This article has been accepted for publication in IEEE Transactions on Intelligent Transportation Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TITS.2022.3214079

Dynamic Conditional Imitation Learning for Autonomous Driving

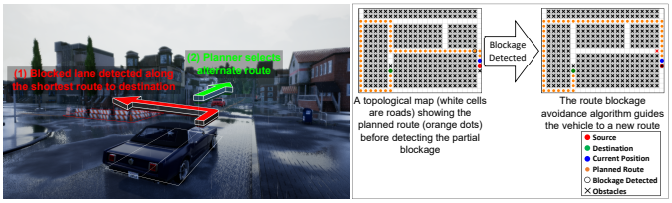
Hesham M. Eraqi[1,2], Mohamed N. Moustafa[1], and Jens Honer[2]

***Abstract*—Conditional imitation learning (CIL) trains deep neural networks, in an end-to-end manner, to mimic human driving. This approach has demonstrated suitable vehicle control when following roads, avoiding obstacles, or taking specific turns in intersections to reach a destination. Unfortunately, performance dramatically decreases when deployed to unseen environments and is inconsistent against varying weather conditions. Most importantly, the current CIL fails to avoid static road blockages. In this work, we propose a solution to those deficiencies. First, we fuse the laser scanner with the regular camera streams, at the features level, to overcome the generalization and consistency challenges. Second, we introduce a new efficient Occupancy Grid Mapping (OGM) method along with new algorithms for road blockages avoidance and global route planning. Consequently, our proposed method dynamically detects partial and full road blockages, and guides the controlled vehicle to another route to reach the destination. Following the original CIL work, we demonstrated the effectiveness of our proposal on CARLA simulator urban driving benchmark. Our experiments showed that our model improved consistency against weather conditions by four times and autonomous driving success rate generalization by 52%. Furthermore, our global route planner improved the driving success rate by 37%. Our proposed road blockages avoidance algorithm improved the driving success rate by 27%. Finally, the average kilometers traveled before a collision with a static object increased by 1.5 times.**

***Index Terms*—Autonomous Driving, Occupancy Grid Mapping, Conditional Imitation Learning, Sensor Fusion, Road Blockages Avoidance**

I. Introduction

DESPITE the recent advances to achieve the promising vision of autonomous driving in terms of significantly reducing accidents [47] and congestion [6], while being environmentally and economically beneficial [17], it is safe to believe that fully autonomous navigation in complex environments is still decades away [31] [27] [21]. Autonomous vehicles employ a "sense-plan-act" design which is the basis of many robotic systems. Advanced forms of LiDAR (laser scanner, an acronym of Light Detection And Ranging), radar, and inertial measurement allowed for a more accurate and quicker sensing of the environment and surrounding objects. Nevertheless, many open challenges remain yet to be fully solved in: 1) planning the vehicle's actions based on understating the driving scene and the interaction between its elements given the sensed data and 2) eventually commanding the vehicle's control system steering, throttle, and brakes. The algorithmic pipeline includes tasks such as mapping, localization, driving scene perception, motion planning, and trajectory optimization which are full of open challenges [24] including requiring expensive data annotation, relying on heuristics and handcrafted rule-based modules, and the potential of adding unnecessary complexity to the problem. Therefore, researchers turned to train end-to-end deep neural networks to directly learn the mapping from front-facing camera data stream to driving commands.



(a)

(b)

Fig. 1. Partial road blockages added to CARLA simulator, the road blockages avoidance algorithm detects them and guides the vehicle to another route to reach the destination as demonstrated in (A), while in (B) the partial blockage is in another lane.

Recently, there has been an increasing amount of literature adopting the end-to-end approach [4] [20]. Such systems are demonstrated suitable when following roads and avoiding obstacles. The conditional imitation learning (CIL) method [12] upgraded the end-to-end approach by allowing the vehicle to be automatically guided at test time to take a specific turn at an upcoming intersection to reach the destination. The model conditions imitation learning on a high-level navigational command input received from a global route planner, just as mapping applications, that instructs the model to control the vehicle to take a specific turn, go straight through an intersection, or follow lane. In [14], CARLA urban driving benchmark is adopted to benchmark the CIL model [12] against other approaches to autonomous driving. CARLA [14] is a widely used open-source simulator for autonomous car development focused on creating realistic virtual environments for the automotive industry. Many contributors constantly

[1]Hesham M. Eraqi and Mohamed N. Moustafa are with Computer Science and Engineering Department, The American University in Cairo, Egypt, e-mails: heraqi@aucegypt.edu, m.moustafa@aucegypt.edu.

[2]Jens Honer and Hesham M. Eraqi are with Driving Assistance department, Valeo Schalter und Sensoren GmbH, Germany, emails: jens.honer@valeo.com, hesham.eraqi@valeo.com.

improve it, which makes it a comprehensive tool for simulating real-world scenarios. The benchmark results demonstrated that the CIL model is responsive to the high-level navigational commands and drives efficiently when tested on the same training environments. However, performance is found inconsistent against varying weather conditions and significantly decreases when tested on environments that are unseen during the model training. Generalization from one town's road layout and environment domains (represented by e.g. different texture sets) to other towns is a concern. Most importantly, the CIL method cannot avoid road unexpected temporary stationary blockages, as work zones in figure 1, which are ever-increasing in number on world roads [50]. Such road blockages should be detected and the global route planner should dynamically estimate and guide the vehicle to a new route towards the destination accordingly. The aim of this work is to address two main issues of the CIL method: 1) lack of generalization to new environments and inconsistency against varying weather conditions and 2) failure to avoid static road blockages.

The contribution of this paper is two-fold. First, we provide a novel architecture that aims to tackle the CIL model [12] challenges of lack of generalization and inconsistency against varying weathers, by extending it via fusing a LiDAR sensor input with the camera. Camera and LiDAR are primary sensor modalities for autonomous driving to capture environment semantic and geometric information respectively. The strengths of each sensor can compensate for the weaknesses of the other. The accurate LiDAR range information resolves the camera depth perception ambiguity, while the camera's dense angular resolution compensates for LiDAR sparsity. LiDAR also is less sensitive to ambient lighting [32]. The proposed model in this work aims to tackle the CIL model [12] challenges of lack of generalization and inconsistency against varying weathers, by extending it via fusing a LiDAR sensor input with the camera. On CARLA simulator urban driving benchmark [14], the proposed model improved the autonomous driving success rate in towns unseen during the training by 52% and improved weather consistency by 3.9 times. It outperforms the CIL model in all the different combinations of tasks and environmental setups while being trained on driving traces recorded automatically.

Our second contribution is a new efficient Occupancy Grid Mapping (OGM) method that inspired from a part of our patent in [23] and used in new road blockages avoidance and topological global route planning algorithms. Detailed knowledge about the environment is useful for autonomous driving. A versatile approach to this task is to use OGM [16] to generate a map from noisy and uncertain sensor measurements. The road blockages avoidance and route planning algorithms allow the vehicle to detect unexpected road closures and to dynamically estimate a new shortest route accordingly in order to reach the destination while avoiding closed lanes or roads. Additionally, the OGM is used to rectify the proposed model output to reduce the chances of driving on the sidewalk. On CARLA benchmark [14], the proposed global route planner method improved the driving success rate by 37% by providing more accurate navigational commands. The CARLA simulator and benchmark are upgraded to support testing navigation while having unexpected temporary stationary road blockages, and our road blockages avoidance algorithm improved the driving success rate by 27% and reduced infractions with static objects by 1.5 times.

II. Related Work

The mediated perception approaches [31] [14] for autonomous driving involve sub-components for detecting driving-relevant objects [25] (as cars [26], pedestrians [38], lanes' markings [34], or more objects), tracking of driving scene objects [8], motion planning [30], drivable free space detection [40] [19], collision avoidance [18], mapping [48], and more models. Then the results from these sub-components are then combined in a rule-based module that produces vehicle driving actions [14]. Such mediated perception approach relies on scene understanding [25] on a level that might add redundant information and unnecessary complexity; a small portion of the detected objects are relevant to driving decisions. It also requires robust solutions to open challenges in scene understanding and expensive data annotation [25]. Direct perception [9] is another approach that learns a mapping from input camera image to several meaningful affordance indicators of the driving situation, then a rule-based controller translates them into driving actions. Those indicators are chosen via heuristics and the controller design is as expensive as the case with the mediated perception rule-based module [49]. Another approach is to learn driving trajectory planning end-to-end while leaving the vehicle control component outside the end-to-end trained module [2] [7].

As an alternative to the mediated perception approach, the end-to-end imitation learning approach directly maps input sensory data to driving actions via deep machine learning regression [4] [20] [31] [5]. Such behavior reflex approach optimizes the aforementioned autonomous driving sub-components simultaneously, the system self-optimizes aiming to maximize overall system performance. Unlike in the mediated perception approach, optimizing human-selected intermediate criteria doesn't guarantee maximizing overall system performance, because such criteria are selected to ease human interpretation [4]. The major drawback of the end-to-end learning approach is that the vehicle cannot be guided to take a specific turn at an upcoming intersection. The CIL model [14] [12] overcomes the end-to-end approach such a limitation by training, on top of a perception Convolutional Neural Network (CNN), multiple different command-conditional modules ("branches") predicting driving commands for each possible navigational command. A topological global route planner is used to estimate a sparse list of waypoints towards the destination and based on it, the navigational command is determined for each waypoint. Based on the navigational command received from the global route planner, the proper branch is selected to control the vehicle in order to reach the destination. In [40], a hybrid A* search algorithm [13] is developed to predict a detailed vehicle drivable trajectory to reach the destination which is more computationally expensive than the adopted planning method in this work that estimates a sparse list of waypoints. Similarly, another approach for global

planning is based on deep learning [51] which is considerably more computationally expensive. Another weakness in such a learning-based planner is that it depends on the structure of a particular environment where the model is trained on and requires a semantic map of the environment. To cope with potential changes in certain regions in the environment, a local learning-based planner can be added which further increases the computational demands.

In [14] [12], the CIL model is demonstrated to drive efficiently when deployed on the same training environments. However, performance dramatically decreases when deployed to new environments that are unseen during training time and is inconsistent against varying weathers. In addition, the CIL method cannot dynamically detect and handle road unexpected temporary stationary blockages like for example due to work zones. Various reasons have led to an ever-increasing number of road work zones of different looks and shapes even during demanding traffic levels due to maintenance, construction, and rehabilitation activities [50]. Autonomous vehicles should dynamically detect such road blockages and the global route planner should estimate a new route to the destination as a result. In Junior car [40], road blockages are avoided using a drivable free space detection rule-based method. The LiDAR point cloud is analyzed against predefined thresholds and a hybrid A* planner was introduced to predict the vehicle trajectory, which makes such a method not directly compatible with the approach of learning the driving policy end-to-end. Moreover, three LiDAR sensors were used to predict a reliable vehicle trajectory. Table I summarizes the comparison between mediated perception and end-to-end approaches.

Limitation	Mediated Perception	End-to-end
Requires robust solutions to open problems in scene understanding	Yes	**No**
Requires rule-based controller	Yes	**No**
Heuristics involved	Usually	**No**
Adds useless complexity in terms of detected that are irrelevant to driving decisions	Yes	**No**
Machine learning algorithm confusion caused by similar inputs being associated with different labels	**No**	Yes*
Requires expensive training data annotation	Yes	**No**
Can't see the bigger picture of the driving situation	**No**	Yes*

* Limitations tackled by the CIL approach

TABLE I
THE ADVANTAGES AND DISADVANTAGES OF THE MEDIATED PERCEPTION AND END-TO-END APPROACHES FOR AUTONOMOUS DRIVING

The Occupancy Grid Mapping (OGM) algorithm is introduced in [16]. The produced map is represented in a top-view grayscale image format, where pixel intensities represent the probability of occupancy given LiDAR point cloud. Within each grid cell, the occupancy is estimated recursively using a binary Bayes filter, which creates a history effect that makes it robust against the problems of missed and false measurements. The original OGM algorithm [16] assumes map cells independence, which induces map grid cell conflicts that lead to inconsistent maps. To overcome such an inconsistency problem, a forward model [43] is introduced to maintain dependencies between neighboring map grid cells. The algorithm is widely adopted for probabilistic localization and mapping in robotics [44], existing autonomous vehicles [40], and vehicle trajectory prediction [35]. Mapping large roadway environments with a high-resolution OGM can impose prohibitive memory requirements. Existing probabilistic quadtrees methods [37] provide compact map representations that significantly reduce the OGM memory footprint, however, they do not guarantee that the stored mapping information is utilized for the roadway information that is most important to the driving situation. This can be achieved by dynamic vehicle positioning, combined with map management techniques based on 2D ring buffers [41] or twisted torus topology [29]. Nevertheless, vehicle dynamic positioning results into map inaccuracies due to the computationally expensive and inaccurate image sub-pixel shifting and rotation operations to compensate for ego-vehicle motion.

III. Proposed Model

Figure 2 introduces our proposed network architecture. The network is end-to-end trainable, given input sensory data, the vehicle driving commands are predicted, in addition to predicted vehicle speed. The network receives the high-level navigational command C as an input, alongside the image coming from a front-facing camera, LiDAR point cloud, and a measurements vector. C is a turn command provided by a global route planner and acts as a switch that selects between specialized output sub-module branches. The planner should set C to select the "follow lane" output branch in case of the vehicle is far away from road intersections, and during intersection, it should select the proper branch which could be to turn left, to turn right, or to go straight, based on the road layout and the desired destination. We adopt a topological global route planner that provides accurate navigational commands as described later in subsection V-A.

The camera and LiDAR input modalities are processed independently. The currently observed camera RGB image is fed into eight convolutional layers, and the associated LiDAR point cloud is encoded to a grayscale image and it is then fed into four convolutional layers. The LiDAR point cloud image follows a Polar Grid View (PGV) representation. As in figure 2, the currently observed LiDAR point cloud is encoded to a grayscale image using Polar Grid View representation (PGV). Figure 3 shows a sample camera RGB image, the corresponding LiDAR point cloud full scan top view projection, and the generated PGV which provides a 2D dense proximity spherical representation of the environment. Each LiDAR layer is associated with a PGV row, and each beam is associated with a single PGV column based on its horizontal angle. A PGV pixel holds the average depth values for all LiDAR beams that are associated with it. Such a projection-based method maps the 3D sparse point cloud into a 2D image representation that is more dense and compact compared to 3D LiDAR scan points. Consequently, standard 2D CNN can be leveraged to process those range images to achieve real-time performance [11].

We use a ReLU activation function [39] in all hidden layers, and a linear activation for the output layers. Figure 2 describes the number of neurons per layer, and for the convolutional

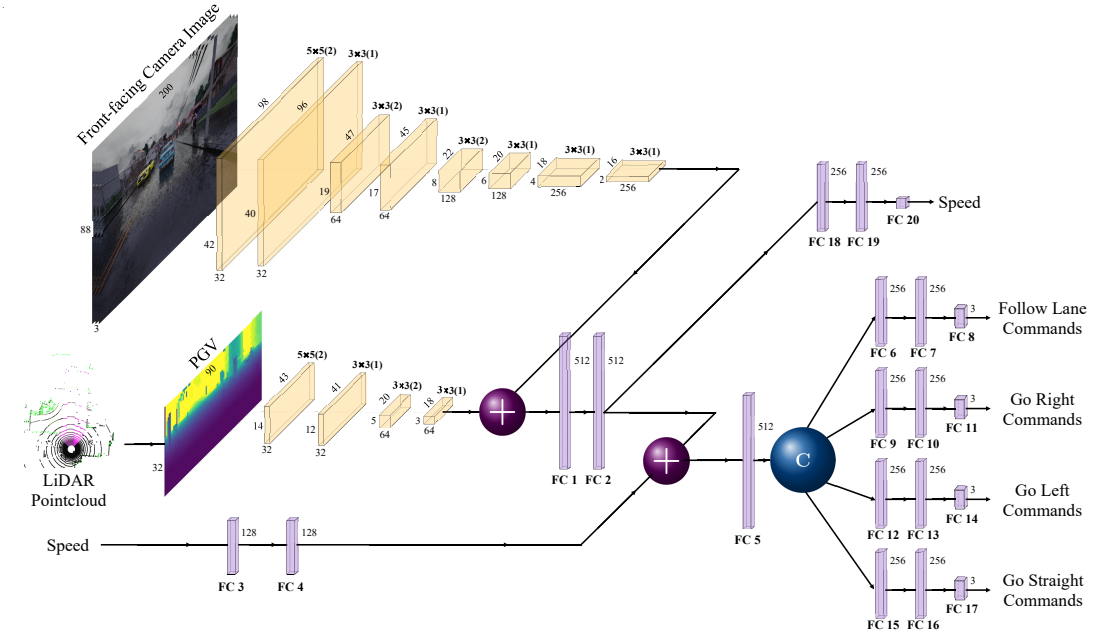


Fig. 2. Proposed Network Architecture

layers describes kernel sizes and the used padding as well. Batch normalization is applied after all the convolutional layers, and we apply 50% dropout after fully-connected hidden layers. For input measurements, we only use the actual vehicle speed as in [12]. The speed, throttle, and brake values are scaled between 0 and 1, according to minimum and maximum possible values. The steering wheel angle is scaled between -1 and 1, with extreme values corresponding to full left and full right. The camera RGB images and the LiDAR PGV images are normalized to be in the range of $[0, 1]$. For each output branch, driving actions a are three-dimensional vectors that include steering wheel angle s, throttle t, and braking b; $a = [s, t, b]$. Given ground-truth actions a_g and speeds v_g, and predicted actions a and speeds v, the loss function L is defined as follows: $L = \lambda_s \|s - s_g\|^2 + \lambda_t \|t - t_g\|^2 + \lambda_b \|b - b_g\|^2 + \lambda_v \|v - v_g\|^2$, where λ_s, λ_t, λ_b, and λ_v are empirically set to 0.5, 0.2, 0.15, and 0.15 respectively. The model is trained using Adam optimizer [36] with $\beta_1 = 0.7$, $\beta_2 = 0.85$, and initial learning rate of 0.0002 that is multiplied by 0.5 every 10 epochs. We used mini-batches of 120 samples, where each min-batch has the same number of samples for each navigational command C. Half of the images in every mini-batch are augmented as described later in this section. Figure 4 graph shows the training and validation losses per epoch.

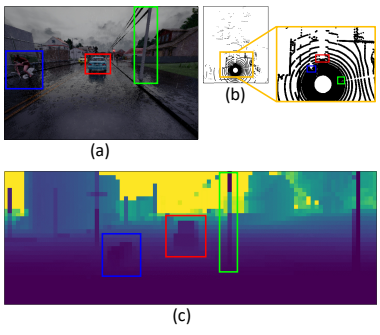


Fig. 3. (a) Sample RGB camera image. (b) Corresponding LiDAR point cloud top view projection. (c) Generated PGV from the LiDAR point cloud. Three objects are matched in the figures: a vehicle, a bicyclist, and a light pole.

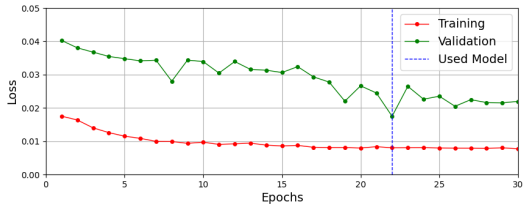
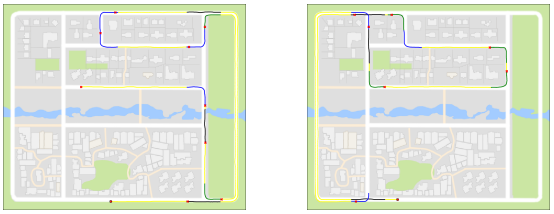


Fig. 4. Training and validation losses per epoch

The original CIL model is trained on a dataset collected by a human driver using CARLA simulator, who uses a signal to record his intent when approaching intersections [12]. That signal was used as the ground-truth navigational high-level command. In contrast, our model is trained on data that is automatically recorded using two different methods. The first data collection method relies on CARLA simulator autopilot feature. In each data collection episode, the weather, traffic and pedestrians density, and vehicle starting position are randomly chosen. The ego-vehicle purposelessly follows lane and take random turning decisions in intersections and avoid obstacles for a predefined time of 10 minutes. After each episode, the navigational high-level command is generated by looking-ahead in future frames to determine the turn the vehicle decided to take. Figure 5 shows the generated high-level command generated for two sample episodes. In the second data collection method, we utilize CARLA route



(a) Sample episode 1 (b) Sample episode 2

Fig. 5. High-level commands generated for two sample episodes during training data collection using CARLA autopilot. The red circle represents start position and the colored trajectory shows the driving path. Colors represent the command; yellow for "follow-lane", blue for "go left", green for "go right", and black for "go straight". Red crosses represent samples dropped out due to long traffic stopping.

planner and PID (proportional integral derivative) controllers. In each episode a random pair of source and destination are chosen, then CARLA provides navigational waypoints that the ego-vehicle should follow to reach destination. The modular pipeline system introduced in [12] is used to follow waypoints and avoid obstacles while making use of simulator privileged information. The training town has 2.9 km of drivable roads. For each data collection method, 200 episodes are conducted. Each episode has a number of vehicles and pedestrians uniformly randomly sampled from the the ranges [30, 60] and [50, 100] respectively.

As in [12], temporally-correlated noise is injected into the steering during training. The noise simulates gradual drift away from the desired trajectory, then the vehicle is let to recover from these perturbations to provide the network with examples of recovery from unexpected disturbances. During model training, online data augmentation is applied on half of the mini-batch images before feeding them to the network. To augment an image, it is passed through a pipeline of a sequential series of augmentation methods. Each augmentation method in the pipeline has a predefined probability of occurrence which defines the percentage of augmented images having that method existing in their augmentation pipeline. In addition, each augmentation method has stochastic parameters to let each image be augmented differently. As an example, when adding Gaussian noise, for each image to be augmented, the Gaussian noise variance is sampled from a parameterized uniform probability distribution. Two different types of data augmentation methods are adopted. The first type is for photometric transformations: changing brightness, lighting conditions, and applying additive white Gaussian noise and Gaussian blurring [33]. The second type is for geometric transformations: horizontal flipping. In the case of horizontal flipping, the sign of the ground-truth steering wheel angle is flipped as well.

The proposed model is able to drive on two-lane roads (one traveling in one direction, and one traveling in the opposite direction) while having intersections and traffic lights. The training and testing towns of CARLA urban driving benchmark [14] do not include multi-lane roads and roundabouts which are beyond the scope of this study. The model can be trained to drive on multi-lane roads as the LiDAR 360-degree field of view can enable for learning lane change maneuvering; however, the adopted route planner described in subsection V-A needs to be adapted to support modeling more complex road networks based on schemes as Lanelets [3] [42] and OpenDrive [15]. The proposed model is also compatible to learn to drive through roundabouts as long as the route planner timely provides the navigational commands required to exit them.

IV. Efficient Occupancy Grid Mapping (OGM)

The original OGM algorithm [16] assumes map cells independence, which induces map grid cell conflicts, because a single sensor measurement may update several grid cells, which lead to inconsistent maps [43]. To overcome such inconsistency problem, a forward model [43] can be used which produces more accurate OGM by maintaining dependencies between neighboring map grid cells. The forward model approach uses the Expectation-Maximization algorithm to build the map and a Laplacian approximation to model uncertainty. In this work, we introduce a new simpler and faster way to acquire OGM that preserves map quality by maintaining dependencies between neighboring map cells. As in [43], our inverse sensor model handles cells overlapping multiple measurements issue in [16] by generating maps from all full scan measurements at once, not incrementally on single bream measurements. Unlike [43], our method makes use of unreflected beams (beams with no echo returned), because they indicate important free space information. Given full scan measurements and vehicle position, the map grid cells to be updated can be determined by the convex hull or the bounding polygon of the full scan measurements. This approach assumes having dense measurements, which is a convenient assumption with LiDAR sensors. As shown in figure 6, both methods produce equivalent maps except for small parts around unreflected beams areas. Our method is described in Algorithm 1. Horizontal arrow symbols in the algorithm description indicate appending to a list. The *filter* function removes scan points from ground and dynamic objects based on 2D camera semantic segmentation [1] projected into LiDAR 3D space. The function also removes overhanging objects above a predefined height threshold (the vehicle height plus a safety buffer), like traffic signs, high tree leaves, and billboards. To make the algorithm faster, scan points could be down-sampled systematically by keeping each k^{th} scan point.

Table II compares our OGM method with [43] and [16]. As in [43], our model adopts Bayes filtering in log-odds representation of the occupancy probabilities incremental composition, which is more computationally efficient than [16]. In the Algorithm 1, the *position_circle* function executes our vehicle positioning method (Algorithm 2) which aims to make the map incremental composition faster and more accurate as detailed in subsection IV-A. That method provides better memory utilization via efficient vehicle positioning in the map while preserving map accuracy by preventing map rotation and sub-pixel shifting transformations which introduce cumulative artifacts to the map.

Algorithm 1: Proposed Occupancy Grid Map (OGM) method

Input: $scans$: $n \times 2$ array with full scan n measurements, each measurement represented by its x and y coordinates
P_w^t: vehicle position in world coordinates
P_w^{t-1}: previous vehicle position in world coordinates
s: vehicle speed
M: OGM to be updated

Parameters: log_odd_{free}: log odd for free
log_odd_{occ}: log odd for occupancy
w: wall (obstacle) depth
α: beam width angle

Output: M: updated OGM

// Transform map and calculate vehicle position and orientation in map local coordinate system (run Algorithm 2)

$P_{local}, M = \textbf{position\_circle}(s, P_w^t, P_w^{t-1}, P_{local}, M)$

$scans = \textbf{filter}(scans)$

// Transform to map coordinate system and shift scan points

$scans = scans. \begin{bmatrix} \cos(P_{local}.yaw) & -\sin(P_{local}.yaw) \\ \sin(P_{local}.yaw) & \cos(P_{local}.yaw) \end{bmatrix}$

allocate 2 empty lists $angles$ and $distances$

foreach scan point $scan_i \in scans$ **do**

$angles \leftarrow \text{atan2}(scan_i.y - P_{local}.y, scan_i.x - P_{local}.x)$

$scan_i.x \mathrel{+}= w.\cos(angles[i])$

$scan_i.y \mathrel{+}= w.\sin(angles[i])$

$distances \leftarrow \sqrt{(scan_i.x - P_{local}.x)^2 + (scan_i.y - P_{local}.y)^2}$

$area = \textbf{get\_affected\_area}(M, scans)$ // convex hull or polygon

foreach cell $cell \in area$ **do**

$cell_{dist} = \sqrt{(cell.x - P_{local}.x)^2 + (cell.y - P_{local}.y)^2}$

$cell_{angle} = \text{atan2}(cell.y - P_{local}.y, cell.x - P_{local}.x)$

$near\_beams =$ list of i where $|cell_{angle} - angles[i]| < \frac{\alpha}{2}$

if $near\_beams$ list is not empty **then**

$scan_{dist} = \min(distances[near\_beams])$

if $cell_{dist} < scan_{dist} - w$ **then**

$M[c] \mathrel{-}= log\_odd_{free}$

else if $cell_{dist} \leq scan_{dist}$ **then**

$M[c] \mathrel{+}= log\_odd_{occ}$

else

$nearest\_beam = \arg\min_i (|cell_a - angles[i]|)$

if $cell_{dist} < distances[nearest\_beam]$ **then**

$M[c] \mathrel{-}= log\_odd_{free}$

TABLE II
OCCUPANCY GRID MAPPING ALGORITHMS

Criteria	Elfes [16]	Thrun [43]	Ours
Inverse sensor model accuracy	Low	**High**	**High**
Incremental composition speed	Slow	**Fast**	**Fast**
Uses unreflected beams information	No	No	**Yes**
Memory utilization	Low	Low	**High**
Map transformation speed	Low	Low	**High**

A. OGM Vehicle Positioning

Using a global grid map is convenient for robotics applications in a controlled area [28]. But for a high-speed driving vehicle, it is not a convenient option due to limited memory and the irrelevance of old locations. Regardless of how big the memory storage is, eventually the vehicle will leave the map boundaries. Hence, a common approach is to use a local map that moves with the ego-vehicle. However, this introduces major computational burdens, namely rotation and sub-pixel shifting of the grids to compensate for ego-vehicle motion.

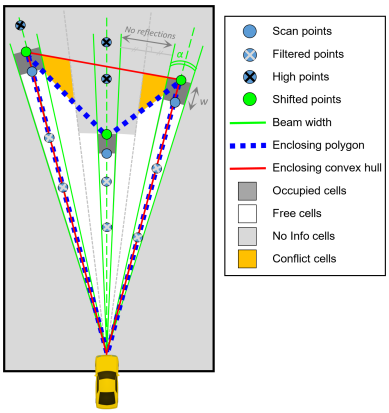


Fig. 6. Comparison between considering the full scan affected area as the scan points convex hull or bounding polygon. The areas in orange color are considered "free" or as "no info" in case of the convex hull and the polygon-based methods respectively.

Sub-pixel shifting is required because the vehicle motion is not necessarily a multiple of grid cell size. Additionally, both operations lead to discretization errors and create accumulated artifacts that lower the overall map quality.

In this work, we introduce a new method of vehicle positioning, inspired by a part of our patent in [23], that allows for more accurate and computationally efficient OGM by avoiding rotation and sub-pixel shifting operations. Map sub-pixel shifting is avoided by moving the ego-vehicle within the map by the non-integer part of the required shifting, while rotation is avoided by keeping the map orientation fixed to some global coordinate system and rotating the ego-vehicle itself. The latter approach requires a square map. Yet in high-speed scenarios such as driving on a highway, the autonomous driving function is more interested in the environment in front of the vehicle. In turn, the square map reserves a lot of memory for regions of low interest if the vehicle is centered in the map. Our solution to this limitation is to grant even more freedom to the location of the ego-vehicle as shown in figure 7. The key idea is to locate the sensor vehicle on a circle with its orientation orthogonal to the circle tangent. The circle center itself is located in the square map center. The size (radius) of the circle and the angle on which the ego-vehicle is placed on it are determined by the speed and the rotation of the vehicle (yaw angle) respectively. Our algorithm is described in Algorithm 2. Initially: $P_l^0 = P_c$, $P_w^0 = P_w^1$, and M is a matrix filled with identical values of the average of P_{Free} and P_{Occ} to represent no prior occupancy information. At each full scan OGM update iteration t, the algorithm is executed to compensate for ego-vehicle motion, and afterwards, OGM Algorithm 1 is executed. The *shift_down* and *shift_left* functions shift up and right if they received negative shift values respectively.

Our algorithm is a utility that can be used to position the vehicle within grid maps, regardless of the algorithm used to build these maps. It's a pure map alignment method that saves computational time by avoiding image rotation and sub-pixel shifting. At the same time, it results in more accurate grid maps by avoiding approximations resulted from such two operations.

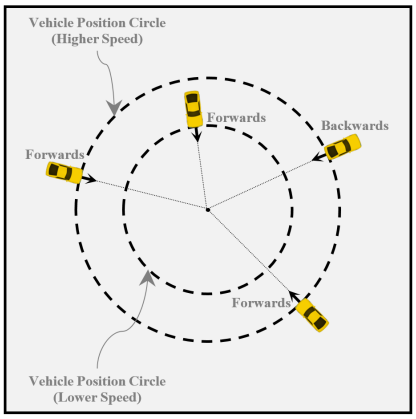


Fig. 7. Position vehicle within OGM on a circle

Algorithm 2: OGM vehicle on a circle algorithm

Input: s^t: vehicle speed
P_w^t: vehicle position in world coordinates
P_w^{t-1}: previous vehicle position in world coordinates
P_l^{t-1}: previous vehicle position in OGM
M^{t-1}: OGM to be updated
Output: P_l^t: vehicle position in OGM
M^t : updated OGM

$P_c = \textbf{get\_position\_circle}(P_w^t.yaw, s^t)$
$P_l^t.yaw = P_w^t.yaw$
// Shift map by shift value integer part
$P_{shift} = P_w^t - P_w^{t-1} + P_l^{t-1} - P_c$
$M^t = \textbf{shift\_down}\ M^{t-1}\ \textbf{by}\ \lfloor P_{shift}.y \rfloor$
$M^t = \textbf{shift\_left}\ M^t\ \textbf{by}\ \lfloor P_{shift}.x \rfloor$
// Shift local position by the shift value fractional part
$P_l^t.x = P_c.x + \{P_{shift}.x\}$
$P_l^t.y = P_c.y + \{P_{shift}.y\}$

On the other hand, the algorithm allows adopting smaller grid map sizes, and hence, less memory consumption and better real-time performance. Because it utilizes a higher proportion of the local map space for important areas based on vehicle movement trajectory. Our method remains compatible with existing efficient map management techniques of representing map cells in terms of 2D ring buffers [41] or twisted torus topology [29].

V. Road Blockages Avoidance

A. Global Route Planner

We adopt a topological global route planning algorithm that is similar to the one used in [12] and [14]. The implementation logic is upgraded to provide more accurate high-level navigational commands C and to execute faster. As in Algorithm 3, a command from four possibilities of turn left, turn right, go straight, and follow lane is returned based on the vehicle destination GPS coordinates and orientation which defines the arrival lane, using the vehicle GPS and compass. As in [12], the planning is carried out on a one-way roads grid map for simplicity and to make planning faster; the A* search algorithm is carried out after setting the map cell after the destination and the cell behind the vehicle as occupied, i.e.; putting 'walls'. In the route planner in [12], the A* algorithm is executed to re-evaluate the shortest path towards the destination every time the vehicle travels to a new map cell.

Algorithm 3: Global Route planner algorithm

Input: $roads$: list of town roads, a road is represented by the GPS coordinates of its start and end
car_gps: vehicle GPS coordinates
$car\_compass$: vehicle orientation in the world
$dest\_gps$: GPS coordinates of the desired destination
$dest\_orient$: the destination orientation (defines the desired arrival lane)
ogm: OGM with vehicle position in it
Parameters: res: planning map resolution, the output command is constant within a cell
far_inters: number of map cells to the nearest intersection to decide it is far away
$inter\_exited$: number of cells away from a visited intersection to decide the vehicle left it
Output: C: a high level navigational command, used by driving model

// Initializing global variables
if algorithm called for first time **then**
  $world\_graph = \textbf{directed\_weighted\_graph}(roads)$ // A node for each roads intersection, weights are road distances
  $intersects =$ list of nodes in $world\_graph$ with $edges > 2$
  $map = \textbf{grid\_map}(world\_graph, res)$ // a one-way roads map where graph nodes are connected via road free cells and all the other cells are marked as walls
  $map = \textbf{add\_destination\_wall}(map, \textbf{gps\_to\_cell}(dest\_gps), dest\_orient)$ // add a wall in the cell after the destination position to plan to arrive in the desired lane
  $prev\_cell, next = None$ & $route\_exited = False$
$car\_cell, dest\_cell = \textbf{gps\_to\_map\_cell}(car\_gps, dest\_gps)$
if $car\_cell = dest\_cell$ ***AND*** $car\_compass = dest\_orient$ **then**
  return $GOAL\_REACHED$
if $prev\_cell \neq None$ ***AND*** $car\_cell \neq prev\_cell$ **then**
  $dist = \textbf{route\_distance}(route, car\_cell, route[next])$
  if $dist \leq 1$ **then**
    $prev\_cell = car\_cell$ & $next \mathrel{+}= dist$
  else $route\_exited = True$ // if car_cell is not on $route$
// Re-estimate route when needed
$map, road\_blocked = road\_blockages(map, route, ogm)$ // Run Algorithm 4 to detect road blockages
if $road\_blocked$ ***OR*** $route\_exited$ ***OR*** $prev\_cell = None$ **then**
  $road\_blocked, route\_exited = False$ & $prev\_cell = car\_cell$
  $map = \textbf{add\_car\_wall}(map, car\_cell, car\_compass)$ // add a wall in the cell behind the car current cell
  $route = \textbf{a\_star}(map, car\_cell, dest\_cell)$ // A* from car_cell to $dest\_cell$
  $next = 1$ // cell after vehicle cell index in $route$
  $cmds =$ list of $route.count$ length and $'Follow\_Lane'$ values
  for i ***from*** 0 ***to*** $route.count$ **do**
    if $route[i]$ ***is in*** $intersects$ **then**
      $s = \textbf{normalized\_cross\_product}(\overrightarrow{route[i], route[i+1]}, \overrightarrow{route[i-1], route[i]})$
      if $s < -0.1$ **then** $cmd =' Go\_Right'$
      else if $s > 0.1$ **then** $cmd =' Go\_Left'$
      else $cmd =' Go\_Straight'$
      $cmds[i - far\_inters : i + inter\_exited] = cmd$
for i ***from*** 0 ***to*** $route.count$ **do**
  if $route[i] = car\_cell$ **then**
    $C = cmds[i]$ & **break**

Our planner is different from the planner in [12] in two respects. Firstly, we make the process faster by executing the A* algorithm only when the vehicle exits the latest planned shortest path towards the destination or when detecting a road

blockage along the path. This is more similar to the global route planner in [40] that is carried out at each checkpoint or when facing a road blockage. Secondly, and more importantly, our algorithm provides more accurate commands around road intersections than [12] as shown in the sample in figure 8(a). Figure 8 shows two sample snapshots for the algorithm during deployment from different towns. In sample 8(a), our planner returns a "follow lane" navigational command, while the planner adopted in [12] returns a "go left" navigational command too early. In step 15 in Algorithm 3, the function *road_blockages* executes Algorithm 4 to check if the planned road ahead has any blockages. The function adds walls to the planning map corresponding to detected road blockages. As discussed later in subsection V-B, the added walls are directed; the A* search algorithm decides whether a cell with a directed wall is blocked or free to traverse based on the direction of reaching it from previous cells.

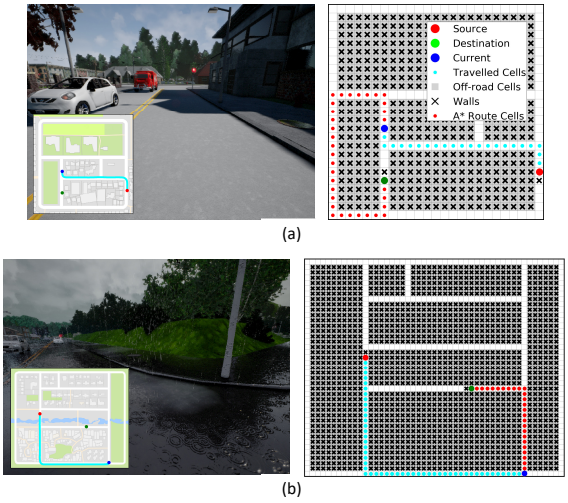


Fig. 8. Two sample snapshots for the proposed global route planning during deployment. The front-facing camera image, the world map, and the planning map are shown for each snapshot. The town in sample 8(b) is larger in area. In sample (a) snapshot, our planner returns a "follow lane" navigational command, while the planner adopted in [12] and [14] returns a "go left" navigational command too early which causes the vehicle to invade the opposite lane. In sample (b), both planners return a "go left" command.

B. OGM-based Road Blockages Avoidance

We updated CARLA simulator to support the insertion of road blockages programmatically as shown in the samples in Figure 1. Algorithm 4 describes our method to avoid road blockages based on OGM that is inspired by a part of our patent in [22]. The algorithm is called by the global route planner Algorithm 3. The inputs are the planning map, an ordered list of planning map cell coordinates of the route towards the destination, and the OGM along with its associated vehicle position information. The cell coordinates are projected and shifted to the lanes' center and are used as control points that are up-sampled and smoothed with a Bezier curve. The OGM is queried for occupancy by the Bezier curve points based on thresholds, if occupancy is detected along the curve, the flag $road\_blocked$ is set to $True$ and the planning map is updated by adding the blocked cell, so

Algorithm 4: Road blockages avoidance algorithm

Input: $planning\_map$: global route planning map
$route$: ordered list of planner route cells
ogm: OGM with vehicle position in it
Parameters: $route\_pts$: the number of cells in $route$ to consider
P_{occ}: OGM occupied cell probability threshold
w: the width and height of the OGM slice around a waypoint
$cell\_occupied\_pts$: minimum number of OGM occupied cells to block a planning map cell
Output: $planning\_map$: planning map updated with added walls if road blockages detected
$road\_blocked$: a flag set to $True$ if a road blockage is detected

$route = route[0 : route\_pts]$ & $road\_blocked = False$
$prev\_cell = route[0]$ // route[0] is vehicle cell
$route\_lane =$ **shift_to_lane**(**cell_to_gps**($route$)) // Project to nearest point middle of the road then shift car lane lane center
$waypoints =$ **smooth_with_bezier**($route\_lane$) // Up-sampled Bezier curve defined by $route\_lane$ control points
$waypoints\_ogm =$ **gps_to_ogm**($waypoints, ogm$)
foreach cell $c \in route[1 : end]$ **do**
  $points =$ list of points in $waypoints\_ogm$ located inside c
  $occ = 0$
  foreach point $p \in points$ **do**
    $OGM_s =$ slice in ogm of width and height w around p
    $occ \mathrel{+}=$ number of cells in OGM_s with value $> P_{occ}$
    if $occ > cell\_occupied\_pts$ **then**
      $blockage\_direction =$ **get_direction**($c, prev\_cell$)
      $planning\_map =$ **add_directed_wall**($planning\_map, c, blockage\_direction$)
      $road\_blocked = True$ & **break**
  $prev\_cell = c$

the global route planner calculates a new $route$ that avoids the detected road blockage and consequently provides the appropriate navigational commands to the model to avoid the blockage. In case of a false positive, the global planner will cause the vehicle to unnecessarily choose a longer route to reach the destination.

The walls added to the planning map due to detected road blockages are directed. The *get_direction* function calculates the direction from four possibilities of right, up, left, and down from the cell that should have the added wall to its preceding cell in $route$. The route planner decides whether the cell with a directed wall is blocked or free to traverse based on the direction of reaching it. The concept of directed walls enables handling partial road blockages where a single lane is occupied while the other lanes could be free to navigate while at the same time leveraging the advantages of planning on a one-way roads map. Figure 1 shows examples for such partial blockages. The workaround of removing the added walls after the vehicle leaves its area, when blockages become irrelevant, instead of having walls directed causes the problem of having the vehicles infinitely looping the same course in some situations as described in Figure 9 scenario. The A* search algorithm was set to prioritize fewer turns when having multiple shortest paths having the same Manhattan distance.

The algorithm considers only the near future waypoints through the parameter $route\_pts$, to save computational power that might be wasted beyond the information in the OGM due to the limited LiDAR range. If $route\_pts$ is set too small, the global planner will detect road blockages too late for the model

to be able to avoid them. The OGM history effect makes it more accurate in near ranges from the vehicle, hence adding walls in cells farther away from the vehicle should be made less likely by making the involved thresholds function of the distance from the vehicle. In addition, based on the OGM and the ego-vehicle current position and speed, a simple rule-based method is employed which determines if the ego-vehicle should slightly reduce or increase the model predicted steering angle to avoid collision with static obstacles. Given the vehicle wheel radius and base, a kinematic bicycle model is utilized to estimate the vehicle position and orientation in near-term future frames. As shown in figure 9 OGM's, the motion model estimates discrete (five in the figure example) vehicle states within three seconds in the future. The discrete trajectory is equidistantly resampled and smoothed giving dense points, and the points within a predefined short distance threshold from the vehicle are considered (named rectified trajectory in the figure). The OGM is queried for being free along the rectified trajectory points. If any points along the trajectory are not free, the steering angle is gradually increased or decreased within a small range until all the trajectory points become possibly free in the OGM. Such steering rectification method is especially useful when the road blockages' visual elements are not well-represented in the model training data.

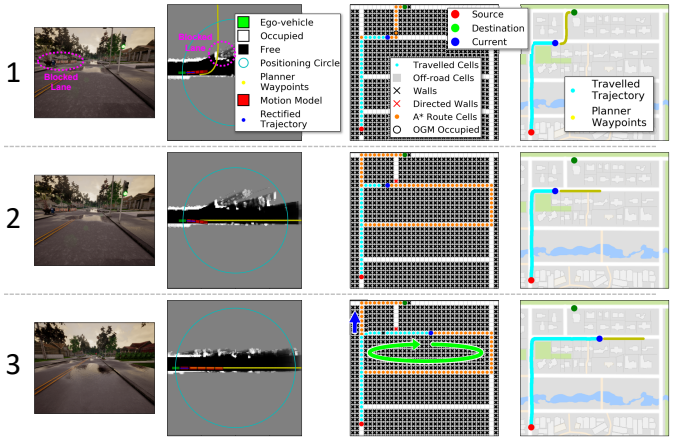


Fig. 9. Three snapshots from a navigation scenario demonstrating the road blockages avoidance method. In the first snapshot, a blockage is detected, and in the second snapshot, the planner successfully re-routes the vehicle to avoid it. If the blockage added wall is directed, the vehicle continues along the route following the blue arrow in the third snapshot route planning map to reach the destination. If the added wall is removed once the vehicle leaves its area, the vehicle will stuck in the navigation loop indicated by the green arrow in case that the shortest path towards the destination is re-evaluated.

VI. Experimental Results

We adopt the experimental setup of the CARLA urban driving benchmark [14] to evaluate the proposed model. The benchmark is composed of four different tasks that are carried out in two towns and six weather conditions. The test town and weather conditions are fully unseen during training. For each combination of a task, a town, and a weather set, testing is conducted over 25 different test scenarios having predefined start and destination locations, this gives a total of 1200 test scenarios for each model under test as described in table III for one town. In the benchmark, each test scenario in the 'Dynamic Navigation' (navigation in traffic) task has 50 pedestrians moving in the driving town and 20 or 15 vehicles for towns 1 and 2 respectively. The parameters for the PID controllers we used during training data recording are tuned in the training town and weathers. Town 1 and 2 have 2.9 km and 1.9 km of drivable two-lane roads in a suburban environment respectively. Both towns include 3-way intersections; 7 and 12 intersections in towns 1 and 2 respectively and do not contain roundabouts.

A test scenario is considered successful if the vehicle reaches the destination within a predetermined deadline. The deadline (maximum allowed time to reach the destination) is set to the time needed to reach the destination along the shortest route at a low speed of 10 km/h as followed in [14] and [10]. A model driving at that low speed has low chances to succeed in a test scenario as it has not to stop due to traffic (for the 'Dynamic Navigation' tasks) or red lights nor to slow down to avoid collisions, while at the same time it has to estimate and follow the shortest route towards the destination without missing a single turn. In addition, such a low-speed deadline allows models that drive at higher, and more reasonable, speeds to succeed in a test scenario even if the shortest route is not followed. The proposed model average vehicle speed over all the benchmark test scenarios was 25.428 km/h compared to 18.697 km/h for the pre-trained model in [12]. The actual vehicle speeds while moving are higher than those two numbers as the averaging includes traffic stopping moments.

A. Success Rate and Distance to Destination

Table IV benchmarks the proposed model before and after LiDAR fusion with the state-of-the-art CIL model [12] on the CARLA urban driving benchmark [14]. The table reports the autonomous driving success rate on different tasks and test conditions, and the average percentage of distance to goal traveled is available between parentheses. The latter metric

TABLE III
IN EACH TOWN IN THE CARLA URBAN DRIVING BENCHMARK [14], THERE ARE 24 EXPERIMENT SETS. EACH SET HAS 25 TEST SCENARIOS (1200 IN TOTAL) REPRESENTING A COMBINATION OF A DRIVING TASK AND A WEATHER CONDITION. "S", "O", "N", AND "DN" STAND FOR "STRAIGHT", "ONE (SINGLE) TURN", "NAVIGATION", AND "DYNAMIC NAVIGATION (ALONG MOVING VEHICLES AND PEDESTRIANS) " TASKS RESPECTIVELY.

Experiment ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Task	S	O	N	DN	S	O	N	DN	S	O	N	DN	S	O	N	DN	S	O	N	DN	S	O	N	DN
Weather Condition	Clear Afternoon (Train)				Wet Noon (Train)				Wet Cloudy Noon (Test)				Hard Rain Noon (Train)				Clear Sunset (Train)				Soft Rain Sunset (Test)			

TABLE IV
DRIVING SUCCESS RATE AVERAGE PERCENTAGE, AVERAGE PERCENTAGE OF DISTANCE TO GOAL TRAVELLED IS BETWEEN PARENTHESES

Task	Model	Percentages of average success rate and distance to goal			
		Training town		New town	
		Training weathers	New weathers	Training weathers	New weathers
Straight	Camera only, [14] results	95 (-)	98 (-)	97 (-)	80 (-)
	Camera only, [14] pre-trained	99 (97.24)	**100 (100.00)**	89 (90.38)	92 (92.70)
	Camera only (trained on our data)	**100 (100.00)**	**100 (100.00)**	99 (95.74)	**100 (100.00)**
	Camera only (trained on our data, no augmentation)	93 (91.46)	88 (87.06)	88 (87.67)	86 (85.71)
	Camera + LiDAR	**100 (100.00)**	**100 (100.00)**	**100 (100.00)**	**100 (100.00)**
	Camera + LiDAR (Our Route Planner)	**100 (100.00)**	**100 (100.00)**	**100 (100.00)**	**100 (100.00)**
Single Turn	Camera only, [14] results	89 (-)	90 (-)	59 (-)	48 (-)
	Camera only, [14] pre-trained	88 (82.71)	94 (85.49)	56 (54.81)	74 (60.60)
	Camera only (trained on our data)	97 (97.33)	98 (97.72)	57 (56.09)	72 (67.18)
	Camera only (trained on our data, no augmentation)	73 (67.29)	72 (65.91)	49 (43.27)	56 (52.57)
	Camera + LiDAR	**100 (100.00)**	**100 (100.00)**	**92 (90.05)**	**92 (91.52)**
	Camera + LiDAR (Our Route Planner)	**100 (100.00)**	**100 (100.00)**	**92 (88.17)**	**92 (88.17)**
Navigation	Camera only, [14] results	86 (-)	84 (-)	40 (-)	44 (-)
	Camera only, [14] pre-trained	78 (88.61)	84 (89.34)	35 (9.68)	58 (45.37)
	Camera only (trained on our data)	87 (91.13)	88 (92.46)	33 (16.92)	34 (16.93)
	Camera only (trained on our data, no augmentation)	65 (64.86)	66 (69.06)	28 (15.02)	25 (14.75)
	Camera + LiDAR	92 (92.70)	92 (92.71)	68 (76.95)	68 (76.86)
	Camera + LiDAR (Our Route Planner)	**96 (96.02)**	**96 (96.04)**	**100 (100.00)**	**100 (100.00)**
Dynamic Navigation	Camera only, [14] results	83 (-)	82 (-)	38 (-)	42 (-)
	Camera only, [14] pre-trained	80 (88.36)	74 (81.53)	28 (17.35)	54 (35.13)
	Camera only (trained on our data)	84 (91.03)	82 (87.34)	26 (9.53)	30 (29.41)
	Camera only (trained on our data, no augmentation)	58 (59.53)	58 (61.41)	24 (11.72)	23 (12.07)
	Camera + LiDAR	86 (93.02)	86 (92.89)	53 (37.51)	64 (59.90)
	Camera + LiDAR (Our Route Planner)	**94 (98.16)**	**96 (98.59)**	**89 (80.88)**	**88 (77.80)**

provides additional insight that cannot be inferred from the success rate. It is not included in the original benchmark, thus we include the results we record from deploying the publicly available CIL pre-trained model. The table includes an ablated model trained without data augmentation.

Table IV benchmark results for the pre-trained model in [14] (”Camera only, [14] pre-trained” model) are correlated to the results reported in [14] (”Camera only, [14] results” model), but are not matching exactly. These discrepancies are due to randomness in evaluation and difference in texture appearance compared to the earlier version of CARLA used in the [14]. Additionally, there are two known sources of non-determinism: 1) textures loading time is not deterministic in the underlying game engine which leads to appearance differences, and 2) the simulator pedestrians algorithms are non-deterministic. The third model in the table (”Camera only (trained on our data)”) trains the same original CIL model from scratch on our dataset. The results are correlated with the preceding two models (”Camera only, ”[14] results” and ”Camera only, ”[14] pre-trained”), but are less accurate especially in the harder tasks and environmental setups, and the problem of generalization is more apparent. That model is trained on our dataset which is automatically collected as detailed in subsection III, while the preceding two models are trained on data collected by a human driver. The ablated model trained without data augmentation performs worst compared to the other models which aligns with the conclusion in [12] that careful data augmentation is crucial for generalization even within the training town. The last two models in the table (”Camera + LiDAR” and ”Camera + LiDAR (Our Route Planner)”) are also trained on our automatically-collected dataset.

The proposed model (”Camera + LiDAR” model) results demonstrate that it performs significantly better than the CIL model [12] in every task and environmental setup combination, even while it is trained on driving data recorded automatically. Both models use the same global route planner algorithm in [14] and [12]. The learned driving policy consistency against varying weather conditions is improved by 3.91 times, as the average (per task and town) success rate difference due to changing from weathers seen during training to unseen weathers becomes 1.375%, while it is 5.375% in case of the ”Camera only, [14] results” model. On the hardest task of ”Dynamic Navigation”, the autonomous driving success rate is improved by 52% when deployed on the new town and weathers that are unseen during training; the success rate improved from 42% to 64%. Succeeding in those ”Dynamic Navigation” test scenarios requires responding to traffic lights. The model is able to recognize and respond to traffic lights as situations involving traffic lights are part of the training data. When our new global route planner is adopted instead, performance is further improved. The autonomous driving success rate is improved by 37%; the success rate improved from 64% to 88%. Figure 8 shows two example snapshots showing the global route planner during deployment. In the first example in figure 8(a), our planner returns a ”follow lane” navigational command, while the planner adopted in [12] returns a faulty ”go left” command causing the vehicle to invade the opposite lane and crash with other vehicles.

B. Infractions Analysis

In table V, we report the average number of kilometers traveled before an infraction for each model on the ”Dynamic Navigation” task test scenarios, the higher the numbers the

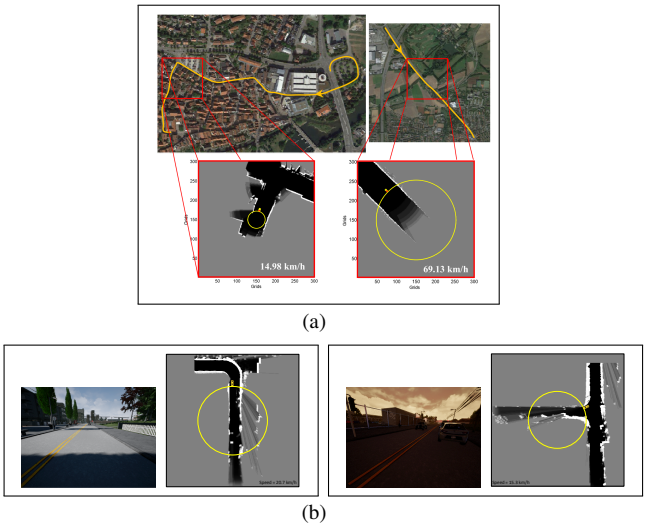
TABLE V
THE AVERAGE NUMBER OF KILOMETRES TRAVELED BEFORE AN INFRACTION

Infractions	Model	Average kilometres traveled before an infraction			
		Training town		New town	
		Training weather	New weathers	Training weather	New weathers
Collision to a Pedestrian	Camera only, [14] pre-trained	7.15	**19.48**	0.99	2.18
	Camera only (trained on our data)	6.08	5.32	1.49	3.14
	Camera + LiDAR	60.96	15.76	25.01	12.60
	Camera + LiDAR (Our Route Planner)	**62.02**	18.79	**25.50**	**16.15**
Collision to a Vehicle	Camera only, [14] pre-trained	1.35	0.89	0.18	0.17
	Camera only (trained on our data)	**1.59**	**1.33**	0.44	0.70
	Camera + LiDAR	1.13	1.09	0.51	0.84
	Camera + LiDAR (Our Route Planner)	1.36	1.20	**1.83**	**2.02**
Collision to a Static Object	Camera only, [14] pre-trained	**5.50**	**5.56**	0.29	0.85
	Camera only (trained on our data)	5.15	2.66	0.26	0.33
	Camera + LiDAR	3.05	3.15	0.54	0.90
	Camera + LiDAR (Our Route Planner)	3.94	3.70	**1.03**	**1.24**
Going Outside of Road	Camera only, [14] pre-trained	14.30	12.98	0.45	0.90
	Camera only (trained on our data)	11.15	7.97	0.59	0.79
	Camera + LiDAR	10.16	10.50	0.96	1.57
	Camera + LiDAR (Our Route Planner)	**64.04**	**32.37**	**2.75**	**3.23**
Invading the Opposite Lane	Camera only, [14] pre-trained	4.77	9.74	0.51	1.69
	Camera only (trained on our data)	13.38	15.95	0.77	0.90
	Camera + LiDAR	8.71	10.50	0.93	1.05
	Camera + LiDAR (Our Route Planner)	**21.35**	**32.37**	**11.00**	**16.15**
Violating Traffic Light	Camera + LiDAR (Our Route Planner)	57.43	53.05	32.88	28.92

better. For the majority of infractions types, the proposed model performed better than the other models especially in the new town unseen during training, which emphasizes the generalization improvement achieved by our model. The model is demonstrated to avoid both dynamic (pedestrian and other vehicles) and static objects. The improvement margin is further increased when our global route planner is adopted. Figure 8(a) shows an example justifying the significant improvement in the rates of invading the opposite lane and going outside the road when using the proposed global route planner. Models trained on data that include situations involving traffic lights are observed to achieve good results on responding to traffic lights. Overall, the infractions analysis is a strong indicator that further progress is still required to produce reliable and safe autonomous driving.

C. Road Blockages Avoidance Benchmark

Figure 10 shows sample real-world and simulation results for our proposed OGM method, with the vehicle position circle shown in yellow. For the real-world results, Valeo ScaLa first-generation LiDAR [45] is used, which is the first laser scanner for automotive volume production. The figure shows two examples out of hours of testing. The laser scanner data for both of the two examples are recorded in Stuttgart, Germany. The first sample is for urban city driving and the second one is for high-way driving. Practically, we found that the *get_affected_area* operation is around five times faster using the convex hull option on average. But the number of grid cells to update is around 2.37 times larger for such an option, which makes it overall around 1.5 times slower than the polygon option. The figure also shows two sample OGM results on CARLA simulator along with the corresponding RGB camera image. In Algorithm 1, log_odd_{occ} and log_odd_{free} are tuned in the training town and set to 0.9 and 0.7 respectively. The resolution is set to 0.5 meters, w is set to 1 meter, and α is set to $2\circ$. The samples shown in figure 10(b) are from the test town demonstrating generalization to new environments. The used 360-degree LiDAR has 32 layers, a vertical field of view from $-30\circ$ to $10\circ$, and a 150 meters range. CARLA LiDAR does not model beam echoes and their pulse width which are not required by our proposed PGV and OGM representations.



(a)

(b)

Fig. 10. Sample real-world and simulation results for our OGM method with the vehicle position circle shown in yellow. (a) Real-world results on urban and highway driving scenarios. (b) Results from CARLA simulator along with the corresponding RGB camera image.

The CARLA urban driving benchmark [14] hardest tasks of navigation and dynamic navigation are adapted to test road blockages avoidance. CARLA simulator is upgraded to support adding random road blockages by following two criteria.

Firstly, each experiment in the benchmark has at least one road blockage and up to five along the shortest route towards the destination. The number of blockages is sampled from a uniform distribution. Secondly, in 50% of the benchmark experiments, there is at least one blockage that mandates dynamically changing the route to reach the destination. In the other 50% of the experiments, the road blockages along the route are partial and do not require rerouting, as an example for being in another lane. For the dynamic navigation task, the simulator autopilot logic is overridden to prevent other vehicles and pedestrians from colliding or being stuck in road blockages. The involved thresholds in the road blockages avoidance algorithm are tuned in the training town. In Algorithm 3, the parameters res, far_inters, $inter\_exited$ are set empirically to 8.215 meters (as in [12]), 4 cells, and 1 cell respectively. In Algorithm 4, $route\_pts$ is set empirically to 5 cells given the used res value in Algorithm 3, while the used OGM width and height are set to 80 meters and the resolution is set to 0.5 meters. The chosen values of those parameters are demonstrated to detect road blockages at around 60 meters ahead in most of the cases as in the examples in figures 9 and 11. The OGM quality, which is directly affected by the weather conditions [46], and occupancy confidence are the main factors determining the blockage detection range. In adverse weather conditions, the range is found reduced to around 30 meters.

Table VI reports the autonomous driving success rates with and without using the proposed road blockages avoidance method, and against the state-of-the-art CIL model [12]. The road blockages avoidance algorithm improved the driving success rate by 27% on the average over all the benchmark tasks and conditions. Table VII reports the average number of kilometers traveled before a collision to a static object on the different condition for the hardest task of "Dynamic Navigation". The table confirms that the introduced road blockage algorithm significantly reduced infractions with static objects. The average kilometers traveled before a collision to a static object increased by more than 1.5 times.

TABLE VI
ROAD BLOCKAGES AVOIDANCE BENCHMARK: DRIVING SUCCESS RATE AVERAGE PERCENTAGE

Task	Model	Percentages of average success rate			
		Training town		New town	
		Training weathers	New weathers	Training weathers	New weathers
Navigation	Camera model in [14]	56	50	32	40
	Camera+LiDAR Model	56	56	44	44
	Camera+LiDAR Model, with road blockages avoidance	**94**	**96**	**72**	**74**
Dynamic Navigation	Camera model in [14]	55	48	29	32
	Camera+LiDAR Model	49	50	45	44
	Camera+LiDAR Model, with road blockages avoidance	**79**	**78**	**57**	**56**

Figure 11 shows an example scenario for our system during deployment in the test town. The ego-vehicle detects two road blockages and dynamically estimates and follows new routes to eventually reach the designation successfully. On average

TABLE VII
ROAD BLOCKAGES AVOIDANCE BENCHMARK: THE AVERAGE NUMBER OF KILOMETRES TRAVELED BEFORE A COLLISION TO A STATIC OBJECT

Model	Average kilometres traveled before an infraction			
	Training town		New town	
	Training weather	New weathers	Training weather	New weathers
Camera model in [14]	0.58	0.56	0.29	0.27
Camera+LiDAR Model	0.6	0.55	0.32	0.34
Camera+LiDAR Model, with road blockages avoidance	**2.05**	**1.73**	**0.69**	**0.52**

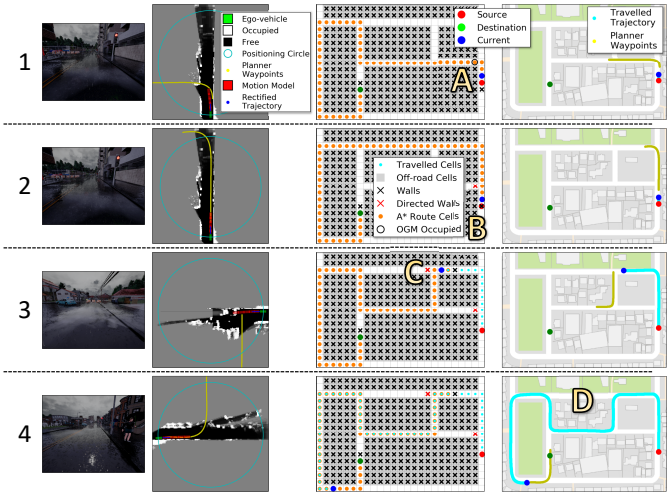


Fig. 11. Snapshots from an example test scenario for the proposed system during deployment in a new town unseen during training. (A) shows the moment a road blockage is detected, then immediately after the vehicle is guided to follow another route to reach the destination as in (B). (C) shows another road blockage detected, and (D) shows that trajectory the vehicle followed by avoiding the two road blockages towards the destination to eventually arrive successfully.

over all the benchmark test scenarios, the overhead of adding the LiDAR to the system, including the PGV and the OGM processing and increasing the model size due to adding the LiDAR input modality, increases the overall system runtime by 68.83%.

VII. CONCLUSION

We proposed a model that extends the state-of-the-art conditional imitation learning method by fusing a LiDAR sensor input with the camera aiming to tackle the challenges of lack of generalization and inconsistency against varying weather conditions. Additionally, we introduced a new efficient Occupancy Grid Mapping method that improves runtime performance, memory utilization, and map accuracy. The OGM is used to upgrade the conditional imitation learning method to dynamically detect partial and full road blockages and guides the controlled vehicle to another route to reach the destination. On CARLA simulator urban driving benchmark, camera and LiDAR fusion is demonstrated to improve weather consistency by around four times. The model has shown to significantly improve the autonomous driving success rate and average distance traveled towards the destination on all the driving

tasks and environments combinations while being trained on automatically recorded data. The generalization to new environments in terms of driving success rate has improved by 52%. The infractions analysis showed improvement as well but overall indicates that further progress is still required to produce reliable and safe autonomous driving.

The global route planner provided more accurate navigational commands and improved the driving success rate further by 37%. CARLA benchmark is upgraded to allow test navigation while having unexpected temporary stationary road blockages. Our road blockages avoidance algorithm improved the driving success rate by 27% and the average kilometers traveled before a collision to a static object increased by more than 1.5 times.

In future work, we need to investigate the proposed model's capability to drive on multi-lane roads and through roundabouts. In addition, knowing whether the generalization issue is caused more by the different road layout or the different environment domains and textures can guide to further improve the model generalization. Moreover, the PGV allows standard CNN to achieve real-time performance; however, the discretization errors can be mitigated by adopting sparse convolution which requires studying the speed-accuracy trade-off.

REFERENCES

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[2] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[3] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 420–425.

[4] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[6] A. Brown, B. Repac, and J. Gonder, "Autonomous vehicles have a wide range of possible energy impacts," NREL, University of Maryland, Tech. Rep., 2013.

[7] P. Cai, Y. Sun, Y. Chen, and M. Liu, "Vision-based trajectory planning via imitation learning for autonomous vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019.

[8] M.-F. Chang *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8748–8757.

[9] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[10] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.

[11] R. Cheng, R. Razani, Y. Ren, and L. Bingbing, "S3net: 3d lidar sparse semantic segmentation network," *arXiv preprint arXiv:2103.08745*, 2021.

[12] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[13] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[15] M. Dupuis, M. Strobl, and H. Grezlikowski, "Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks," in *Proc. of the Driving Simulation Conference Europe*, 2010, pp. 231–242.

[16] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[17] H. M. Eraqi, Y. Abouelnaga, M. H. Saad, and M. N. Moustafa, "Driver distraction identification with an ensemble of convolutional neural networks," *Journal of Advanced Transportation*, vol. 2019, 2019.

[18] H. M. Eraqi, Y. E. Eldin, and M. N. Moustafa, "Reactive collision avoidance using evolutionary neural networks," in *Proceedings of the 8th International Joint Conference on Computational Intelligence - Volume 1: ECTA, (IJCCI 2016)*, INSTICC. SciTePress, 2016, pp. 251–257.

[19] H. M. Eraqi, J. Honer, and S. Zuther, "Static free space detection with laser scanner using occupancy grid maps authors," in *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, October 2017.

[20] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-end deep learning for steering autonomous vehicles considering temporal dependencies," in *Machine Learning for Intelligent Transportation Systems, 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[21] H. M. Eraqi and I. Sobh, "Autonomous driving in the face of unconventional odds," *Communications of the ACM*, vol. 64, no. 4, pp. 64–66, 2021.

[22] H. M. Eraqi, "Occupancy grid mapping-based route planning for work zones avoidance in autonomous driving," December 9 2020, patent Number EP20212769.

[23] H. M. Eraqi and J. Honer, "Resource-saving map for a driver assistance system of a motor vehicle (ressourcensparende karte für ein fahrerassistenzsystem eines kraftfahrzeugs)," May 17 2018, patent Number DE102016122031A1.

[24] L. Fridman *et al.*, "Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, vol. 7, pp. 102 021–102 038, 2019.

[25] 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.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[27] German Ros, Vladlen Koltun, Felipe Codevilla, and Antonio M. Lopez, "Carla autonomous driving challenge 2019 results," 2019. [Online]. Available: https://carlachallenge.org/results-challenge-2019/

[28] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 2432–2437.

[29] A. Guanella, D. Kiper, and P. Verschure, "A model of grid cells based on a twisted torus topology," *International journal of neural systems*, vol. 17, no. 04, pp. 231–240, 2007.

[30] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[31] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art," *arXiv preprint arXiv:1704.05519*, 2017.

[32] M. Jokela, M. Kutila, and P. Pyykönen, "Testing and validation of automotive point-cloud sensors in adverse weather conditions," *Applied Sciences*, vol. 9, no. 11, p. 2341, 2019.

[33] A. B. Jung *et al.*, "imgaug," 2020, online; accessed 01-Feb-2020. [Online]. Available: https://github.com/aleju/imgaug

[34] J. Jung, E. Che, M. J. Olsen, and C. Parrish, "Efficient and robust lane marking extraction from mobile lidar point clouds," *ISPRS journal of photogrammetry and remote sensing*, vol. 147, pp. 1–18, 2019.

[35] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 399–404.

[36] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] G. K. Kraetzschmar, G. P. Gassull, K. Uhl, G. Pags, and G. K. Uhl, "Probabilistic quadtrees for variable-resolution mapping of large environments," in *Proceedings of the 5th IFAC/EURON symposium on intelligent autonomous vehicles*. July, 2004, pp. 1–6.

[38] C. Li, D. Song, R. Tong, and M. Tang, "Illumination-aware faster r-cnn for robust multispectral pedestrian detection," *Pattern Recognition*, vol. 85, pp. 161–171, 2019.

[39] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, 2013, p. 3.
[40] M. Montemerlo *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.
[41] M. Nieuwenhuisen, D. Droeschel, M. Beul, and S. Behnke, "Obstacle detection and navigation planning for autonomous micro aerial vehicles," in *2014 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2014, pp. 1040–1047.
[42] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1672–1679.
[43] S. Thrun, "Learning occupancy grids with forward models," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3. IEEE, 2001, pp. 1676–1681.
[44] ——, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[45] Valeo, "Valeo Scala," 2020, https://www.valeo.com/en/valeo-scala/, accessed 2020-02-02.
[46] A. M. Wallace, A. Halimi, and G. S. Buller, "Full waveform lidar for adverse weather conditions," *IEEE transactions on vehicular technology*, vol. 69, no. 7, pp. 7064–7077, 2020.
[47] WHO, "Global status report on road safety 2018," *World Health Organization*, 2018. [Online]. Available: https://apps.who.int/iris/bitstream/handle/10665/276462/9789241565684-eng.pdf
[48] S. Wirges, T. Fischer, C. Stiller, and J. B. Frias, "Object detection and classification in occupancy grid maps using deep convolutional networks," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3530–3535.
[49] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.
[50] H. Yang, K. Ozbay, O. Ozturk, and K. Xie, "Work zone safety analysis and modeling: a state-of-the-art review," *Traffic injury prevention*, vol. 16, no. 4, pp. 387–396, 2015.
[51] X. Zhou, Y. Gao, and L. Guan, "Towards goal-directed navigation through combining learning based global and local planners," *Sensors*, vol. 19, no. 1, p. 176, 2019.