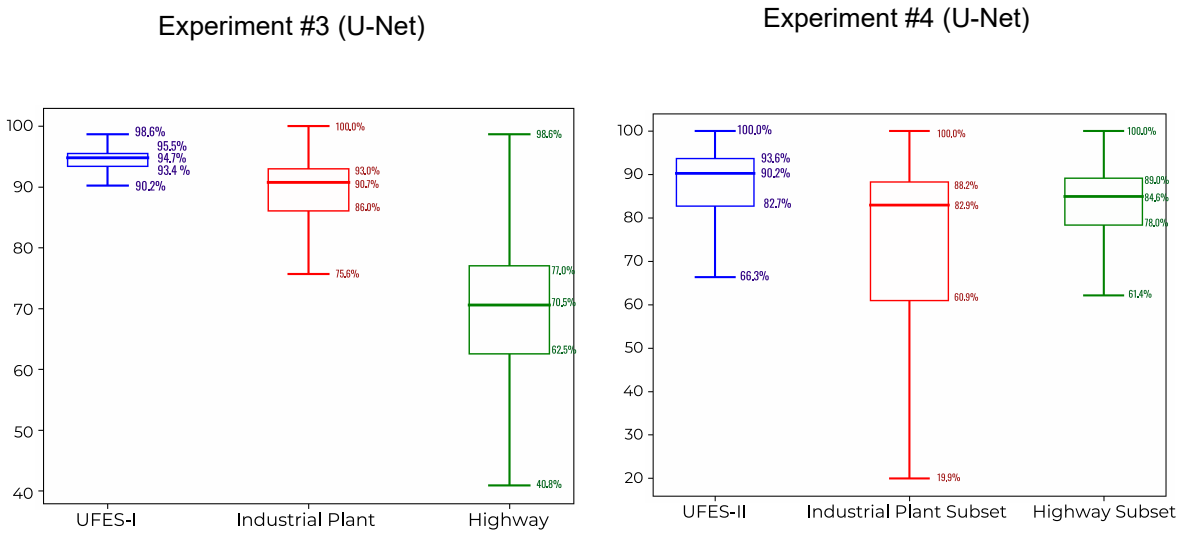


Fig. 39. Box plot of accuracy of U-Net DNN using the datasets from experiments #3 and #4.

TABLE 13 and Fig. 39 show that the accuracy of U-Net, tested on the UFES and Industrial Plant datasets, dropped by 7 percentage points in experiment #4, while the interquartile range (IQR) increased. This reduction can be attributed to the fact that, in experiment #4, the testing dataset contains only samples that were not used for validation during the training. In contrast, performance remained relatively stable when the testing dataset was included in the validation during training. For the Highway dataset, U-Net demonstrated significantly better learning in experiment #4, even when the testing dataset was not part of the validation set during training.

0 compares the performance of the ST-UNet DNN in the fourth experiment with the results from the second experiment. In the table, the legend (t) denotes that the testing dataset was not used for validation during training, while (v) indicates that it was included for validation.

TABLE 14. COMPARED PERFORMANCE OF ST-UNET USING DIFFERENT TRAINING DATASETS

Model	Training Dataset	Testing Dataset	Average Accuracy	Median Accuracy	IQR Accuracy
ST-UNet	Exp #2	UFES-II (v)	94.66%	95.35%	1.68%
ST-UNet	Exp #4	UFES-II (v)	93.06%	95.17%	2.82%
ST-UNet	Exp #4	UFES-II (t)	83.57%	86.05%	16.77%

Model	Training Dataset	Testing Dataset	Average Accuracy	Median Accuracy	IQR Accuracy
ST-UNet	Exp #2	Industrial Plant (t)	62,98%	62,71%	9,09%
ST-UNet	Exp #4	Industrial Plant (v)	89.88%	93.88%	8.06%
ST-UNet	Exp #4	Industrial Plant (t)	77.91%	83.42%	25.00%
ST-UNet	Exp #2	Highway (t)	74.69%	76.51%	10.13%
ST-UNet	Exp #4	Highway (v)	84.25%	85.79%	9.92%
ST-UNet	Exp #4	Highway (t)	83.41%	83.98%	11.92%

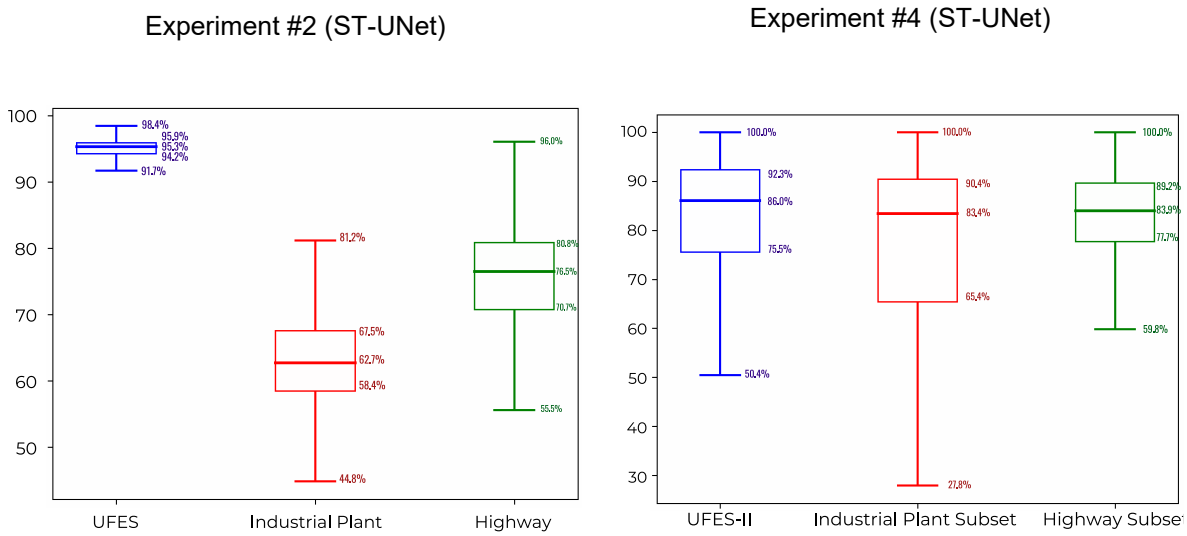


Fig. 40. Box plot of accuracy of ST-UNet DNN using the datasets from experiments #2 and #4.

0 and Fig. 40 show that the accuracy of ST-UNet, tested on the UFES dataset, dropped by 10 percentage points in experiment #4, while the interquartile range (IQR) increased. This reduction can be attributed to the fact that, in experiment #4, the testing dataset contains only samples that were not used for validation during the training. In contrast, performance remained relatively stable when the testing dataset was included in the validation during training. For the Industrial Plant and Highway datasets, ST-UNet demonstrated significantly better learning in experiment #4, when the testing dataset was not part of the validation set during training.

8.5. Experiments with IARA in the Real World

A key question to consider is: what constitutes an appropriate evaluation metric for this specific semantic segmentation task? As highlighted by Csurka et al. [83], this is not a trivial matter. Different segmentation algorithms may perform optimally under different evaluation metrics, making it essential to align the chosen metric with the success criteria of the target application. In our case, the end application is the IARA path planning process, where a high-quality road grid map is defined by its ability to accurately preserve the shape and center of each lane. While commonly used metrics such as average class accuracy, precision, recall, F1 score, and mean IoU are valid and useful, they may not fully capture the overall effectiveness of the proposed system in achieving its specific purpose.

To evaluate the proposed system's ability to generate suitable RDDFs from road grid maps, we first set IARA to drive autonomously using an RDDF extracted from a manually annotated road grid map of UFES's ring road. In this experiment, IARA demonstrated autonomous navigation performance equivalent to that achieved using the previously employed, precisely annotated RDDF, which was used before the implementation described in this research. No human intervention was required.

In the final experiment, we used the ENet DNN to generate a road grid map of UFES's ring road, and IARA was set to drive autonomously using the RDDF extracted from this generated road grid map. Remarkably, IARA's navigation performance in this experiment was equivalent to that observed in the first experiment described above. Once again, no human intervention was needed. A video comparing these two experiments is available[15]. This version of the ENet DNN achieved an average accuracy of 83.7%. Consequently, the experiment with IARA demonstrates that this level of accuracy is sufficient to enable reliable autonomous operation with our approach, requiring no human intervention.

[15] Available at https://bit.ly/road_vid.

A noteworthy aspect of the new IARA System, which now relies on road grid maps, is that the high-level decisions IARA must make during navigation are primarily limited to (i) determining whether to change lanes (for instance, to overtake another vehicle), and (ii) choosing a direction at intersections (whether to turn right, left, or continue straight, as illustrated in Fig. 32). This simplifies the path planning process, allowing the use of online tools such as OpenStreetMap, Google Maps or Waze, similar to how humans plan routes when traveling to a new destination for the first time.

9. DISCUSSION

The purpose of this work is to advance the state of the art in addressing the specific research problem of automatically generating road grid maps and path plans for autonomous vehicles based on sensor data. Autonomous navigation is a complex task that requires a robust understanding of road structures, surrounding environments, and dynamic factors. This work aims to contribute a method that enhances the reliability and adaptability of autonomous vehicles across diverse scenarios.

The proposed method, as detailed in Section 8, demonstrates significant potential through several experiments. One notable strength is its robustness in conditions that challenge traditional approaches. For instance, compared to methods relying on frontal camera images, our approach offers the key advantage of being unaffected by low illumination at night – a limitation shared by the human vision system and many existing autonomous vehicle solutions. This robustness makes it well-suited for nighttime driving and poorly lit environments.

Another advantage lies in its ability to function effectively in environments where lane markings are poor or nonexistent. Traditional methods that rely on lane marking detection can fail in such scenarios, but our approach leverages broader contextual information about the roadway and its surroundings. This enables the system to generate path plans in a wider variety of conditions, such as rural roads, unpaved paths, or temporary roadways, which often lack clearly defined lane markings.

The DNN models used in this work have shown consistent performance when trained with an expanded dataset that includes diverse road scenarios. This highlights the importance of dataset quality and diversity in improving the adaptability of autonomous systems. To further enhance the method, the dataset could be extended to include a broader variety of road pavements, such as asphalt, concrete, pavers, clay, and gravel. Additionally, incorporating images from varied contexts – spanning urban and rural environments, with different types of sidewalks, buildings, and

vegetation – would improve the model's ability to generalize and adapt to new environments.

These enhancements are crucial because road and environmental diversity plays a significant role in real-world autonomous navigation. By accounting for differences in texture, lighting, and structural context, the DNN can better predict paths and generate accurate road grid maps, even in challenging or unfamiliar conditions.

While the method offers promising advancements, it is important to acknowledge its limitations. The accuracy of the proposed approach heavily depends on the quality and completeness of input sensor data. If the sensor data fails to capture critical road characteristics, such as ground and background textures or roadway context, the DNN may struggle to identify driving lanes.

Abnormal situations pose another significant challenge. For example, conditions such as flooded roads, roadsides covered with mud, or obstacles like debris and fallen trees can prevent the DNN model from properly identifying paths. However, it is worth noting that such extreme conditions often also disable human drivers' ability to navigate safely, rendering the affected roads inaccessible to traffic regardless of the navigation method used.

Addressing these limitations presents an opportunity for future research. One avenue is the integration of multi-modal sensor data, such as combining LiDAR, radar, and thermal imaging, to enhance the system's ability to handle edge cases like occluded or poorly visible roadways. Another direction is the development of adaptive learning techniques that enable the model to refine its understanding of road features over time, even in changing environments.

Finally, the inclusion of real-time feedback mechanisms could allow the system to dynamically adjust its behavior in response to environmental changes, improving its resilience to unexpected conditions. These advancements, combined with ongoing improvements in dataset diversity and model architecture, have the potential to push the boundaries of autonomous vehicle navigation, making it safer and more reliable across all operating conditions.

10. CONCLUSION

In this work, we introduced a novel approach for enabling autonomous vehicle operation on roads with degraded or missing horizontal markings. We employed various DNNs for semantic segmentation of road grid maps, derived from remission grid maps obtained via LiDAR data. These road grid maps are structured as square matrices, with each element representing a small section of region of real-world space and encoding semantic information about the road structure. Simple algorithms were developed to extract path plans (RDDFs) from these road grid maps, which were then employed to navigate an autonomous vehicle from its initial position to a desired destination.

We conducted experimental evaluations of selected DNNs using datasets we created by manually annotating tens of kilometers of roadways. These datasets have been made publicly available to support future research involving various types of autonomous vehicles. The results demonstrated a segmentation accuracy up to 94.7%, indicating that the DNNs were able to successfully infer road features, even identifying unmarked lanes in the ground truth.

We further validated this approach using the IARA autonomous vehicle over a 3.7 km stretch of urban roads, demonstrating performance comparable to that of a system reliant on manually generated RDDFs. This result indicates that a segmentation accuracy of 83.7% suffices for effective autonomous operation with our method.

For future work, there are several opportunities to enhance both the accuracy and generalization capacity of the DNN used for road grid map segmentation. The training datasets could be expanded with additional ground truth examples, and the DNN architectures could be further optimized or replaced with new models to improve performance.

Moreover, improving the methods for extracting viable path plans from road grid maps could facilitate the creation of more comprehensive road network graphs, which

would significantly benefit tasks related to route planning, path planning, and trajectory generation for autonomous vehicles.

REFERENCES

[1] Haselhoff, A., & Kummert, A. (2010, June). On visual crosswalk detection for driver assistance systems. In *2010 IEEE Intelligent Vehicles Symposium* (pp. 883-888). IEEE.

[2] Foucher, P., Sebsadji, Y., Tarel, J. P., Charbonnier, P., & Nicolle, P. (2011, October). Detection and recognition of urban road markings using images. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 1747-1752). IEEE.

[3] Berriel, R. F., de Aguiar, E., De Souza, A. F., & Oliveira-Santos, T. (2017). Ego-lane analysis system (elas): Dataset and algorithms. *Image and Vision Computing*, *68*, 64-75.

[4] Berriel, R. F., Rossi, F. S., de Souza, A. F., & Oliveira-Santos, T. (2017). Automatic large-scale data acquisition via crowdsourcing for crosswalk classification: A deep learning approach. *Computers & graphics*, *68*, 32-42.

[5] Hata, A., & Wolf, D. (2014, October). Road marking detection using LIDAR reflective intensity data and its application to vehicle localization. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (pp. 584-589). IEEE.

[6] Joshi, A., & James, M. R. (2015). Generation of accurate lane-level maps from coarse prior maps and lidar. *IEEE Intelligent Transportation Systems Magazine*, *7*(1), 19-29.

[7] Cui, D., Xue, J., & Zheng, N. (2015). Real-time global localization of robotic cars in lane level via lane marking detection and shape registration. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1039-1050.

[8] Gwon, G. P., Hur, W. S., Kim, S. W., & Seo, S. W. (2016). Generation of a precise and efficient lane-level road map for intelligent vehicle systems. *IEEE Transactions on Vehicular Technology*, *66*(6), 4517-4533.

[9] DARPA. (2005). DARPA Grand Challenge 2005: Route data definition file. *http://www.cajunbot.com/wiki/images/d/dc/2005_rddf_format_description.pdf*

[10] Carneiro, R. V., Nascimento, R. C., Guidolini, R., Cardoso, V. B., Oliveira-Santos, T., Badue, C., & De Souza, A. F. (2018, July). Mapping road lanes using laser remission and deep neural networks. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1-8). IEEE.

[11] Zakaria, N. J., Shapiai, M. I., Abd Ghani, R., Yassin, M. N. M., Ibrahim, M. Z., & Wahid, N. (2023). Lane detection in autonomous vehicles: A systematic review. *IEEE access*, *11*, 3729-3765.

[12] Li, J. (2023). Lane detection with deep learning: Methods and datasets. *Information Technology and Control*, *52*(2), 297-308.

[13] Jung, S., Youn, J., & Sull, S. (2015). Efficient lane detection based on spatiotemporal images. *IEEE transactions on intelligent transportation systems*, *17*(1), 289-295.

[14] Sultana, S., Ahmed, B., Paul, M., Islam, M. R., & Ahmad, S. (2023). Vision-based robust lane detection and tracking in challenging conditions. *IEEE Access*.

[15] Sang, I. C., & Norris, W. R. (2024). A Robust Lane Detection Algorithm Adaptable to Challenging Weather Conditions. *IEEE Access*.

[16] Bharadwaj, G. S., Gogula, A. R., Anduri, N. K., Pemma, T. G., & Kumar, S. V. (2023, June). Road Lane Line Detection for Autonomous Cars. In *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)* (pp. 1215-1219). IEEE.

[17] Yu, G., & Qiu, D. (2021, October). Research on Lane Detection Method of Intelligent Vehicle in Multi-road Condition. In *2021 China Automation Congress (CAC)* (pp. 2779-2783). IEEE.

[18] Alamsyah, S. A., Purwanto, D., & Attamimi, M. (2021, July). Lane Detection Using Edge Detection and Spatio-Temporal Incremental Clustering. In *2021 International Seminar on Intelligent Technology and Its Applications (ISITIA)* (pp. 364-369). IEEE.

[19] Banerjee, P., Thakur, K., & Banerjee, P. (2023, August). Lane Detection for Self-Driving Cars Using AI. In *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)* (pp. 1-5). IEEE.

[20] Padmaraja, V. P., Rohith, R., & Chittesh, S. (2023, June). Lane Detection using Image Processing for Autonomous Vehicles. In *2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)* (pp. 1-5). IEEE.

[21] Garg, M., Sehrawat, A., & Savaridassan, P. (2023, January). Vehicle Lane Detection for Accident Prevention and Smart Autodrive Using OpenCV. In *2023 International Conference on Computer Communication and Informatics (ICCCI)* (pp. 1-5). IEEE.

[22] Bin Abdul Razak, N., bin Mazlan, M. Z., Johari, J. B., Abdullah, S. A. B. C., & Mun, N. K. (2022, December). A Lane Detection Using Image Processing Technique for Two-Lane Road. In *2022 IEEE 10th Conference on Systems, Process & Control (ICSPC)* (pp. 214-219). IEEE.

[23] Lee, S., Kim, J., Shin Yoon, J., Shin, S., Bailo, O., Kim, N., ... & So Kweon, I. (2017). Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *Proceedings of the IEEE international conference on computer vision* (pp. 1947-1955).

[24] Yalabaka, S., Prasad, C. R., Sanjana, P., Lokesh, B., Shradha, T., & Samala, S. (2022, December). Lane Detection using Deep Learning Techniques. In *2022 8th International Conference on Signal Processing and Communication (ICSC)* (pp. 412-416). IEEE.

[25] Luo, Y., Zheng, C., Yan, X., Kun, T., Zheng, C., Cui, S., & Li, Z. (2023). Latr: 3d lane detection from monocular images with transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 7941-7952).

[26] Yao, Y., & Xiong, H. (2022, November). LaneFormer: An Efficient Transformer-based Network for Fast Lane Detection. In *2022 China Automation Congress (CAC)* (pp. 3111-3116). IEEE.

[27] Li, S., Wu, X., & Wu, Z. (2023). Efficient multi-lane detection based on large-kernel convolution and location. *IEEE Access*, *11*, 58125-58135.

[28] Alzubaidi, L. H., abed Almoussawi, Z., Jagadish, S., Madhavan, S., & Alassedi, Z. (2023, December). Autonomous Vehicle Lane Detection Using Hyperbolic Neural Network with Bi-Directional Long Short-Term Memory. In *2023 3rd International Conference on Mobile Networks and Wireless Communications (ICMNWC)* (pp. 1-6). IEEE.

[29] Islam, M. R., Siddique, T. A., Sakib, M. I. H., & Hossain, S. (2021, November). A convolutional neural network for end to end structural prediction and lane detection for

autonomous vehicle. In *2021 5th International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)* (pp. 1-6). IEEE.

[30] Rath, M. K., Swain, P. K., & Banerjee, S. (2022, October). An Optimised Deep Learning Approach of Lane Detection in Complex Indian Environment. In *2022 1st IEEE International Conference on Industrial Electronics: Developments & Applications (ICIDeA)* (pp. 1-5). IEEE.

[31] Moraes, G., Mozart, A., Azevedo, P., Piumbini, M., Cardoso, V. B., Oliveira-Santos, T., ... & Badue, C. (2020, July). Image-based real-time path generation using deep neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.

[32] Tabelini, L., Berriel, R., De Souza, A. F., Badue, C., & Oliveira-Santos, T. (2022, July). Lane marking detection and classification using spatial-temporal feature pooling. In *2022 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-7). IEEE.

[33] Wang, B., Frémont, V., & Rodríguez, S. A. (2014, June). Color-based road detection and its evaluation on the KITTI road benchmark. In *2014 IEEE Intelligent Vehicles Symposium Proceedings* (pp. 31-36). IEEE.

[34] Laddha, A., Kocamaz, M. K., Navarro-Serment, L. E., & Hebert, M. (2016, June). Map-supervised road detection. In *2016 IEEE Intelligent Vehicles Symposium (IV)* (pp. 118-123). IEEE.

[35] Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., & Urtasun, R. (2018, June). Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE intelligent vehicles symposium (IV)* (pp. 1013-1020). IEEE.

[36] Wang, Q., Gao, J., & Yuan, Y. (2017). Embedding structured contour and location prior in siamesed fully convolutional networks for road detection. *IEEE Transactions on Intelligent Transportation Systems*, *19*(1), 230-241.

[37] Zhang, R., Wu, Y., Gou, W., & Chen, J. (2021). RS-Lane: A Robust Lane Detection Method Based on ResNeSt and Self-Attention Distillation for Challenging Traffic Situations. *Journal of advanced transportation*, *2021*(1), 7544355.

[38] Zhou, X., Zhan, J., & Jiang, W. (2021, October). Work-in-Progress: Achieving Fast Lane Detection of Autonomous Driving by CNN Based Differentiation. In *2021 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)* (pp. 21-22). IEEE.

[39] Suresh, K., Gandhi, A. S., Soumya, M., & Dakshina, R. (2022, December). Smart Lane Detection Using Open CV And Image Processing. In *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)* (pp. 1-4). IEEE.

[40] Zhang, Y. M., Lin, S. W., Chou, T. H., Jhong, S. Y., & Chen, Y. Y. (2022, July). Robust lane detection via filter estimator and data augmentation. In *2022 IEEE International Conference on Consumer Electronics-Taiwan* (pp. 291-292). IEEE.

[41] Gautam, S., Mathuria, T., & Meena, S. (2022, June). Image Segmentation for Self-Driving Car. In *2022 2nd International Conference on Intelligent Technologies (CONIT)* (pp. 1-6). IEEE.

[42] Long, J., Yan, Z., Peng, L., & Li, T. (2021). The geometric attention-aware network for lane detection in complex road scenes. *Plos one*, *16*(7), e0254521.

[43] Han, X., Wang, H., Lu, J., & Zhao, C. (2017). Road detection based on the fusion of Lidar and image data. *International Journal of Advanced Robotic Systems*, *14*(6), 1729881417738102.

[44] Caltagirone, L., Scheidegger, S., Svensson, L., & Wahde, M. (2017, June). Fast LIDAR-based road detection using fully convolutional neural networks. In *2017 IEEE Intelligent Vehicles Symposium (iv)* (pp. 1019-1024). IEEE.

[45] Caltagirone, L., Bellone, M., Svensson, L., & Wahde, M. (2017, October). LIDAR-based driving path generation using fully convolutional neural networks. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)* (pp. 1-6). IEEE.

[46] Hernández, D. C., Filonenko, A., Seo, D., & Jo, K. H. (2015, March). Laser scanner based heading angle and distance estimation. In *2015 IEEE International Conference on Industrial Technology (ICIT)* (pp. 1718-1722). IEEE.

[47] Kim, D., Chung, T., & Yi, K. (2015, June). Lane map building and localization for automated driving using 2D laser rangefinder. In *2015 IEEE intelligent vehicles symposium (IV)* (pp. 680-685). IEEE.

[48] Reddy, B. R., Kumari, M. U., Baitha, T., Akshitha, S., & Anjali, M. (2023, November). Implementation of Lane Detection Algorithms for Autonomous Vehicle Using Lidar Point Cloud Data. In *2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)* (pp. 1-6). IEEE.

[49] Huang, J., Choudhury, P. K., Yin, S., & Zhu, L. (2021). Real-time road curb and lane detection for autonomous driving using LiDAR point clouds. *IEEE Access*, *9*, 144940-144951.

[50] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., ... & De Souza, A. F. (2021). Self-driving cars: A survey. *Expert systems with applications*, *165*, 113816.

[51] Montemerlo, M., Roy, N., & Thrun, S. (2003, October). Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453)* (Vol. 3, pp. 2436-2441). IEEE.

[52] Simmons, R. (2005). The inter-process communication (IPC) system. *http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc. html*.

[53] Mutz, F., Veronese, L. P., Oliveira-Santos, T., De Aguiar, E., Cheein, F. A. A., & De Souza, A. F. (2016). Large-scale mapping in complex field scenarios using an autonomous car. *Expert Systems with Applications*, *46*, 439-462.

[54] Levinson, J., & Thrun, S. (2010, May). Robust vehicle localization in urban environments using probabilistic maps. In *2010 IEEE international conference on robotics and automation* (pp. 4372-4378). IEEE.

[55] Veronese, L. D. P., Guivant, J., Cheein, F. A. A., Oliveira-Santos, T., Mutz, F., De Aguiar, E., ... & De Souza, A. F. (2016, November). A light-weight yet accurate localization system for autonomous cars in large-scale and complex environments. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 520-525). IEEE.

[56] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. The MIT Press.

[57] Mozart, A., Moraes, G., Guidolini, R., Cardoso, V. B., Oliveira-Santos, T., De Souza, A. F., & Badue, C. (2021). Path Planning in Unstructured Urban Environments for Self-driving Cars. In *18th International Conference on Informatics in Control, Automation and Robotics ICINCO* (pp. 290-300).

[58] Azevedo, P., Carneiro, R. V., De Souza, A. F., Oliveira-Santos, T., & Badue, C. S. (2023). A Planning Pipeline for Software Systems of Autonomous Vehicles. *Available at SSRN 4690921*.

[59] Cardoso, V., Oliveira, J., Teixeira, T., Badue, C., Mutz, F., Oliveira-Santos, T., ... & De Souza, A. F. (2017, May). A model-predictive motion planner for the iara autonomous car. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 225-230). IEEE.

[60] Guidolini, R., Badue, C., Berger, M., de Paula Veronese, L., & De Souza, A. F. (2016, November). A simple yet effective obstacle avoider for the IARA autonomous car. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 1914-1919). IEEE.

[61] Guidolini, R., De Souza, A. F., Mutz, F., & Badue, C. (2017, May). Neural-based model predictive control for tackling steering delays of autonomous cars. In *2017 International joint conference on neural networks (IJCNN)* (pp. 4324-4331). IEEE.

[62] Thrun, S., & Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, *25*(5-6), 403-429.

[63] Oliveira, J., Mutz, F., Forechi, A., Azevedo, P., Oliveira-Santos, T., De Souza, A. F., & Badue, C. (2021). Long-Term Map Maintenance in Complex Environments. In *Intelligent Systems: 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29–December 3, 2021, Proceedings, Part II 10* (pp. 146-161). Springer International Publishing.

[64] Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1724-1734).

[65] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18* (pp. 234-241). Springer International Publishing.

[66] Alsabhan, W., Alotaiby, T., & Dudin, B. (2022). Detecting buildings and non-buildings from satellite images using u-net. *Computational Intelligence and Neuroscience*, 2022(1), 4831223.

[67] Chowdhury, M. E., Rahman, T., Khandakar, A., Ayari, M. A., Khan, A. U., Khan, M. S., ... & Ali, S. H. M. (2021). Automatic and reliable leaf disease detection using deep learning techniques. *AgriEngineering*, 3(2), 294-312.

[68] Lau, S. L., Chong, E. K., Yang, X., Wang, X. (2020). Automated pavement crack segmentation using u-net-based convolutional neural network. *IEEE Access 2020*, 8, 114892–114899.

[69] Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*.

[70] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[71] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[72] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14* (pp. 630-645). Springer International Publishing.

[73] Badrinarayanan, V., Handa, A., & Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*.

[74] Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, *39*(12), 2481-2495.

[75] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (Vol. 30). Curran Associates, Inc.

[76] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations*.

[77] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 10012-10022).

[78] Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q., & Wang, M. (2022, October). Swin-unet: Unet-like pure transformer for medical image segmentation. In *European conference on computer vision* (pp. 205-218). Cham: Springer Nature Switzerland.

[79] He, X., Zhou, Y., Zhao, J., Zhang, D., Yao, R., & Xue, Y. (2022). Swin transformer embedding UNet for remote sensing image semantic segmentation. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1-15.

[80] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[81] Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The international journal of robotics research*, *29*(5), 485-501.

[82] Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 648-656).

[83] Csurka, G., Larlus, D., Perronnin, F., & Meylan, F. (2013, September). What is a good evaluation measure for semantic segmentation?. In *BMVC* (Vol. 27, No. 2013, pp. 10-5244).

Universidade Federal do Espírito Santo
Pró-Reitoria de Pesquisa e Pós-Graduação
Centro Tecnológico
Programa de Pós-Graduação em Informática (PPGI)

DECLARAÇÃO DE VERSÃO FINAL

Eu, então discente do Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Espírito Santo (UFES) e autor desta _____tese_____ (dissertação/tese), e o meu orientador, declaramos que a versão aqui apresentada incorpora as correções exigidas pelos membros examinadores da banca no dia da defesa.

gov.br Documento assinado digitalmente
RAPHAEL VIVACQUA CARNEIRO
Data: 01/12/2024 21:49:55-0300
Verifique em https://validar.iti.gov.br

Assinatura do autor
via Assinatura Eletrônica do gov.br
(assinar apenas depois de anexar este documento no PDF final da versão corrigida)

gov.br Documento assinado digitalmente
ALBERTO FERREIRA DE SOUZA
Data: 01/12/2024 17:35:25-0300
Verifique em https://validar.iti.gov.br

Assinatura do orientador
via Assinatura Eletrônica do gov.br
(assinar apenas depois de anexar este documento no PDF final da versão corrigida)