UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Thiago Gonçalves Cavalcante

Visual Global Localization Based on Deep Neural Networks for Self-Driving Cars

Vitória, ES
May 11, 2022

Thiago Gonçalves Cavalcante

Visual Global Localization Based on Deep Neural Networks for Self-Driving Cars

Master Thesis submitted to the *Programa de Pós-Graduação em Informática* of *Centro Tecnológ*ico of *Universidade Federal do Espírito Santo*, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: Prof. Dr. Claudine Badue
Co-advisor: Prof. Dr. Avelino Forechi

Vitória, ES
May 11, 2022

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

Cavalcante, Thiago Gonçalves, 1984-
C376v Visual global localization based on deep neural networks for self-driving cars / Thiago Gonçalves Cavalcante. - 2022.
58 f. : il.

Orientadora: Claudine Santos Badue.
Coorientador: Avelino Forechi.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Robótica. 2. Redes neurais (Computação). 3. Veículos autônomos. 4. Sistema de Posicionamento Global. 5. Visão por computador. I. Badue, Claudine Santos. II. Forechi, Avelino. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

Visual Global Localization Based on Deep Neural Networks for Self-Driving Cars

Thiago Gonçalves Cavalcante

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 11 de maio de 2022.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória/ES, 11 de maio de 2022

AGRADECIMENTOS

Agradeço à Universidade Federal do Espírito Santo (UFES) e ao Laboratório de Computação de Alto Desempenho (LCAD) pela oportunidade de cursar o Mestrado em Informática e desenvolver este estudo. Este trabalho foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código Financeiro 001; Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq); e Fundação de Pesquisa do Espírito Santo - Brasil (FAPES) - bolsas 75537958 e 84412844.

Agradeço a Deus por termos chegado até aqui com saúde em meio ao caos de 2020. Agradeço aos meus orientadores, Dra Claudine Santos Badue, Dr Avelino Forechi e Dr Alberto Ferreira De Souza, pela oportunidade e paciência durante essa jornada de aprendizado e amadurecimento. Agradeço também aos amigos e colegas que sempre dispostos a ajudar e apoiar nos momentos de dúvida.

Finalmente, gostaria de agradecer à minha família e principalmente à minha esposa Eliene, minha maior apoiadora.

RESUMO

Neste trabalho, apresentamos um sistema de localização global visual baseado em Redes Neurais Profundas (DNNs) para carros autônomos, denominado DeepVGL (Deep Visual Global Localization), que recebe imagens em tempo real de uma câmera frontal instalada no teto do carro e infere sua posição correspondente em coordenadas globais. Para tanto, o DeepVGL é treinado com pares de coordenadas e imagens associadas pertencentes a conjuntos de dados de veículos autônomos construídos com dados de sensores alinhados no tempo e no espaço por meio de um processo de Localização e Mapeamento Simultâneo (SLAM). Para avaliar o desempenho do DeepVGL, realizamos experimentos usando conjuntos de dados compostos por imagens de câmeras coletadas por diferentes carros autônomos em viagens feitas em longos intervalos de tempo (mais de 4 anos), incluindo mudanças significativas no ambiente, volume de tráfego e condições climáticas, bem como diferentes horas do dia e estações do ano. Também comparamos o DeepVGL com um sistema de localização global de última geração baseado em Redes Neurais Sem Peso (WNN). Por fim, executamos experimentos usando conjuntos de dados compostos por imagens da faixa LIDAR obtidas por um caminhão autônomo em viagens feitas em intervalos de tempo razoáveis (mais de 3 meses). Os resultados experimentais mostram que o DeepVGL pode estimar corretamente a localização global do carro autônomo em até 75% do tempo para uma precisão de 0,2 me até 96% do tempo para uma precisão de 5 m. Os resultados também mostram que o DeepVGL supera a WNN, que pode localizar corretamente o carro autônomo em até 76% do tempo para precisão de 0,2 m, mas apenas até 89% do tempo para precisão de 5 m. Por fim, os resultados mostram que o DeepVGL funciona melhor com imagens de alcance LIDAR do que com imagens de câmera, localizando o caminhão autônomo em até 95% do tempo para precisão de 0,2 m e 98% do tempo para precisão de 5 m.

ABSTRACT

In this work, we present a visual global localization system based on Deep Neural Networks (DNNs) for self-driving cars, called DeepVGL (Deep Visual Global Localization), which receives real-time images from a forward-facing camera installed on car’s roof and infers their corresponding position in global coordinates. To this end, DeepVGL is trained with pairs of coordinates and associated images belonging to datasets of autonomous vehicles built with sensor data aligned in time and space through a process of Simultaneous Localization And Mapping (SLAM). To assess the performance of DeepVGL, we carried out experiments using datasets composed of camera images collected by different self-driving cars on trips made over long time spans (over 4 years), thus including significant changes in the environment, traffic volume and weather conditions, as well as different times of the day and seasons of the year. We also compared DeepVGL with a state-of-the-art global localization system based on Weightless Neural Networks (WNN). Finally, we executed experiments using datasets composed of LIDAR range images obtained by a self-driving truck on trips made over reasonable time spans (over 3 months). The experimental results show that DeepVGL can correctly estimate the global localization of the self-driving car up to 75% of the time for an accuracy of 0.2 m and up to 96% of the time for an accuracy of 5 m. The results also show that DeepVGL outperforms WNN, which can correctly locate the self-driving car up to 76% of the time for 0.2 m accuracy, but only up to 89% of the time for 5 m accuracy. Finally, the results show that DeepVGL works better with LIDAR range images than camera images, locating the autonomous truck up to 95% of the time for 0.2 m accuracy and 98% of the time for 5 m accuracy.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF EQUATIONS

LIST OF ABREVATIONS AND ACRONYMS

ART	Autonomous Robotic Truck
CARMEN	Carnegie Mellon Robot Navigation Toolkit
CNN	Convolutional Neural Networks
DI	Departamento de Informática (Department of Informatics)
DNN	Deep Neural Networks
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IARA	Intelligent and Autonomous Robotic Automobile
ICP	Iterative closest point
IMU	Inertial Measurement Unit
LCAD	Laboratório De Computação de Alto Desempenho (High Performance Computing Laboratory)
LIDAR	Light Detection and Ranging
MAE	Maximum Allowed Error
OGM	Occupancy Grid Map
RTK	Real-Time Kinematic
SLAM	Simultaneous Localization And Mapping
UFES	Universidade Federal do Espírito Santo (Federal University of Espírito Santo)
WNN	Weightless Neural Networks

1 INTRODUCTION

Self-driving cars based on precise localization [1] operate in environments where, with properly constructed maps, they are able to carry on missions that involve travels between their current position to some specified goal position. For that, they require precise maps of the operating environment, with centimeter accuracy. In this context, it is essential that such self-driving cars can solve the problems of position tracking, global localization and kidnapped robot [2].

The position tracking problem is that of inferring to where the car has moved in the vicinity after the control commands are applied to the car during a short interval of time (typically a few milliseconds), considering that its current position is known to some degree of certainty (in the probabilistic sense). Currently, the position tracking problem is solved, for the most self-driving car systems, using probabilistic robotics algorithms, such as Kalman filters, particle filters, etc. [2] [3].

The global localization problem is that of inferring where the car is, considering the whole map. Since, in this case, the car can be anywhere in the map, this problem is significantly harder than that of position tracking. However, it can be simplified using, for example, Global Navigation Satellite System (GNSS) systems. Currently available hardware, firmware and software allow GNSS global localization with centimeter accuracy, which makes them a possible solution to the global localization problem (or even all localization problems). However, to work properly, the GNSS system must “see” a significant part of its satellite constellation. This is not possible in several relevant situations, such as inside tunnels, within urban canyons (roads surrounded by tall buildings), or in regions where trees or other elements of geography prevent the reception of signals from a sufficiently high number of satellites to allow precise localization. This makes GNSS systems an incomplete solution to the global localization problem.

The kidnapped robot problem is that of inferring where the car is on the entire map, after a global localization failure. The kidnapped robot problem is even more difficult than the global localization problem, as the self-driving car system may believe it knows where the car is, while it does not.

In this work, we present a visual global localization system, based on Deep Neural Networks (DNNs), for self-driving cars, named DeepVGL (Deep Visual Global Localization).

We examined the performance of DeepVGL in the Intelligent Autonomous Robotic Automobile (IARA, Figure 1), a self-driving car whose autonomy system is based on precise localization [1] – in IARA's autonomous system DeepVGL is new part of the Localizer subsystem (see Figure 1).

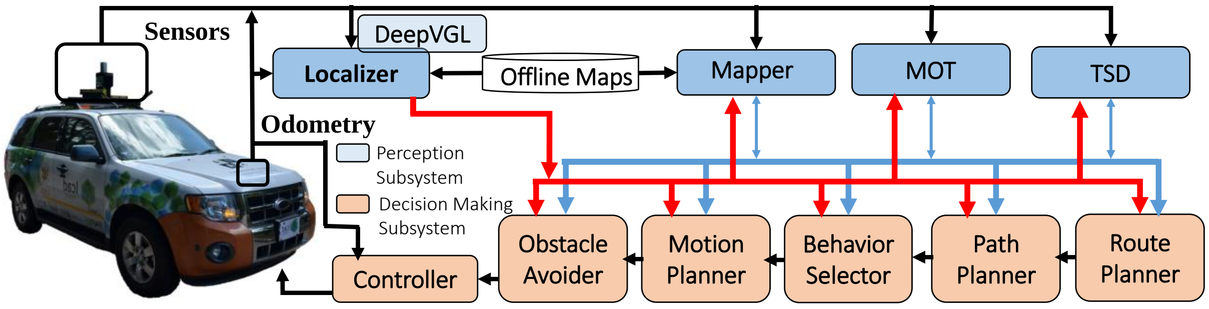

Figure 1. Intelligent Autonomous Robotic Automobile (IARA) and its autonomy system. In blue, perception subsystems, including DeepVGL (as a new part of the Localizer), the focus of this work. In orange, decision making subsystems. TSD denotes Traffic Signalization Detection and MOT, Moving Objects Tracking. Red arrows show the State of IARA, produced by its Localizer subsystem, and shared with most subsystems; blue arrows show IARA's internal representation of the environment, jointly produced by several perception subsystems and shared with most decision-making subsystems.

IARA is a research self-driving car based on a Ford Escape Hybrid adapted with a variety of sensors and processing units. Its autonomy system is composed of many subsystems, which include a Localizer [3], a Mapper [4], a Moving Obstacle Tracker [5], a Traffic Signalization Detector [6] [7], a Route Planner, a Path Planner, a Behavior Selector, a Motion Planner [8], an Obstacle Avoider [9], and a Controller [10], among other subsystems. For more details on IARA autonomy system, readers are referred to Badue et al. [1].

We also examined the performance of DeepVGL in the Autonomous Robotic Truck (ART, Figure 2), a research self-driving truck based on a Mercedes-Benz Atego 2430 retrofitted with sensors and computers. Its autonomy system is the same as the IARA system.

The approach presented here tries and solves the global localization problem building a neural map of images and associated coordinates. For that, during the process of building the Occupancy Grid Map (OGM) [2] [4] [11] [12] , we save images and their associated coordinates computed with respect to the map. After the mapping process, we build datasets of images and associated coordinates arranged to have these pairs (of coordinates and associated images) separated by a small offset (a few meters) and yet allow global localization with a precision that is within the capacity of the position tracking localization subsystem to later correct to the necessary level of accuracy (a few centimeters). DeepVGL, then, is trained with these datasets in order to be able to, given an image of the operating environment, infer the coordinates where the car was when the image was captured.

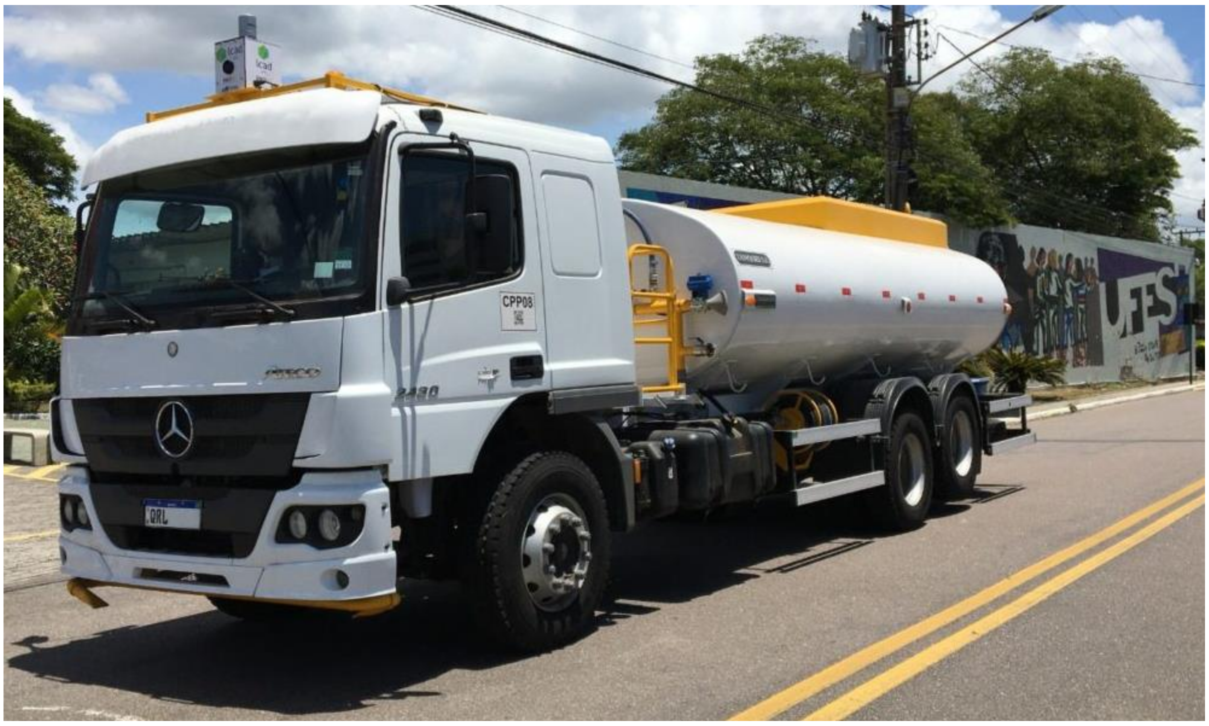

Figure 2. Autonomous Robotic Truck (ART)

The approach presented here solves the problem of the kidnapped robot as well. The position tracking subsystem is able to detect when the tracking has been lost. In such cases, DeepVGL is invoked and, after global localization, the position tracking resumes.

In summary, DeepVGL (Figure 3) is a system that allows training a DNN with datasets of images and associated coordinates, captured during mapping, and using this DNN to infer the coordinates as an output, given an image of the operating environment not present in the dataset. These inferred coordinates should be close enough to the place where the image was captured to allow the self-driving car's position tracking localization subsystem to start tracking the self-driving car position with the precision necessary to allow an accurate localization.

We used Volta-da-UFES and Apollo-DaoxiangLake datasets, among others, to evaluate the performance of DeepVGL. The Volta-da-UFES datasets contain 75,014 images and associated poses collected in 10 trips made over a period of 4 years along the 3.5 km of the ring road of the main campus of Universidade Federal do Espírito Santo (UFES), in Vitória, Brazil. The Apollo-DaoxiangLake datasets are composed of 143,636 images and associated poses collected in 8 trips made over a period of 14 weeks along the 11.8 km of the road adjacent to the Daoxiang Lake Park, in Beijing, China. The Volta-da-UFES includes significant changes in the campus infrastructure, traffic volume and weather conditions, while the Apollo-DaoxiangLake includes different times of the day, for example, noon, afternoon, sunset, as-well-as season changes (e.g., sunny and snowy days). Our experimental results show that DeepVGL obtained consistent results in both datasets, being able to correctly perform global localization of the self-driving car 75% of the time, if we consider an accuracy of 0.2 m, and 96% of the time, considering an accuracy of 5 m (5 m is well within the capabilities of the position tracking system to get to a centimeter-precision localization) for Volta-da-UFES, and 70% and 93% of the time, respectively, for Apollo-DaoxiangLake.

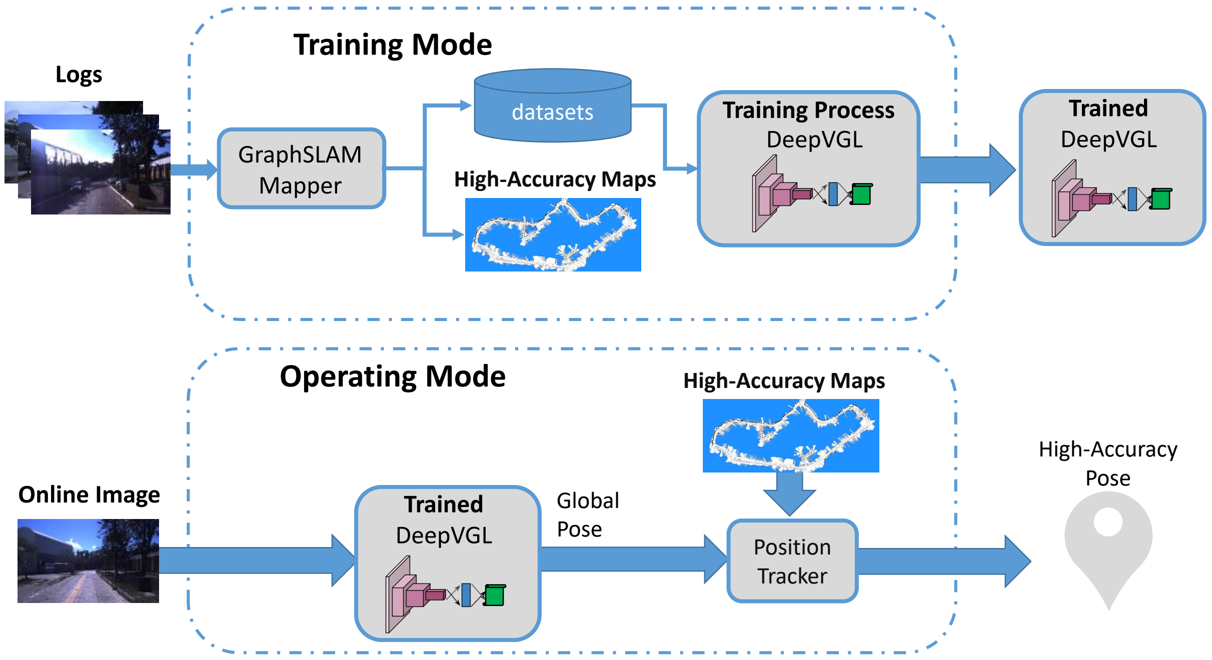

Figure 3. Overview of Deep Visual Global Localization (DeepVGL)

We also compared DeepVGL with a state-of-the-art global localization system based on Weightless Neural Networks (WNN) [13]. Our experimental results show that DeepVGL surpasses WNN, which can correctly locate the self-driving car 76% of the time for 0.2 m accuracy, but only 89% of the time for 5 m accuracy with the Volta-da-UFES, and 60% and 66% of the time, respectively, with the Apollo-DaoxiangLake.

Finally, we evaluated the performance of DeepVGL with two new datasets built from sensor data collected using ART, namely LCAD-ART-CAMERA and LCAD-ART-LIDAR. Both datasets were collected in 4 trips made over a period of 3 months along the 3.5 km of the ring road of the main campus of UFES. The LCAD-ART-CAMERA dataset is composed of 36,825 camera images and associated poses, and the LCAD-ART-LIDAR dataset contains 44,231 LIDAR range images. Our experimental results show that DeepVGL obtained consistently better results with the LCAD-ART-LIDAR dataset (LIDAR range images) than with the LCAD-ART-CAMERA dataset (camera images), being able to correctly locate the self-driving truck 78% of the time for 0.2 m accuracy and 93% of the time for 5 m accuracy with the LCAD-ART-CAMERA, and 95% and 98% of the time, respectively, with the LCAD-ART-LIDAR.

Please see the video available at https://github.com/LCAD-UFES/DeepVGL to better visualize DeepVGL operation and performance in real world scenarios.

1.1 Motivation

This work is part of the research on self-driving cars carried out by the High-Performance Computing Laboratory (*Laboratório de Computação de Alto Desempenho* - LCAD) of the Informatics Department (*Departamento de Informática* - DI) of the Federal University of Espírito Santo (*Universidade Federal do Espírito Santo* - UFES). LCAD developed the IARA (Intelligent Autonomous Robotic Automobile) and the ART (Autonomous Robotic Truck), self-driving vehicles that have been used as research platforms in several projects.

To navigate autonomously, IARA and ART need to know their position in the world with centimeter accuracy, a task that is currently performed using a Real-Time Kinematic (RTK) Global Navigation Satellite System (GNSS). However, on many occasions, the use of GNSSs is limited, as the signal reception can suffer interference due to urban canyons, treetops and other elements of the environment. Under these conditions, the correct functioning of the self-driving vehicle is impaired. In this context, the main motivation of this work is to provide a reliable visual global localization system, even in GNSS-denied environments, that tries and solves the three main localization problems: global localization, position tracking and kidnapped robot.

In the context of self-driving cars, the most successful visual global location systems presented in the literature use 2D and 3D features extracted from camera images and LIDAR data to provide state-of-the-art performance [16][17][18][19][20][21]. Nevertheless, these systems fail on large datasets or do not provide the same performance when considering accuracies smaller than 5 m. The system proposed by Forechi et al. [13], based on WNNs, uses only camera images, but outperforms these approaches with large datasets and small accuracies. The system proposed in this work, based on DNNs and camera images, is equivalent to the one presented by Forechi et al. [13] if we consider smaller accuracies (up to 0.2 m), but surpasses it considering larger accuracies (up to 5 m).

1.2 Objectives

The objective of this work is to propose a reliable visual global localization system based on DNNs, which is compatible with IARA’s autonomy system and operates in real-time. It is also an objective of this work to evaluate the performance of the proposed system through tests in real world scenarios.

1.3 Contribution

The main contributions of this work are:

1) Proposal of a reliable visual global localization system based on DNNs, named DeepVGL, which is compatible with IARA’s autonomy system and operates in real-time;
2) Development of DeepVGL and its integration to the IARA’s autonomy system;
3) Construction of two new datasets from camera and LIDAR range images collected from real-world self-driving vehicles;
4) Evaluation of the performance of DeepVGL through a series of experiments using sensor data collected from real-world autonomous vehicles.

The results mentioned above were presented in the following scientific paper:

T. G. Cavalcante, A. Forechi, T. Oliveira-Santos, A. F. De Souza, C. Badue, “Visual Global Localization Based on Deep Neural Netwoks for Self-Driving Cars”, 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1-7.

1.4 Organization

The text of this dissertation is organized as follows. After this introduction, in Chapter 2, we present related works. In Chapter 3, we detail DeepVGL. In Chapter 4, we present the experimental methodology used to evaluate DeepVGL and, in Chapter 5, the experimental results. Finally, in Chapter 6, we present our conclusions and directions for future work.

2 RELATED WORK

The visual global localization problem for self-driving cars is addressed in the literature as a classification task [16] [17], which uses discrete locations as labels, or as a regression task [18] [19] [20], which regresses the relative distance vector between two or more images, or both as a classification and a regression task [13].

2.1 Localization as a Classification Task

Oertel et al. [17] employed a Siamese-like network architecture. They used an image as input on one branch and a voxel grid (3D input) derived from the structure-from-motion point cloud on the other. Their network outperformed state-of-the-art descriptors by up to 90%, especially at low descriptor dimensionalities. However, the maximum permitted error is still high (from 5 to 20 m) compared to our results (up to 5 m).

2.2 Localization as a Regression Task

Instead of using multiple inputs, T. Xie et al. [20] proposed a multi-task CNN for robot relocalization that can simultaneously perform pose regression and scene recognition. Despite outperforming PoseNet [21] in indoor and outdoor environments, it struggles in outdoor environments trying to distinguish very similar places. Finally, when comparing the size of the environments and the localization accuracy, the robustness achieved by our method in outdoor environments is far superior.

Schonberger et al. [19] proposed a different multi-task CNN based on a joint 3D geometric and semantic understanding of the world. They employed a generative model for learning the descriptor, trained on semantic scene completion as an auxiliary task. During on-line operation, they used these learned descriptors to create matches between a query and a database map. The matches are then used to estimate an alignment between the two maps, which defines a pose estimate for the query. They demonstrated reliable localization on two large-scale localization datasets under extreme viewpoint, illumination, and geometry changes. Nevertheless, those datasets do not fit our requirements for a proper comparison. The KITTI dataset [22], for instance, fails to enclose multiple trips of the same road. The NCLT dataset [23] does it, but, unfortunately, its images are not well aligned with 3D LIDAR scans, necessary to build the ground-truth.

2.3 Localization as a Classification and Regression Task

Forechi et al. [13] proposed a global localization system for self-driving cars composed of two components: a place recognition subsystem based on WNNs; and a visual localization subsystem based on Convolutional Neural Networks (CNNs). The place recognition component receives an online image and outputs the most similar (recollected) image learned in the training phase and the associated pose. The visual localization component receives the online image and the recollected image and returns the relative camera pose represented by these images. To estimate the relative camera pose between the recollected and the online images, the CNN is trained with the two images as input and a relative pose vector as output. Experimental results show that the proposed system correctly localizes a self-driving car 90% of the time with a mean error of 1.20 m. We compared DeepVGL with the place recognition component using the Volta-da-UFES datasets and verified that DeepVGL has a performance similar to this component, providing correct global localization 75% of the time for 0.2 m accuracy, while the component does that 76% of the time, but outperforms the component 96% of the time for 5 m accuracy, while the component achieves that only 89% of the time in this situation. In the Apollo-DaoxiangLake datasets, DeepVGL outperforms the place recognition component in all scenarios, achieving 70% global localization performance for 0.2 m, while the component achieves 60%, and 93% global localization performance for 5 m, while the component achieves only 66% in this scenario.

2.4 Localization Based on Deep Neural Networks

Self-driving cars usually require images to be captured sequentially, as part of their trajectory, either to identify a global or relative pose of the vehicle. In this context, several approaches similar to ours achieve state-of-the-art results, but fail to deal with large datasets [18] or in accuracy with spacing sampling less than 5m [16]. Our approach proved to be effective in both situations, obtaining good results on large datasets with competitive precision.

3 DEEP VISUAL GLOBAL LOCALIZATION (DEEPVGL)

DeepVGL is part of the Localizer subsystem of IARA (see Figure 1). Before the addition of DeepVGL, IARA used only a Global Navigation Satellite System (GNSS) receiver for global localization. DeepVGL performs the same function of IARA's GNSS, and it is automatically employed in cases where the GNSS signal is absent or has a too low precision for global localization. For convenience, we describe DeepVGL here in the context of the IARA's autonomy system; however, DeepVGL can be used in any self-driving car.

DeepVGL is composed of two parts, a Deep Neural Network and a table (Figure 4), and works in two modes, training mode and operating mode (Figure 3).

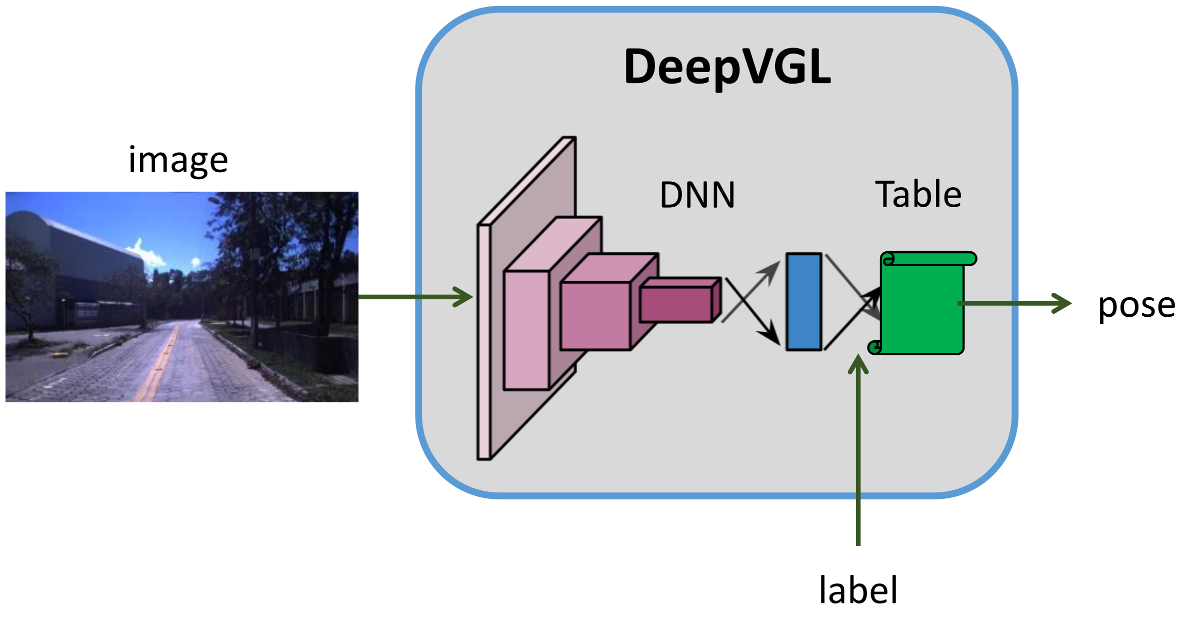

Figure 4. Deep Visual Global Localization (DeepVGL)

3.1 DeepVGL Training Mode

In the training mode, the DNN of DeepVGL is trained with image-label pairs from datasets built during the process of construction of the Occupancy Grid Map (OGM). The occupancy grid mapping process is responsible for generating high-accuracy maps of the operating environment (in IARA's case, map cells have 20 × 20 cm size), which are used for position tracking and other purposes. Figure 5 shows an example of such maps.

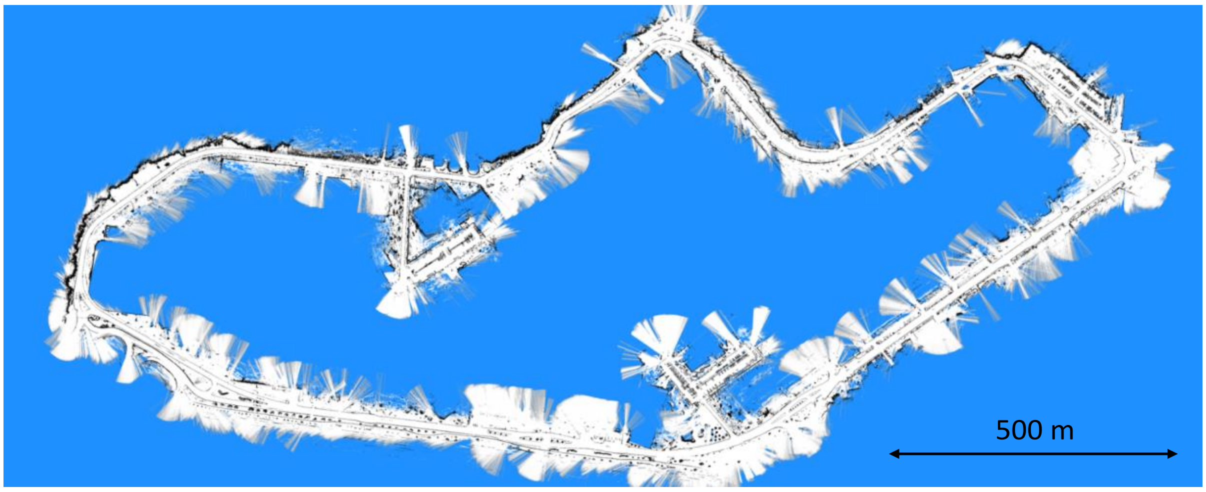

Figure 5. Occupancy Grid Map (OGM)

OGMs for self-driving cars are typically computed using a process known as simultaneous localization and mapping (SLAM [2]). There are several algorithms for SLAM and, in this work, we use GraphSLAM [2] [4] [11]. In the data acquisition processes required for occupancy grid mapping, we save all sensors data in files we call logs. Our version of GraphSLAM produces as output a precise pose (*x*, *y* and *yaw*) for each data sample captured in the data acquisition processes of each one of the IARA's sensors. In this way, we are able to build datasets of image-pose pairs of the operating environment. We use a unique integer as a label for each pose – the Table (Figure 4) of DeepVGL stores the label-pose pairs – and train the DNN of DeepVGL to output the labels of the images of the datasets. By training the DNN with several images for each relevant pose of the operating environment, we allow the DNN to generalize on the appearance (captured by the images) of each relevant pose of the environment and produce the correct label for never seen images of relevant positions of the operating environment.

3.2 DeepVGL Operating Mode

In the operating mode (Figure 3), the DNN of DeepVGL receives online images and outputs labels inferred from these images. The DNN also outputs its confidence in the association of the inferred label to the online image. DeepVGL uses the inferred label to get its unique pose from the Table – the Global Pose (Figure 3). The Global Pose and associated confidence are used by the IARA's Localizer (Figure 1) for global localization when required.

3.3 DeepVGL's DNN Architecture

The DNN employed by DeepVGL to infer labels associated with environment poses is a Convolutional Neural Network (CNN). The CNN architecture adopted is a modified version of YOLOv2 [24], where the input size was modified to 448 × 448 pixels (Figure 6) or 512 × 384 pixels, for camera and LIDAR range images, respectively, and the output size of the softmax layer was set according to the maximum number of unique poses in the datasets. The layers of YOLOv2 responsible for object detection were not used and, therefore, removed. YOLOv2's softmax layer computes a probability associated with each possible output label, i.e., a probability (or confidence) associated with each possible pose in our case.

Figure 6 shows the architecture of the DeepVGL DNN, used for camera images, with an input size of 448 × 448 pixels and an output of 658 classes. The total of classes is a direct relationship between the total distance of the analyzed path and the defined spacing (between key poses). This process is discussed in Section 8.1. For a path of 3290 meters and a spacing of 5 meters between poses, 658 key poses are generated.

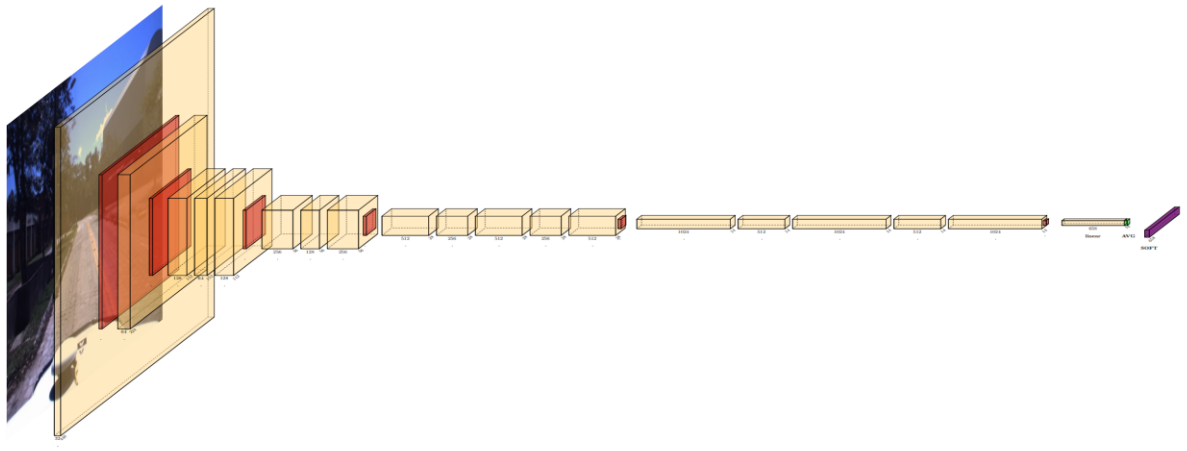

Figure 6. DeepVGL DNN architecture with an input size of 448 × 448 pixels and an output of 658 classes. All convolutional layers have stride 1. All maxpooling have stride 2 and size 2 × 2, which reduces the dimensions of the previous layer's output by half. The 1st convolutional layer has 32 filters, is 3 × 3 pixels in size, receives an image of 448 × 448 pixels and is followed by a 2 × 2 maxpooling, which reduces the output to 224 × 224 pixels. The 2nd convolutional layer has 64 filters and is 3 × 3 pixels in size. The 2nd maxpooling reduces the output to 112 × 112 pixels. Convolutional layers 3 and 5 has 128 filters and are 3 × 3 pixels in size. Convolutional layer 4 has 64 filters and is 1 × 1 pixel in size. The 3rd maxpooling reduces the output to 56 × 56 pixels. Convolutional layers 6 and 8 have 256 filters and are 3 × 3 pixels in size. Convolutional layer 7 has 138 filters and is size 1 × 1 pixel in size. The 4th maxpooling reduces the output to 28 × 28 pixels. Convolutional layers 9, 11 and 13 have 512 filters and are size 3 × 3 pixels in size. Convolutional layers 10 and 12 have 256 filters and are 1 × 1 pixel in size. The 5th maxpooling reduces the output to 14 × 14 pixels. Convolutional layers 14, 16 and 18 have 1024 filters and are 3 × 3 pixels in size. Convolutional layers 15 and 17 have 512 filters and are size 1 × 1 pixels in size. Convolutional layer 19 must have the number of filters equal to the number of classes (in this case, 658), size of 1 × 1 pixel and output of 14 × 14 pixels. The penultimate layer is an average pool with output equal to the number of classes, followed by a softmax with 658 classes in this example.

The larger the number of distinct poses, the larger the size of the softmax layer of DeepVGL's DNN. This somehow imposes a practical limit on the number of possible poses that DeepVGL can infer and, therefore, in the size of the operating environment in which it can be employed. Nevertheless, DeepVGL can be extended to operate in very large environments by using a technique commonly used for occupancy grid mapping called map tiling [4].

In very large environments, the whole OGM is sliced into square tiles that are loaded when the self-driving car approaches the border of each tile. In such a scenario, the weights of the DNN and the table of DeepVGL associated with each map tile are loaded together with the OGM tile.

4 EXPERIMENTAL METHODOLOGY

In this chapter, we present the methodology used in the experiments conducted to evaluate the performance of the proposed visual global localization system. In Section 4.1, we describe the infrastructure of IARA and ART. In Section 4.2, we present the datasets. Finally, in Section 4.3, we describe the metric used to evaluate the DeepVGL's performance and, in Section 4.4, the robotics operating system (CARMEN).

4.1 Autonomous Vehicle Platforms

IARA is a self-driving car developed by the computational intelligence research group of LCAD at UFES, in Brazil. IARA is based on a Ford Escape Hybrid, which was modified to allow electronic control of steering, throttle, brakes, gears and several signalization items; and to provide the car odometry for the IARA's autonomy system, and power supply for computers and sensors. Its main computer is a Dell Precision R5500 with two Xeon X5690 six-core 3.4 GHz processors and one NVIDIA TITAN Xp. Its sensors include one Velodyne HDL 32-E LIDAR, one Trimble RTK GNSS, one Xsens MTi IMU and one Bumblebee XB3 stereo camera.

ART is based on a Mercedes-Benz Atego 2430, which was also adapted to enable electronic vehicle control, and to provide the vehicle odometry for the ART's autonomy system, which is basically the same as the IARA, and power for computers and sensors. Its main computer is a Dell XPS 8940 with an Intel Core i7-10700 octa-core 2.9 GHz to 4.8 GHz processor and on NVIDIA GeForce RTX 2060 Super. Its sensors include one Velodyne HDL 32-E LIDAR, one Trimble RTK GNSS, one MPU-9250 Nine-Axis MEMS MotionTracking Device and one Intelbras VIP 1020 B G2 camera.

4.2 Datasets

4.2.1 Volta-da-UFES Datasets

IARA's Bumblebee XB3 stereo camera captures stereo-image pairs at 16 frames per second (FPS), where each image of each pair has 640×480 pixel size. The closest-in-time pose (*x*, *y*, *yaw*) computed by IARA's Localizer subsystem is associated to each stereo image. To compute these poses, the Localizer uses IARA's odometry, IMU and LIDAR – please see details in Veronese et al. [3]. The accuracy of IARA's Localizer subsystem with respect to the OGM, while performing position tracking, is 0.21 m, for maps with cells of 0.2×0.2 meter size [3]. We used a single reference map to build the Volta-da-UFES datasets. This reference map was produced using our implementation of the GraphSLAM algorithm [4] [11] while using a log not employed for building the datasets.

The Volta-da-UFES datasets consist of 10 text files and images associated to each line of each one of the files, grouped in directories, one for each file. Each file line contains the name of the associated image file, its label, 6D pose (*x*, *y*, *z*, *roll*, *pitch*, and *yaw*) and timestamp. There can be several lines with the same label, since the labels correspond to 3D poses (*x*, *y*, *yaw*) 5 m apart from each other along the ring road on the reference map – images receive the label of the nearest position.

Each file corresponds to a log associated with a trip along the whole ring road of UFES main campus (Figure 7), except datasets 4 and 6 (see Table 1). In each trip, IARA was driven at a maximum speed of 60 km/h. A complete trip is approximately 3.5 km long. Image and pose data were collected synchronously, producing approximately 75,000 image-pose pairs. Table 1 summarizes the Volta-da-UFES datasets.

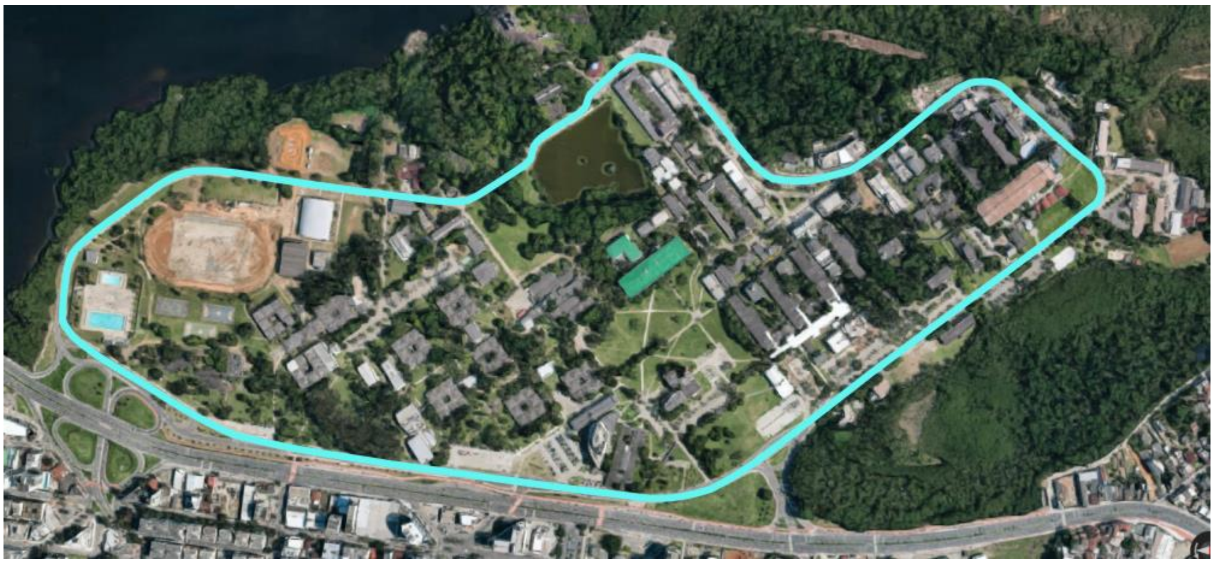

Figure 7. Ring road of UFES main campus of approximately 3.5 km long. Image taken from Google Maps.

Table 1: Volta-da-UFES datasets

Dataset	**Date (mm-dd-yyyy)**	**Sampling Spacing**		
		< 1 m	**1 m**	**5 m**
UFES-LAP-01	08-25-2016	7,165	2,742	651
UFES-LAP-02	08-25-2016	6,939	2,599	651
UFES-LAP-03	08-25-2016	6,404	2,538	651
UFES-LAP-04	08-30-2016	1,706	723	169
UFES-LAP-05	10-21-2016	9,404	2,853	651
UFES-LAP-06	01-19-2017	1,804	701	171
UFES-LAP-07	12-05-2017	17,911	5,703	651
UFES-LAP-08	01-12-2018	8,220	2,828	651
UFES-LAP-09	01-12-2018	7,714	2,843	645
UFES-LAP-10	10-03-2019	7,747	2,868	651
Total		**75,014**	**26,398**	**5,542**

Table 1 contains 5 columns. The first two are the dataset name and the date of the log from which it was built. It is important to note that the difference in time between the datasets reaches more than 3 years. This long-time span poses a challenge, as during such a long time period there were significant changes in the campus infrastructure. These changes include different traffic and weather conditions, time of the day, road and buildings maintenances, occlusions, and viewpoint entanglements. The last 3 columns of Table 1 show the number of images (spacing < 1 m) in each dataset, the number of images if they are taking at a spacing of about 1 m and the number of images if taking at a spacing of 5 m. The Volta-da-UFES datasets are available at https://github.com/LCAD-UFES/DeepVGL.

The Volta-da-UFES datasets were divided into training, validation and test sets: the training set consists of datasets 1, 2, 3, 5, 7, 8 and 9; the validation set of datasets 4 and 6; and the test set of dataset 10. The 1 m spacing was used for training, validation, and test. So, altogether, the training set has 44,212 images (we used both images of each stereo pair), the validation set has 2,848 images (we used both images of each stereo pair), and the test set has 2,868 images (we used only the left image of each stereo pair).

4.2.2 Apollo-DaoxiangLake Datasets

To examine the performance of DeepVGL in different environment and weather conditions, we used datasets made available by Apollo-DaoxiangLake, which contain pairs of images and poses from 8 different dates collected throughout the year of 2019 [25]. The images-pose pairs were collected along a course of approximately 11.8 km in the “Daoxianghu Natural Wetland Park” region (see Figure 8).

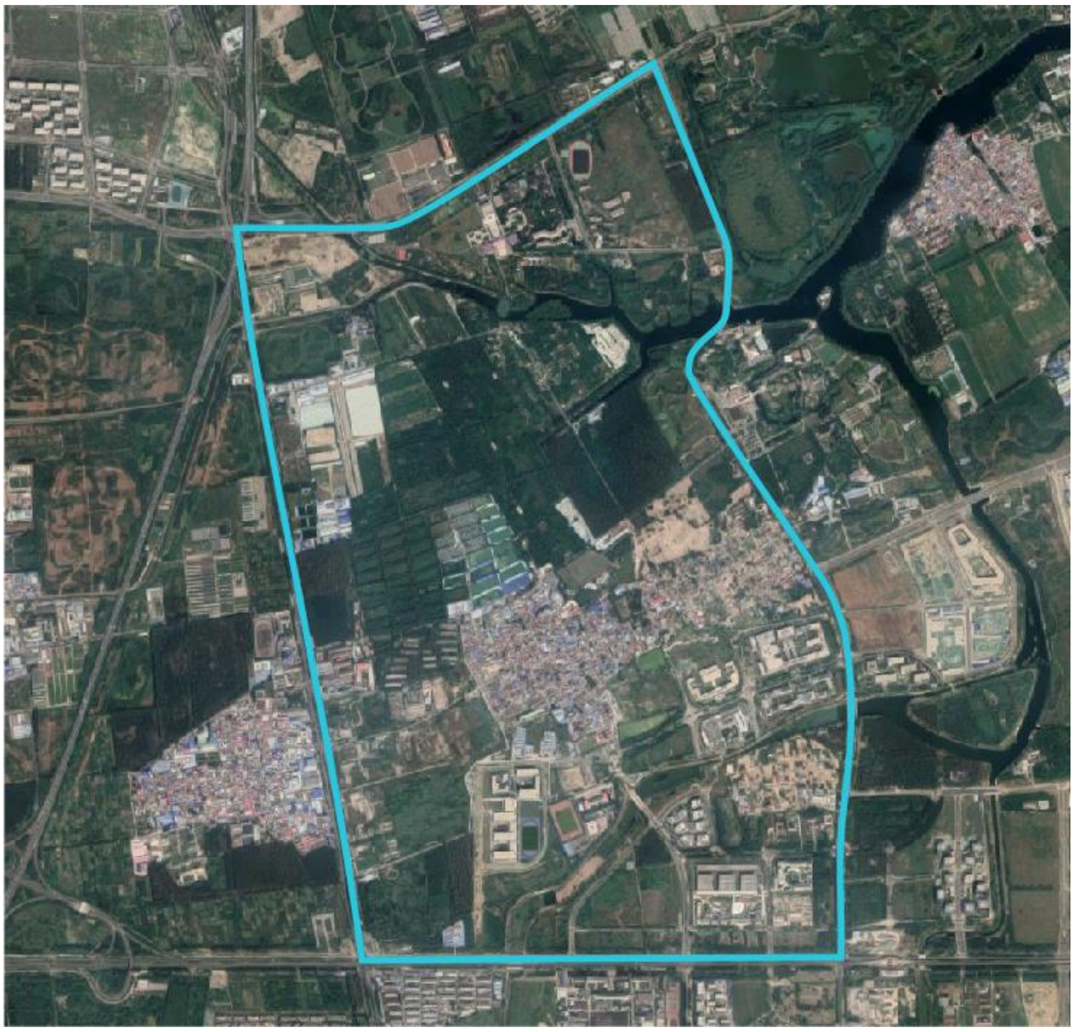

Figure 8. Daoxianghu Narutal Wetland Park, China, where the Apollo-DaoxiangLake datasets were collected. Image taken from Google Maps.

The Apollo-DaoxiangLake datasets consist of 8 text files and images associated to each line of each one of the files, grouped in directories, one for each file. Each file line has the same format of the Volta-da-UFES datasets. Their system collected images from 3 cameras while running along the course in both directions, but we used images taking only from the front camera while running in the counterclockwise direction. Table 2 summarizes these datasets. BAIDU_1 to BAIDU_6 was used for training, and BAIDU_7 and BAIDU_8 for testing and validation, respectively.

Table 2: Apollo-DaoxiangLake datasets

Dataset	Date (mm-dd-yyyy)	Sampling Spacing			
		< 1 m Both Directions	< 1 m Counter Clockwise	1 m	5 m
BAIDU_1	2019-09-18 14:33	26,850	7,636	5,377	1,990
BAIDU_2	2019-09-24 12:48	39,975	21,000	13,809	1,990
BAIDU_3	2019-10-14 14:25	22,955	11,000	7,807	1,990
BAIDU_4	2019-10-21 16:21	25,308	13,000	8,900	1,990
BAIDU_5	2019-10-25 10:47	23,515	12,000	8,248	1,990
BAIDU_6	2019-11-30 11:28	50,301	35,000	22,201	1,990
BAIDU_7	2019-12-16 12:33	41,189	21,000	13,552	1,990
BAIDU_8	2019-12-25 15:36	42,705	23,000	14,851	1,990
Total		**245,948**	**143,636**	**94,745**	**15,920**

4.2.3 LCAD-ART-CAMERA Dataset

ART's Intelbras VIP 1020 B G2 camera captures images at 16 FPS, where each image has 640×480 pixel size. As in IARA, the closest-in-time pose (*x*, *y*, *yaw*) computed by ART's Localizer subsystem [3] is associated to each image.

The LCAD-ART-CAMERA datasets consist of 4 text files and images associated to each line of each of the files, grouped in directories, one for each file. Each file line contains the name of the associated image file, its label, 6D pose (*x, y, z, roll, pitch and yaw*) and timestamp. The labels correspond to positions (*x*, *y*, *yaw*) 3 m apart from each other along the ring road on the reference map and the images are labeled with the closest position.

Each file corresponds to a log associated with a trip along the ring road of the main campus of UFES (Figure 7). In each trip, ART was driven at a maximum speed of 60 km/h. Image and pose data were collected synchronously, producing about 37,000 image-pose pairs. Table 3 summarizes the Volta-da-UFES datasets.

Table 3: LCAD-ART-CAMERA datasets

Dataset	Date (yyyy-mm-dd)	Number of Images	Number of Images Used
ART_1_C	2021-01-20	15,766	10,914
ART_2_C	2021-01-31	9,067	7,315
ART_3_C	2021-02-12	1,940	782
ART_4_C	2021-03-05	10,052	6,893
Total		**36,825**	**25,904**

Table 3 contains 4 columns. The first two are the dataset name and the date of the log from which it was built. The last 2 columns of Table 3 show the number of images captured in each dataset and the number of images used. During the process of building the datasets, some images are discarded – please see details in Section 8.1.

The LCAD-ART-CAMERA datasets were divided into training, validation and test sets: the datasets 1, 2 e 3 were merged and split into training and validation sets, and the dataset 4 was used as the test set. The training set has 17,109 images, the validation set has 1,902 images and the test set has 6,893 images.

4.2.4 LCAD-ART-LIDAR Dataset

ART's Velodyne HDL 32-E LIDAR produces point clouds with 32 lines and approximately 1,084 columns at a rate of 20 Hz. As with the camera, the closest-in-time pose (x, y, yaw) computed by ART's Localizer subsystem [3] is associated with each point cloud. The LIDAR range images are generated from the point clouds, where each image has 640×480 pixel size.

To generate LIDAR range images, LIDAR distance measurements are stored in a 32x1084 matrix. The values are converted to a color scale and the matrix is then converted to an image. Each channel in the RBG image is related to a range measured by LIDAR, where R starts from 0 to 25 m, G from 25 to 50 and B from 50 to 75 (limit of our scale). All readings above 75m produce white pixels and all readings equal to zero are treated the same. This produces an image with pixels ranging from dark red, through orange and yellow, to white. In figure X, we see an example of the resulting LIDAR range image in contrast to the camera image associated with the same scene. For details on the process of generating images from point clouds, please see Section 8.2.

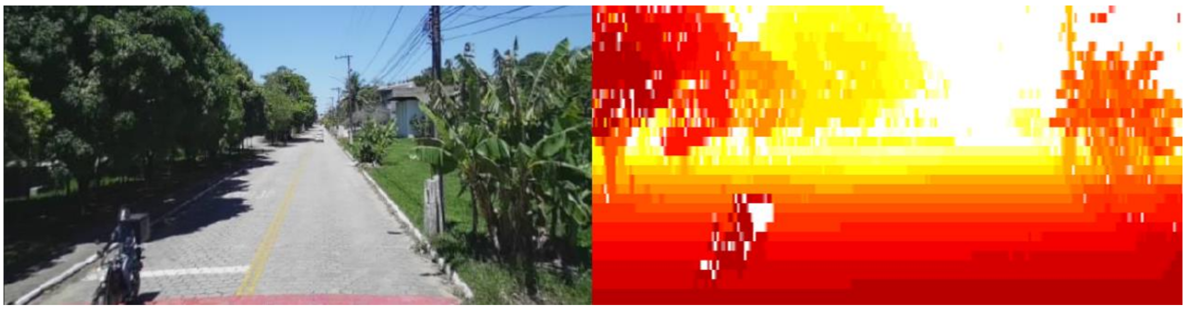

Figure 9. Comparison between camera image and LIDAR range image. The color scale in the image on the right represents LIDAR distance measurements that range from values close to zero (dark red) to 75 m (white) which is the defined limit of the scale.

The LCAD-ART-LIDAR datasets consist of 4 text files and LIDAR range images associated to each line of each one of the files. Each file line contains the name of the associated image file, its label, 6D pose (*x*, *y*, *z*, *roll*, *pitch and yaw*) and timestamp. The labels correspond to positions (*x*, *y*, *yaw*) 3 m apart from each other along the ring road on the reference map and the images are labeled with the closest position.

The files correspond to logs associated with trips along the ring road of the main campus of UFES (Figure 7), the same trips used to collect the LCAD-ART-CAMERA datasets (Table 3). LIDAR and pose data were collected synchronously, producing approximately 44,321 pairs of LIDAR range image and pose. Table 4 summarizes the LCAD-ART-LIDAR datasets. This table has the same format of Table 3.

Table 4: LCAD-ART-LIDAR datasets

Dataset	**Date (yyyy-mm-dd)**	**Number of LIDAR Range Images**	**Number of LIDAR Range Images Used**
ART_1_L	2021-01-20	15,794	11,289
ART_2_L	2021-01-31	12,400	10,564
ART_3_L	2021-02-12	2,599	1,438
ART_4_L	2021-03-05	13,438	10,109
Total		**44,231**	**33,400**

The LCAD-ART-LIDAR datasets were divided into training, validation and test sets: the datasets 1, 2 e 3 were merged and split into training and validation sets, and the dataset 4 was used as the test set. The training set has 20,962 LIDAR range images, the validation set has 2,329 range images and the test set has 10,109 range images.

4.3 Evaluation Metrics

To evaluate the performance of DeepVGL, we used two metrics: Accuracy and Maximum Allowed Error. The Accuracy metric is the percentage of times the DNN outputs the correct label when making inferences on the validation or test sets. The Maximum Allowed Error (MAE) refers to the number of poses one needs to go forward or backward in the sequence of poses to find the expected pose for a given DNN input image.

4.4 Carmen

The subsystems of the IARA autonomy system (Figure 1) are implemented as one or more software modules using the Carnegie Mellon Robot Navigation Toolkit (CARMEN). CARMEN is a modular collection of software for controlling mobile robot. It was designed to provide basic navigation primitives, and to support several robots and sensors. It was created by Carnegie Mellon University (http://carmen.sourceforge.net/).

In CARMEN, communication between modules is done through the Inter Process Communication (IPC), where each module can publish messages and/or subscribe to messages of interest to the module and receive them. This exchange of messages can even happen between different computers using TCP/IP. The messages are sent through a publication to the Central server, which is responsible for receiving the messages and passing them on to the modules that signed the specific message. Modules that publish messages are called Publishers, while modules that receive messages are called Subscribers. A module can be both at the same time.

Since 2011, CARMEN has been extended and maintained by LCAD at UFES, and made available to the public at https://github.com/LCAD-UFES/carmen_lcad. CARMEN-LCAD can be integrated into any type of vehicle platform. For that, it is necessary to adapt the platform to enable electronic vehicle control; provide the vehicle odometry to the autonomy system, and power for computers and sensors; and install sensors, e.g., LIDAR, RTK GNSS, IMU and cameras. CARMEN-LCAD has now been tested in cars, electric pods, trucks and even in planes. For those tests, sensors were encapsulated in a box (called the sensor box) that was fixed to the roof of the vehicles.

5 EXPERIMENTAL RESULTS

In this chapter, we present the results of the experiments we carried out to evaluate the DeepVGL performance. In Section 5.1, we present the functions and parameters used for training the DeepVGL DNN. In Section 5.2, we present the results obtained with Volta-da-UFES and Apollo-DaoxiangLake datasets. Finally, in Section 5.3, we present the results obtained with the LCAD-ART-CAMERA and LCAD-ART-LIDAR datasets.

5.1 DNN Training Functions and Parameters

For Volta-da-UFES and Apollo-DaoxiangLake datasets, we trained the DeepVGL DNN using the original Darknet YOLOv2. For LCAD-ART-CAMERA and LCAD-ART-LIDAR datasets, we trained the DeepVGL DNN as in the original Darknet YOLOv2, but using the Python implementation [26], which provides more options for loss function and data augmentation – instead of the Multi-Part Loss [27] employed by the original Darknet YOLOv2, we used the Cross-Entropy Loss as the loss function.

We employed an initial learning rate of 0.001 and used three distinct hyperparameter configurations: poly, steps and sgdr [27]. The remaining YOLOv2 training hyperparameters have not been modified. The DNN was pre-trained with the ImageNet dataset [28].

All experiments with camera images were performed on a workstation with 64 GB of RAM and an Nvidia Titan V GPU with 12 GB of video memory. It took 120 epochs to complete a training and the average time was around 48h. The experiments in python using images from the LIDAR line were performed on a personal computer with 16 GB of RAM and an NVIDIA GeForce GTX 1050 Ti GPU. Generally, it took 4 to 12 seasons for training and the average time was around 12 hours. In both configurations, tests can be run in real time.

5.2 Results with Volta-da-UFES and Apollo-DaoxiangLake Datasets

5.2.1 DNN Accuracy

We trained the DNN of DeepVGL for 130 epochs with Volta-da-UFES training set using the 3 different optimization methods. Figure 10 presents the results for Volta-da-UFES validation set. As Figure 10 shows, the choice of the optimization method does not affect the DNN Accuracy much (step show a small advantage). The highest achieved Accuracy was ~76%.

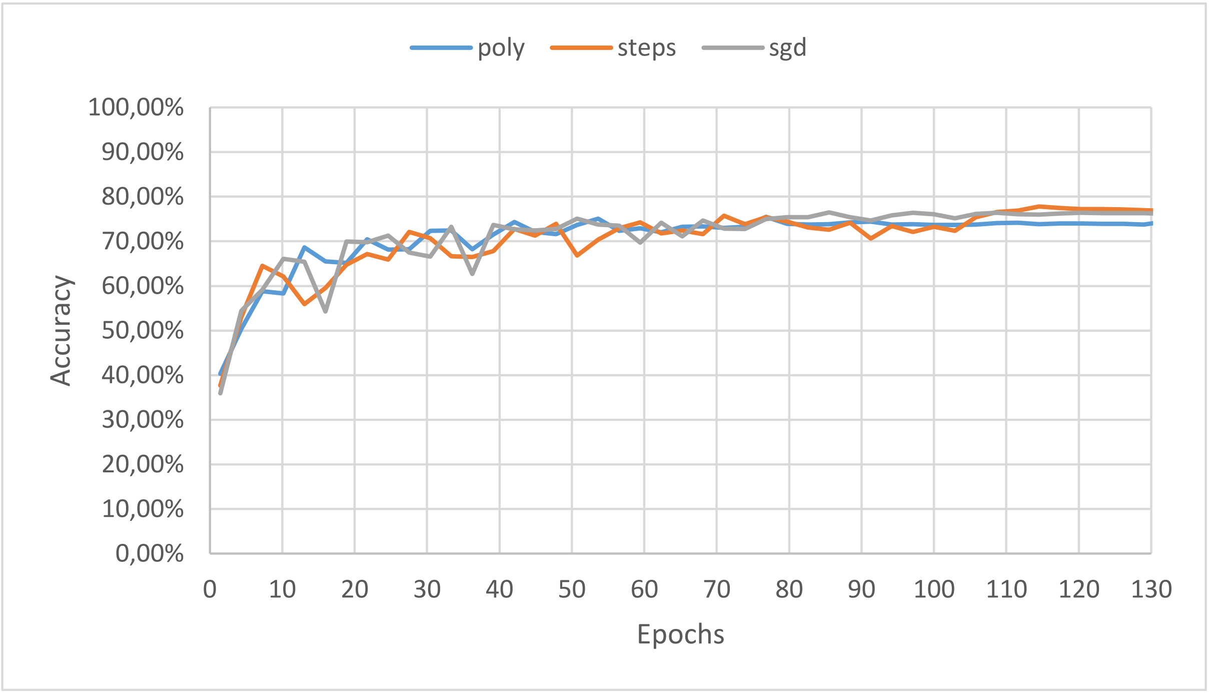

Figure 10. Accuracy of DeepVGL for the Volta-da-UFES validation set

We trained the DNN of DeepVGL for 230 epochs with Apollo-DaoxiangLake training set using the step optimization method. 0 presents the results for the Apollo-DaoxiangLake validation set. As 0 shows, highest achieved Accuracy was ~81%.

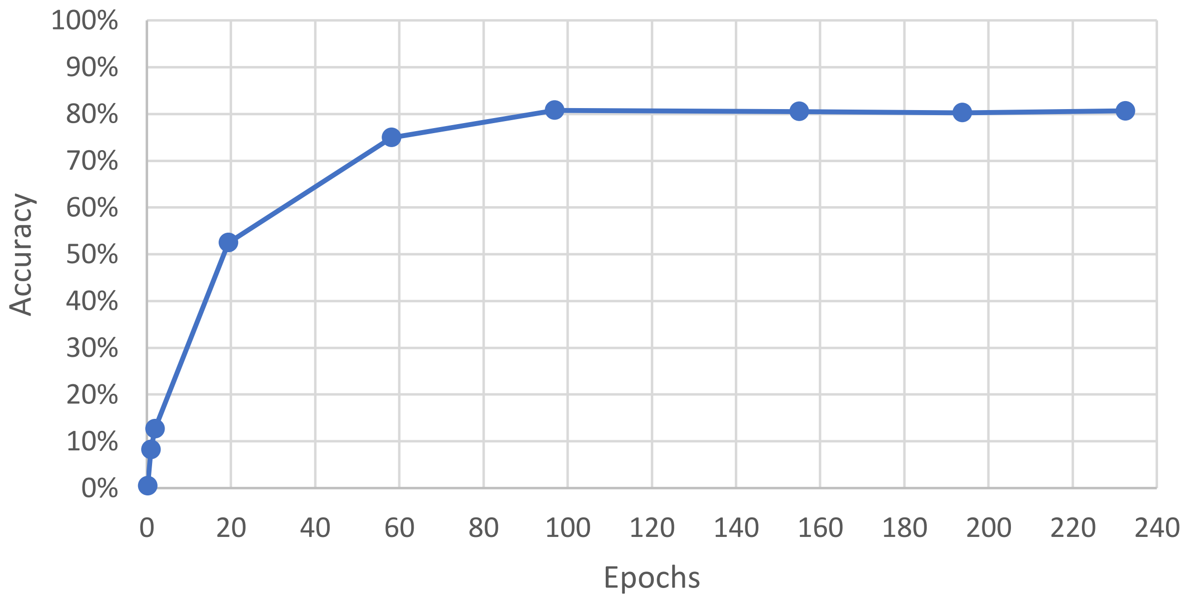

Figure 11. Accuracy of DeepVGL for the Apollo-DaoxiangLake validation set

5.2.2 DeepVGL MAE Performance

DeepVGL performance for different MAE levels in the Volta-da-UFES validation and test sets are shown in Table 5. In this table, the first column presents the MAE (number of sequential poses one is allowed to move forward or backward to find the correct pose of an input image), while the second and third columns, the Accuracy considering the MAE for the validation and test sets. As Table 5 shows, the Accuracy increases significantly as MAE goes from 0 to 2. In our tests, the IARA's Localizer was always able to start proper position tracking after a correct label inference for a MAE of up to 2. Note that the Accuracies for the validation and test sets are similar.

Table 5: DeepVGL Volta-da-UFES Accuracy considering MAE, given by the number of sequential poses, or labels, one is allowed to move forward or backward to find the correct label of an input image

MAE	Accuracy for each MAE (%)	
	Volta-da-UFES Validation Set	Volta-da-UFES Test Set
0	73	75
1	96	95
2	99	97

Table 6 presents the Accuracy of DeepVGL for different MAE levels considering the Apollo-DaoxiangLake validation and test sets. It has the same format of Table 6. As Table 7 shows, the results for Apollo-DaoxiangLake are similar to the results of Volta-da-UFES, although somewhat worse. This is to be expected, since the number images per label of the Apollo-DaoxiangLake datasets (33 images/label) is smaller than the Volta-da-UFES datasets (67 images/label).

Table 6: DeepVGL Apollo-DaoxiangLake accuracy considering MAE

MAE	Accuracy (%)	
	Apollo-DaoxiangLake Validation Set	Apollo-DaoxiangLake Test Set
0	81	71
1	97	90
2	97	93

5.2.3 Comparison with a WNN

We compared our results with that of Forechi et al. [13], who employed a Weightless Neural Network (WNN) approach to solve the same global localization problem. To our knowledge, this is the best previous result for this problem in the context of self-driving cars. Figure 12 presents the result of this comparison.

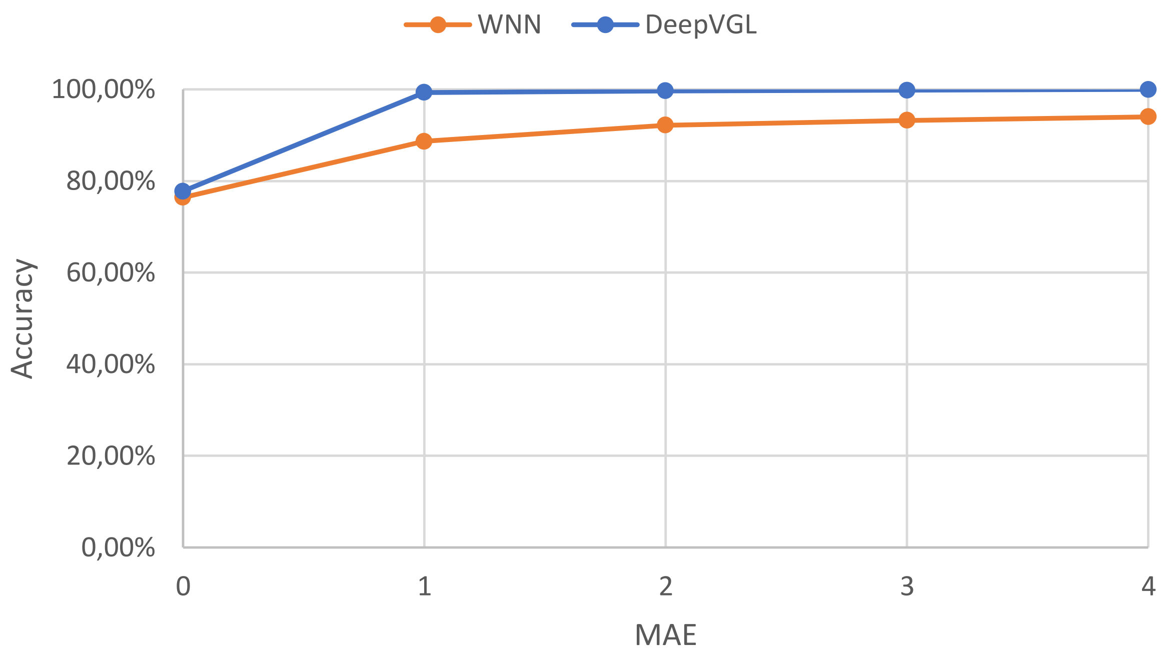

Figure 12. Accuracy of DeepVGL and WNN for 5 values of Maximum Allowed Error (MAE) in frames with Volta-da-UFES datasets

In Figure 12, the curve in orange represents the results of the WNN approach and the curve in blue the results of DeepVGL. To produce the results of this figure, we trained and tested the WNN system with the same Volta-da-UFES datasets presented in this work. As Figure 12 shows, DeepVGL presents a performance significantly higher than the WNN approach in the Volta-da-UFES datasets.

Figure 13 has the same format of Figure 12 and presents the results of WNN and DeepVGL for the Apollo-DaoxiangLake datasets. Again, as Figure 13, DeepVGL shows a performance significantly higher than the WNN approach.

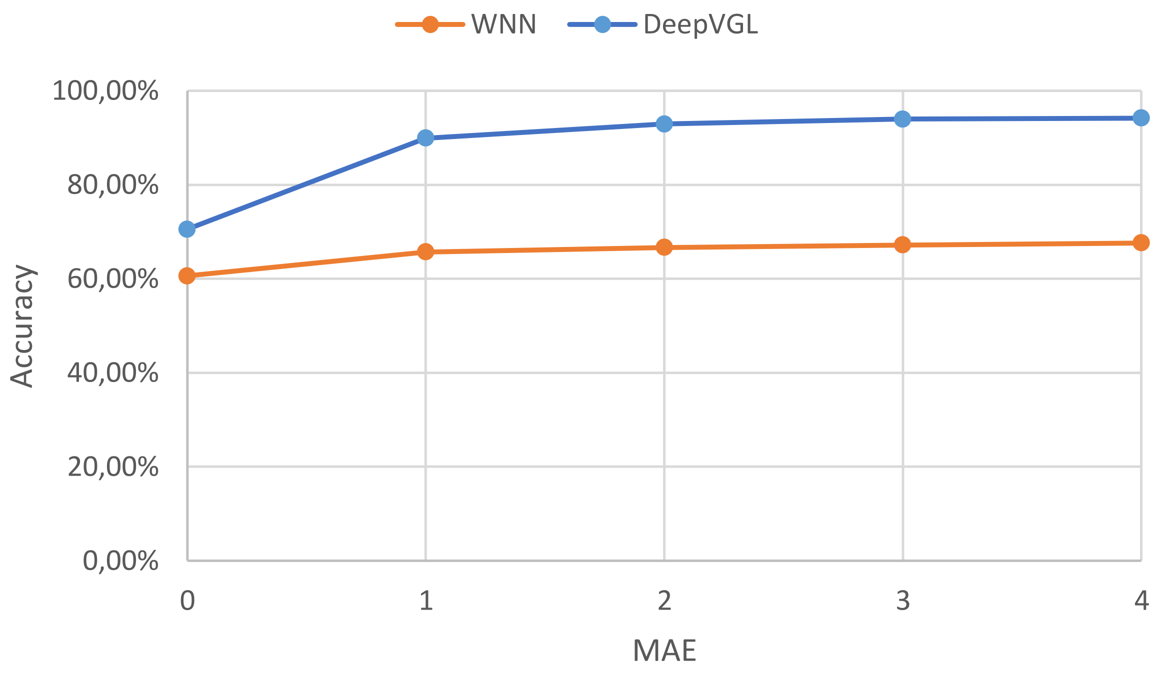

Figure 13. Accuracy of DeepVGL and WNN for 5 values of Maximum Allowed Error (MAE) in frames with Apollo-DaoxiangLake datasets

5.3 Results with the LCAD-ART-CAMERA and LCAD-ART-LIDAR Datasets

Table 7 shows the DeepVGL performance for two MAE levels in the LCAD-ART-CAMERA validation and test sets. As Table 7 shows, the results for LCAD-ART-CAMERA are similar to the results of Volta-da-UFES (Table 5). However, the Accuracies for the test set are worse than the Accuracies for the validation set. This is to be expected, as the number images per label of the for LCAD-ART-CAMERA datasets (39 images/label) is smaller than the Volta-da-UFES datasets (67 images/label).

Table 7: DeepVGL LCAD-ART-CAMERA accuracy considering MAE

MAE	Accuracy (%)	
	LCAD-ART-CAMERA Validation Set	LCAD-ART-CAMERA Test Set
0	88	78
1	98	93

Table 8 presents the DeepVGL performance for two MAE levels in the LCAD-ART-LIDAR validation and test sets. As Table 8 shows, the results for LCAD-ART-LIDAR are significantly better than the results of LCAD-ART-CAMERA. For a MAE level of 0 (zero), the Accuracy for the LCAD-ART-LIDAR test set is 95 (Table 8, line 2, column 3), while for the LCAD-CAMERA-LIDAR test set is 78 (Table 7, line 2, column 3), i.e., approximately 20% higher; and, for a MAE level of 1 (one), the Accuracy for the LCAD-ART-LIDAR test set is 98 (Table 8, line 3, column 3), while for the LCAD-CAMERA-LIDAR test set is 93 (Table 7, line 3, column 3), i.e., about 5% higher.

Table 8: DeepVGL LCAD-ART-LIDAR accuracy considering MAE

MAE	**Accuracy (%)**	
	LCAD-ART-LIDAR Validation Set	**LCAD-ART-LIDAR Test Set**
0	98	95
1	99	98

The reason for these results is that camera images provide visual information about the characteristics of objects that make up the environment around the car; colors, shapes and how they relate can be easily seen. However, range images extracted from LIDAR data incorporate distance information of objects around the car, which is invariant to changes in lighting and lateral displacement, for example. (In our experiments, we did not evaluate changes in the lateral displacement of objects, but only changes in lighting.) In addition, LIDAR range images also incorporate a wider field of view. Therefore, LIDAR range images aggregate information that compensates for their low resolution compared to camera images and provide better results in estimating global localization.

6 CONCLUSIONS AND FUTURE WORK

In this chapter, we present the conclusions and directions for future work. In Section 6.1, we present the conclusions based on the experimental results of this research work. In Section 6.2, we present directions for future work.

6.1 Conclusions

In this work, we presented DeepVGL, a robust and efficient solution for the global localization problem using images. DeepVGL provides a relevant alternative to situations where GNSS receivers cannot work due to satellite occlusion (within tunnels, urban canyons, garages, etc.). This makes it highly applicable in the context of self-driving cars.

We evaluated the performance of DeepVGL using datasets composed of tens of thousands of camera images and associated poses, which were obtained with experimental self-driving cars. The datasets include significant changes in building infrastructure, traffic volume and weather conditions, as well as different times of the day and season of the year. Our results showed that DeepVGL can correctly estimate the global localization of the self-driving car up to 75% of the time with an accuracy of 0.2 m and up to 96% of the time with an accuracy of 5 m, which is well within the capabilities of the position tracking system to get to a centimeter-precision localization.

We also compared DeepVGL with a state-of-the-art global localization system based on WNN [13]. Our results showed that DeepVGL outperforms WNN, which can correctly locate the self-driving car up to 76% of the time for 0.2 m accuracy, but only up to 89% of the time for 5 m accuracy.

Finally, we tested the performance of DeepVGL with datasets composed of tens of thousands of LIDAR range images and associated poses, which were obtained with an experimental self-driving truck. Our results showed that DeepVGL obtained consistently better results with LIDAR range images than with camera images, being able to correctly locate the self-driving truck up to 95% of the time for 0.2 m accuracy and 98% of the time for 5 m accuracy. These results showed that LIDAR range images – which incorporate distance information from objects around the car and a wider field of view – provide superior results than camera images – which incorporate colors and shapes – on the global localization of self-driving cars.

6.2 Future Work

As directions for future works, we plan to enlarge the input of the DeepVGL DNN and reexamine its performance considering this change when using camera images. Another direction for future research is to increase the number of convolutional layers of the DNN architecture. Synthetic data augmentation through the random insertion of black or white blocks, and mosaics with images of movable objects common to the road environment, is also a promising alternative for investigation in the direction of improving accuracy and generalization. Moreover, we intend to extend the results obtained with LIDAR images and apply ICP registration algorithm between learned and live point clouds to increase the subsystem accuracy and prediction frequency. Furthermore, we plan to analyze the performance of DeepVGL DNN in a self-driving car using weights learned from another car and evaluate the possibility of eliminating the need to collect sensor data whenever the vehicle platform changes. Finally, we intend to compare DeepVGL with some other approaches that solve the problem of visual global localization in the context of self-driving cars.

7 REFERENCES

[1] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. F. R. Jesus, R. F. Berriel, T. M. Paixão, F. Mutz, T. Oliveira-Santos and A. F. De Souza, “Self-Driving Cars: A Survey”, Expert Systems with Applications, vol. 165, pp. 113816, 2021.

[2] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics, MIT Press, 2006.

[3] L. de P. Veronese, J. Guivant, F. A. A. Cheein, T. Oliveira-Santos, F. Mutz, E. de Aguiar, C. Badue and A. F. De Souza, “A Light-Weight Yet Accurate Localization System for Autonomous Cars in Large- Scale and Complex Environments”, International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, pp. 520–525, 2016.

[4] F. Mutz, L. P. Veronese, T. Oliveira-Santos, E. de Aguiar, F. A. Auat Cheein and A. F. De Souza, “Large-Scale Mapping in Complex Field Scenarios Using an Autonomous Car”, Expert Systems with Applications (ESWA), vol. 46, pp. 439–462, 2016.

[5] R. Sarcinelli, R. Guidolini, V. B. Cardoso, T. M. Paixão, R. F. Berriel, P. Azevedo, A. F. De Souza, C. Badue and T. Oliveira-Santos, “Handling Pedestrians in Self-Driving Cars Using Image Tracking and Alternative Path Generation with Frenét Frames”, Computers & Graphics, vol. 84, pp. 173-184, 2019.

[6] L. C. Possatti, R. Guidolini, V. B. Cardoso, R. F. Berriel, T. M. Paixão, C. Badue, A. F. De Souza and T. Oliveira-Santos, “Traffic Light Recognition Using Deep Learning and Prior Maps for Autonomous Cars”, International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2019, pp. 1-8.

[7] L. T. Torres, T. M. Paixão, R. F. Berriel, A. F. De Souza, C. Badue, N. Sebe and T. Oliveira-Santos, “Effortless Deep Training for Traffic Sign Detection Using Templates and Arbitrary Natural Images”, International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2019, pp. 1-7.

[8] V. Cardoso, J. Oliveira, T. Teixeira, C. Badue, F. Mutz, T. Oliveira-Santos, L. Veronese and A. F. De Souza, “A Model-Predictive Motion Planner for the IARA Autonomous Car”, International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 225-230.

[9] R. Guidolini, C. Badue, M. Berger, L. P. Veronese and A. F. De Souza, “A Simple Yet Effective Obstacle Avoider for the IARA Autonomous Car”, International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, 2016, pp. 1914-1919.

[10] R. Guidolini, A. F. De Souza, F. Mutz and C. Badue, “Neural-Based Model Predictive Control for Tackling Steering Delays of Autonomous Cars”, International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 4324-4331.

[11] F. Mutz, T. Oliveira-Santos, A. Forechi, K. S. Komati, C. Badue, F. M. G. França and A. F. De Souza, “What Is the Best Grid-Map for Self-Driving Cars Localization? An Evaluation under Diverse Types of Illumination, Traffic, and Environment”, Expert Systems with Applications, vol. 179, pp. 115077, 2021.

[12] J. Oliveira et al., “Long-Term Map Maintenance in Complex Environments”, Brazilian Conference on Intelligent Systems (BRACIS), Lecture Notes in Computer Science, vol 13074, pp.146-161, 2021.

[13] A. Forechi, T. Oliveira-Santos, C. Badue and A. F. De Souza, “Visual Global Localization with a Hybrid WNN-CNN Approach”, International Joint Conference on Neural Networks (IJCNN), 2018.

[14] A. Forechi, A. F. De Souza, C. Badue and T. Oliveira-Santos, “Sequential Appearance-Based Global Localization Using an Ensemble of KNN-DTW Classifiers”, 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 2782-2789.

[15] A. Forechi, A. F. A. De Souza, J. J. de Oliveira Neto, E. d. E. Aguiar, C. Badue, A. Garcez, O.-S. Thiago, A. d’Avila Garcez and T. Oliveira-Santos, “Fat-Fast VG-RAM WNN: A High Performance Approach”, Neurocomputing, vol. 183, no. Weightless Neural Systems, pp. 56–69, 2015

[16] E. Brosh et al., “Accurate Visual Localization for Automotive Applications”, Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 1307–1316.

[17] A. Oertel, T. Cieslewski and D. Scaramuzza, “Augmenting Visual Place Recognition With Structural Cues”, Robotics and Automation Letters (RA-Letters), vol. 5, no. 4, pp. 5534–5541, 2020.

[18] T. Naseer and W. Burgard, “Deep Regression for Monocular Camera-based 6-DoF Global Localization in Outdoor Environments”, Autonomous Robots, pp. 1525–1530, 2017.

[19] J. L. Schonberger, M. Pollefeys, A. Geiger and T. Sattler, “Semantic Visual Localization”, Conference on Computer Vision and Pattern Recognition, pp. 6896–6906, 2018.

[20] T. Xie, K. Wang, R. Li and X. Tang, “Visual Robot Relocalization Based on Multi-Task CNN and Image-Similarity Strategy”, Sensors, vol. 20, no. 23, pp. 1–20, 2020.

[21] A. Kendall, M. Grimes and R. Cipolla, “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization”, International Conference on Computer Vision (ICCV), 2015, pp. 2938–2946.

[22] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Vision Meets Robotics: The KITTI Dataset”, The International Journal of Robotics Research, vol. 32, no. 11, pp. 1231–1237, 2013.

[23] N. Carlevaris-Bianco, A. K. Ushani and R. M. Eustice, “University of Michigan North Campus Long-Term Vision and LIDAR Dataset”, The International Journal of Robotics Research, vol. 35, no. 9, pp. 1023-1035, 2016.

[24] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger”, Conference of Computer Vision and Pattern Recognition (CVPR), 2017, pp. 6517–6525.

[25] Y. Zhou et al., “DA4AD: End-to-End Deep Attention-Based Visual Localization for Autonomous Driving”, European Conference on Computer Vision (ECCV), 2020, pp. 271–289.

[26] Tanguy Ophoff, “Lightnet: Building Blocks to Recreate Darknet Networks in Pytorch”, https://gitlab.com/EAVISE/lightnet, 2018.

[27] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, 2016 IEEE Conference on Computer

Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

8 APPENDIX: DATASET BUILDING PROCESS

In this chapter, we present the process of building the datasets used in this work. In Section 8.1, we explain the generation of key poses and image clusters, which explains how to label or discard images. In Section 6.2, we describe the generation of images from point clouds.

8.1 Generation of Key Poses and Image Clusters

The datasets used in this work consist of text files and images associated with the lines of the files. The images are grouped in directories, one directory for each file. Each file line contains the name of the associated image file, its label, 6D pose (*x*, *y*, *z*, *roll*, *pitch*, and *yaw*) and timestamp. The timestamps correspond to the times when the images were captured by the camera. The 6D poses correspond to the closest-in-time poses computed by the autonomous vehicle Localizer subsystem; to compute these poses, the Localizer uses the vehicle odometry, IMU and LIDAR – please see details in Veronese et al. [3]. The labels correspond to 3D poses (*x*, *y*, *yaw*), fixed sampling space in meters apart from each other along the path on the reference map, named key poses; the images receive the label of the closest key pose, which produces clusters of images around key poses (i.e., centroids of the clusters). Note that if the distance between the image pose and the closest key pose is greater than half of the sampling space, the image is discarded.

Figure 14 shows the distribution of clusters of images along the path, according to Python scripts for building datasets provided by Forechi et al. [13]. In this figure, each circle denotes the region of an image cluster, and images whose distance to the closest key pose is greater than the circle's radius are discarded. Note that depending on how images are distributed along the path, some images may be very similar, but associated with different key poses, especially those in the border regions of the clusters. This could generate ambiguous images and decrease the classification accuracy. Figure 15 shows the distribution of images and key poses from a dataset along the path, according to scripts by Forechi et al. [13]. Note that it is difficult to distinguish when images are no longer associated with a key pose, but with another one, before or after, along the path.

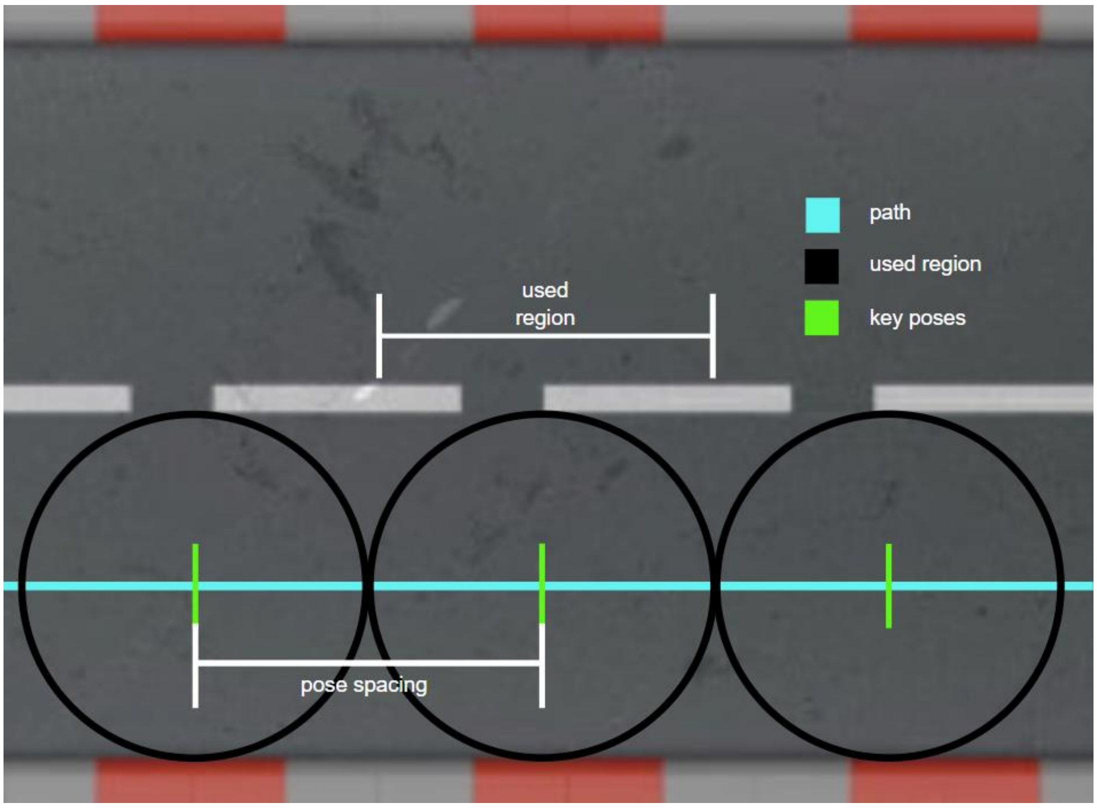

Figure 14. Distribution of clusters of images along the path. The black circles denote the regions of image clusters, the blue line denotes the path on the reference log and the green lines denote the key poses.

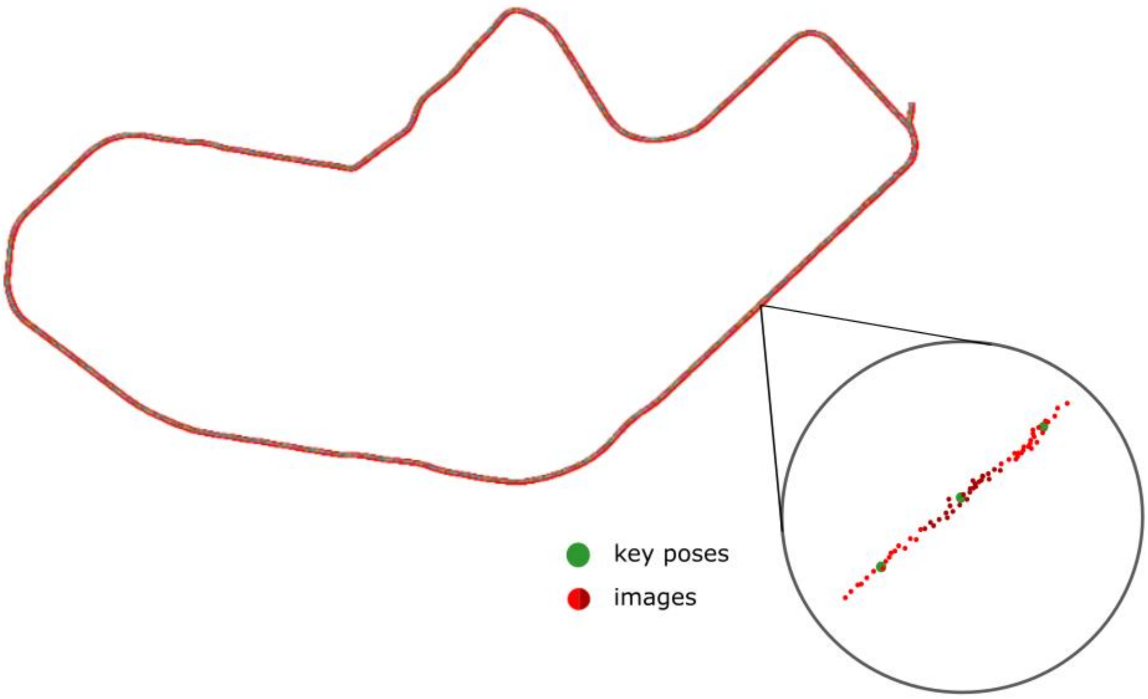

Figure 15. Distribution of images and key poses from a dataset along the path. The green dots denote the key poses and the red dots denote the images.

To avoid the problem of very similar images associated with different key poses, in this work, we reduced the radius of the circles that define the image clusters and kept the sampling space between key poses.

Figure 16 shows the new distribution of clusters of images along the path, according to our modified version of the Python scripts for building datasets provided by Forechi et al. [13]. This figure has the same format of Figure 14, except for smaller black circles and new red circles, which denote the border regions of the old image clusters that are now disregarded. Note that no matter how the images are distributed along the path, there are always gaps between the image clusters, which avoids ambiguous images that negatively affect classification accuracy.

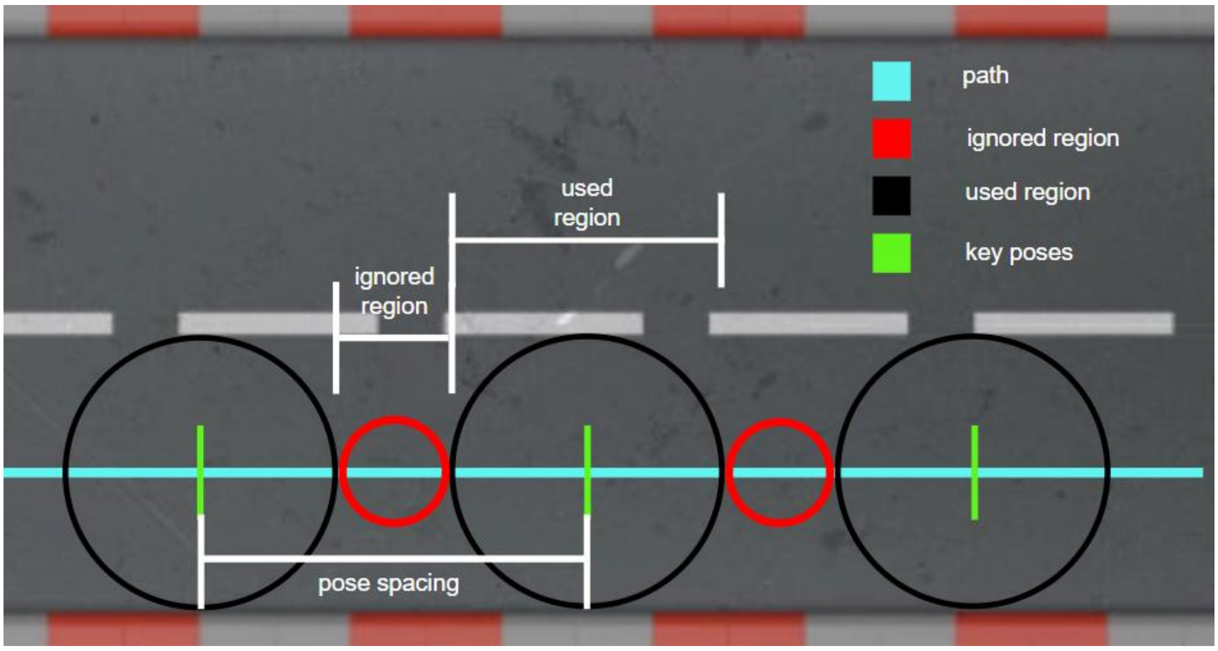

Figure 16. New distribution of clusters of images along the path. The black circles denote the regions of image clusters (images outside the black circles are discarded), the blue line denotes the path on the reference log, the green lines denote the key poses and the red circles denote the border regions of the old image clusters that are now disregarded.

Figure 17 shows the new distribution of images from a dataset along the path, according to our Python scripts. Note that there is no doubt on which images belong to each cluster. In other words, there is no question about where each image cluster starts and ends.

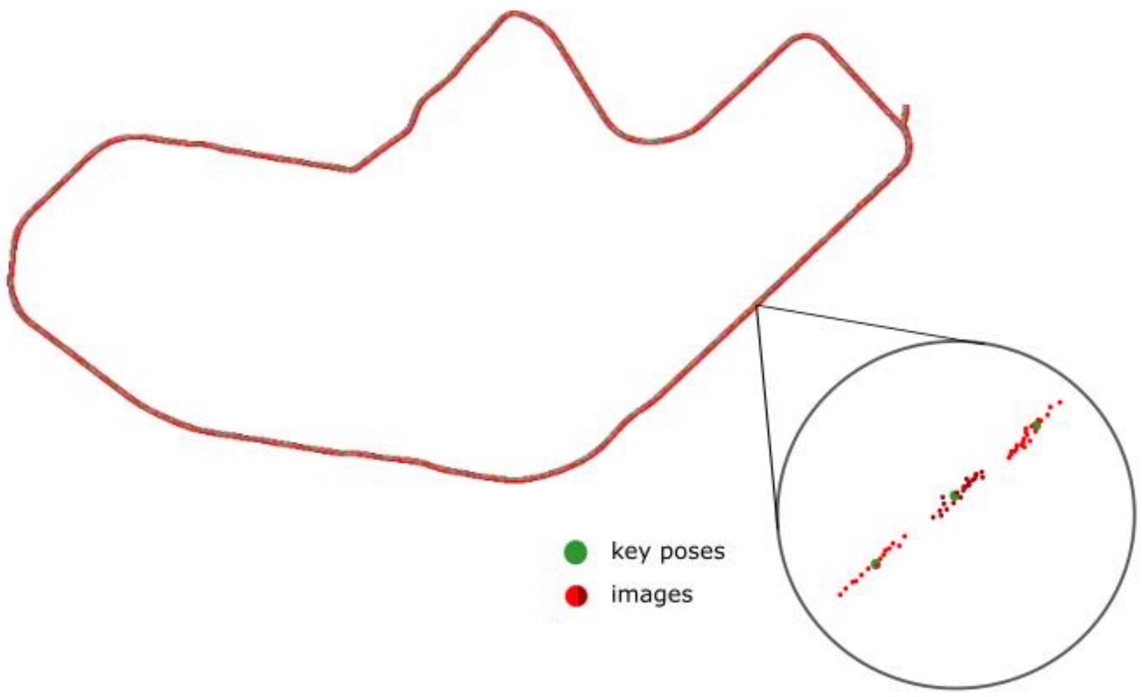

Figure 17. New distribution of images from a dataset along the path. The green dots denote the key poses and the red dots denote the images.

8.1.1 Algorithms

Algorithm 1 shows the pseudocode for generating the list of key poses. It receives as input (i) the list of poses from the reference log, “Poses”, and (ii) the sampling space, “Spacing”. In line 1, we initialize key_pose to an empty array. On line 2, we initialize the "pose_distance" variable to zero. In line 3, we initialize the "next_key_pose" variable to null. On line 4, we iterate over the "Poses" list, received as an argument of the function. In line 5, we validate if it's the first time in the loop, and if so, we set the value of "next_key_pose" to the value of "pose" (line 6). In line 8, we set the value of the variable "distance" with the Euclidean distance between "pose" and "next_key_pose". In line 9, we check if the calculated distance is greater than zero and less than "Spacing", and if so, in line 10 we set the value of "next_key_pose" with the value of "pose". In line 11, we append the value from "pose" to the "key_poses" array. Finally, in line 14, the algorithm returns the list of key poses.

Algorithm 1. Pseudocode for generating the list of key poses

```
GET_KEY_POSES (Poses,Spacing)

1.   INIT key_poses as empty array
2.   SET pose_distance to 0
3.   SET next_key_pose to NULL
4.   FOR each pose on Poses do
5.       IF next_key_pose is null THEN
6.           SET next_key_pose to pose
7.       ENDIF
8.       SET distance to euclidian distance between pose and next_key_pose
9.       IF distance > 0 and distance < Spacing THEN
10.          SET next_key_pose to pose
11.          CALL appendTo with pose, key_poses
12.      ENDIF
13.  ENDFOR
14.  RETURN key_poses
```

Algorithm 2 shows the pseudocode for labeling an image by the closest key pose. It receives as input (i) the pose of the image, “imgpos”, (ii) the list of key poses and (iii) the maximum distance allowed between the image pose and the closet key pose, “Radius”. In line 1, we initialize "closest_pose" to null. In line 2, we set the value of "min_dist" to zero. In line 3, we iterate over the "key_poses" list received as the function's argument. In line 4, we define the value of the variable "distance" with the Euclidean distance between "pose" and "imgpos" (taken as an argument of the function). In line 5, we check if the calculated distance is greater than zero, less than "Radius" and less than "min_dist", if yes, we set the value of "closest_pose" with the value of "pose" (in line 6). Finally, in line 8, the algorithm returns the key pose closest to the input image.

Algorithm 2. Pseudocode for labeling an image by the closest key pose

```
FIND_CLOSEST_KEYPOSE (imgpos, KEY_POSES, Radius )

1.  SET closest_pose to null
2.  SET min_dist to 0
3.  FOR each pose on KEY_POSES do
4.      SET distance to euclidian distance between pose and imgpos
5.      IF distance > 0 and distance < Radius and distance < min_dist then
6.          SET closest_pose to pose
7.  ENDFOR
8.  RETURN closest_pose
```

Finally, Algorithm 3 shows the pseudocode for clustering the images around the closest key poses. It receives as input (i) a list of all image-pose pairs, "pairs_of_images_and_poses" and (ii) the list of key poses, key_poses. In line 1, we initialize the “closest_pose” variable to the null value. In line 2, we iterate through the list of "pose and image" pairs. In line 3, we call the function that finds the keypose closest to the pose of the iterated image. In line 4, we check if no pose was found (null value). Finally, in line 5, if a valid pose was found, the algorithm returns the closest key pose of the image-pose pair.

Algorithm 3. Pseudocode for clustering the images around the closest key poses

```
GROUP_IMAGES (pairs_of_images_and_poses,key_poses)

1.  SET closest_pose to NULL
2.  FOR each image_pose on pairs_of_images_and_poses do
3.      CALL find_closest_keypose with image_pose, key_poses RETURNING closest_pose
4.      IF closest_pose is not NULL then
5.          OUTPUT closest_pose and image_pose
6.      ENDIF
7.  ENDFOR
```

8.2 Generation of Images from LIDAR Measurements

ART's Velodyne HDL 32-E LIDAR emits 32 vertical laser beams and rotates horizontally 360 degrees at a rate of about 20 Hz, collecting 32 readings associated with the vertical field of view (VFOV) and 1,084 columns associated with the horizontal field of view (HFOV). The Velodyne LIDAR readings correspond to distance measurements up to about 75 m.

In this work, the LIDAR distance measurements are stored in a 32×1084 matrix that was converted into a LIDAR range image of 640×480 pixels. For this, we firstly defined a new depth scale that ranges from 0 to 765 and corresponds to LIDAR distance measurements from 0 to 75 m.

Subsequently, we split the depth scale range into 3 sections, which are associated with the three channels of the RGB color scheme, and filled the channels for each image pixel according to Equation 1:

$$B = \begin{cases} (r - 510), & r > 510 \\ 0, & r \leq 510 \\ 255, & r = 0 \end{cases}$$

$$G = \begin{cases} (r - B - 255), & r > 255 \\ 0, & r \leq 255 \\ 255, & r = 0 \end{cases} \qquad \text{Equation 1}$$

$$R = \begin{cases} (r - G - B), & r > 0 \\ 255, & r = 0 \end{cases}$$

where r is the LIDAR distance measurement (from 0 to 75 m) converted to the new depth scale (from 0 to 765), and R, G and B are the channels of an image pixel, which are stored in a vector. For example, let the LIDAR distance measurement be $r = 600$. As $r > 510$, then blue channel, B, will be filled with $B = r - 510 = 90$, the green channel, G, with be filled with $G = r - B - 255 = r - 90 - 255 = 255$ and the red channel, R, with $R = r - G - B = r - 255 - 90 = 255$. Note that LIDAR distance measurements that ranges from 1 to 255 will fill the R channel only, measurements from 256 to 510 will fill both the R and G channels and the measurements from 511 to 765 the R, G and B channels. Note also that pixels approaching dark red represent measurements of objects closer to the self-driving car and pixels approaching white represent objects further away from the car. Finally, note that the LIDAR distance measurement is a float, but we converted it to an integer in the new depth scale.

Specifically, in the case of ART, the part of the LIDAR range image that depicts only the rear of the truck was discarded, which produced an image of 32 × 813 pixels that represents 270 degrees of the LIDAR, from -135 degrees to +135 degrees, taking the front as 0 degree.

Finally, we scaled the LIDAR range image to 640 × 480 pixels.

Figure 18 shows an example of an image generated from LIDAR distance measurements.

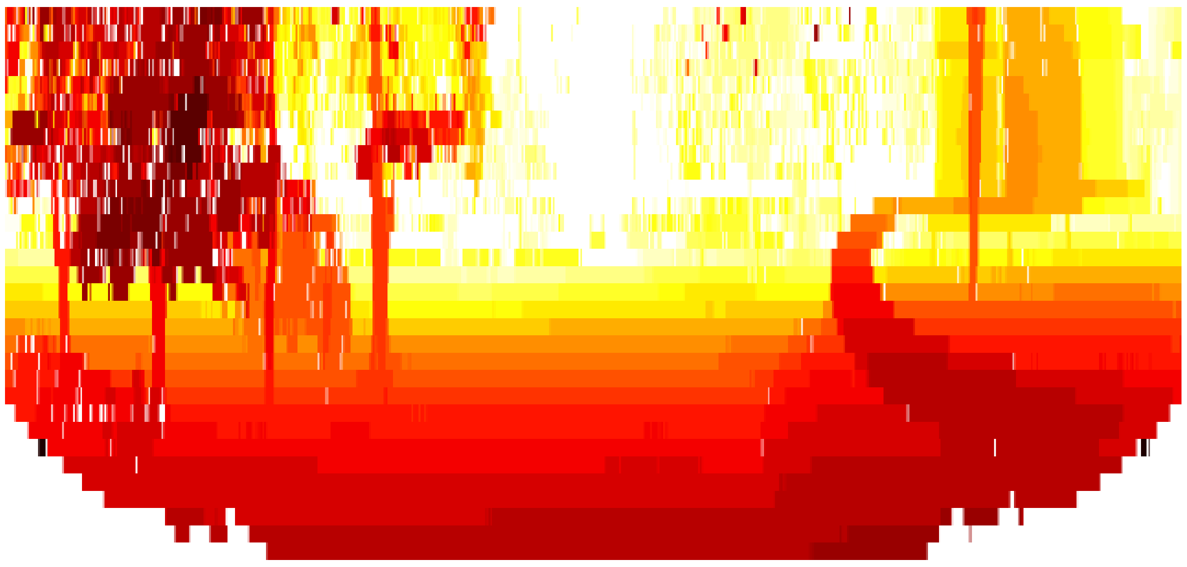

Figure 18. Example of an image generated from LIDAR distance measurements. Pixels approaching dark red represent measurements of objects closer to the autonomous vehicle, while pixels approaching white represent objects further away from the vehicle.

In this work, the LIDAR HFOV (360^{o}) is much larger than the camera HFOV (66^{o}). In addition, LIDAR provides distance measurements. However, while the camera VFOV is slightly larger and much denser (50^{o} and 480 rows) than the LIDAR VFOV (40^{o} and 32 rows), these camera advantages does not outweigh the LIDAR advantage of having a higher HFOV and providing depth information via distance measurements.

Figure 19 shows the difference between the camera FOV and LIDAR FOV. In this figure, we observe that the LIDAR image covers a wider view, which helps DeepVGL in the task of distinguishing very similar scenes considering all their surroundings.

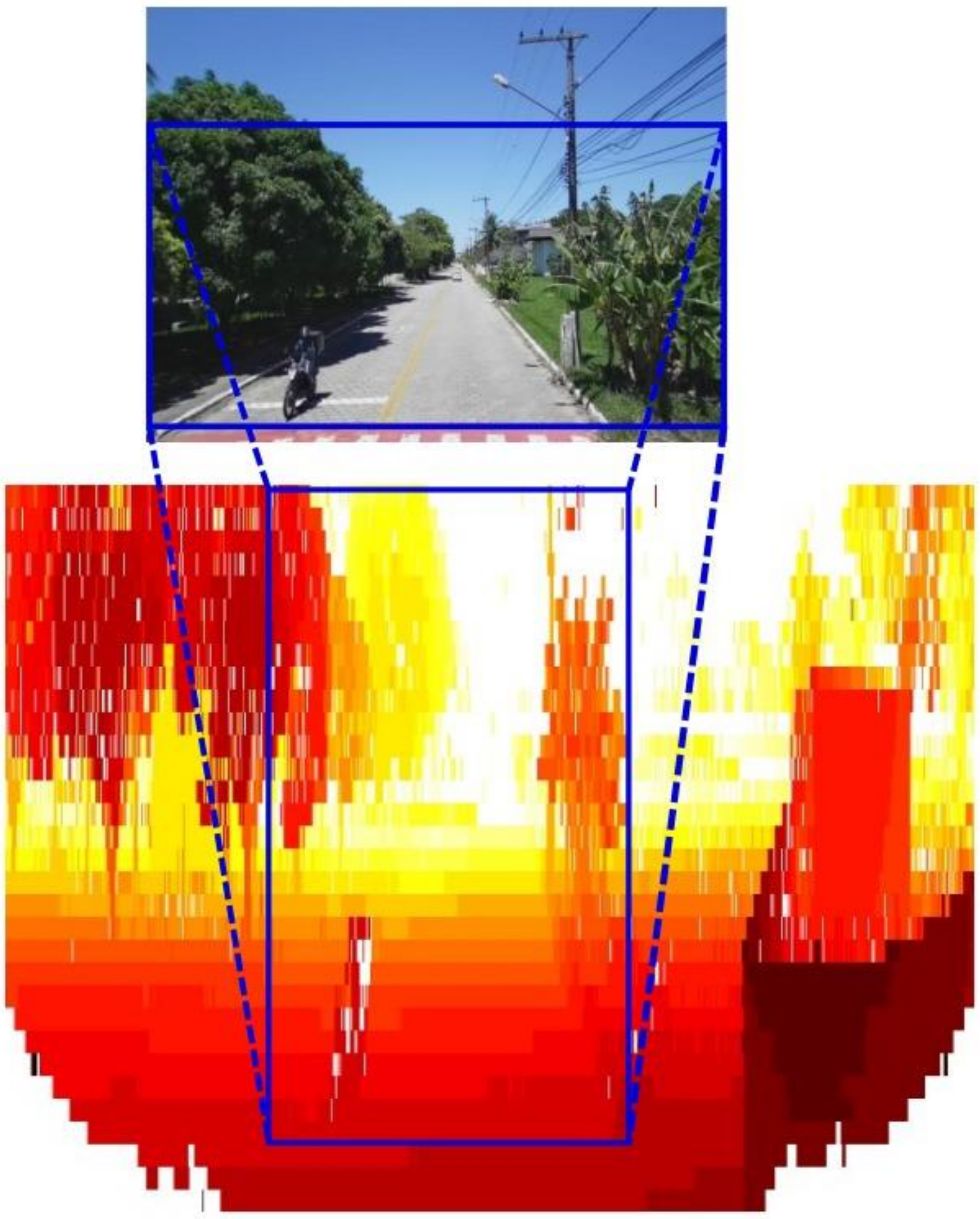

Figure 19. Difference between camera FOV and LIDAR FOV. The image at the bottom denotes the LIDAR range image and the image at the top denotes the camera image. The scene represented by the bounding box in the LIDAR range image corresponds to the scene represented by the bounding box in the camera image.