Vision-based Autonomous Driving for Unstructured Environments Using Imitation Learning
Joonwoo Ahn, Minsoo Kim, and Jaeheung Park

Abstract: Unstructured environments are difficult for autonomous driving. This is because various unknown obstacles are laid in drivable space without lanes, and its width and curvature change widely. In such complex environments, searching for a path in real-time is difficult. Also, inaccurate localization data reduce the path tracking accuracy, increasing the risk of collision. Instead of searching and tracking the path, an alternative approach has been proposed that reactively avoids obstacles in real-time. Some methods are available for tracking global path while avoiding obstacles using the candidate paths and the artificial potential field. However, these methods require heuristics to find specific parameters for handling various complex environments. In addition, it is difficult to track the global path accurately in practice because of inaccurate localization data. If the drivable space is not accurately recognized (i.e., noisy state), the vehicle may not smoothly drive or may collide with obstacles. In this study, a method in which the vehicle drives toward drivable space only using a vision-based occupancy grid map is proposed. The proposed method uses imitation learning, where a deep neural network is trained with expert driving data. The network can learn driving patterns suited for various complex and noisy situations because these situations are contained in the training data. Experiments with a vehicle in actual parking lots demonstrated the limitations of general model-based methods and the effectiveness of the proposed imitation learning method.

Keywords: Vision-based Autonomous driving, Imitation learning, Data Aggregation (DAgger) Algorithm

1. INTRODUCTION

Autonomous driving technology for unstructured environments such as parking lots and alleyways is important to realize fully autonomous driving. Also, it is more difficult than driving in structured environments. In a structured environment, autonomous driving involves a global plan with a road network, and a vehicle stays within a lane through lateral control and maintains a safe distance from vehicles in front while following a target speed through longitudinal control. However, applying this method to unstructured environments is difficult as drivable space has no lanes and variable width. Moreover, the curvature can rapidly change, such as at right-angled corners, and the drivable space can be narrowed because of double-parking or illegal parking. Other obstacles include vehicles, humans, curbs, and bollards, which vary in shape, size, and location. Such obstacles typically are unknown in advance.

For a vehicle to drive in an unstructured environment, a global path is generated on a global map to reach the destination. The vehicle tracks the path based on localization data (i.e., the position and heading of the vehicle relative to the path) [1]. While tracking the global path, the vehicle checks for obstacles in its vicinity. Object detection algorithms detect the position and size of obstacles by using camera or LiDAR sensors with pattern recognition or deep learning. If obstacles are detected near the global path, motion-planning is performed to find a local path or waypoint that can reach the global path without collision. The planned solution must also satisfy dynamic and kinematic constraints on the motion of the vehicle. Motion-planning algorithms developed for robotics have been applied to autonomous vehicles [2, 3]. These can be categorized according to the method and calculation time. Figure 1 shows an overview of motion-planning algorithms, which include optimization, graph search, and incremental search path planning methods to find a solution for the local area.

The path planning method using optimization theory, such as model predictive control (MPC) [4] and convex optimization [5], uses a kinematic and dynamic model of the vehicle to predict its future trajectory. This method provides an optimal solution that satisfies the objective function and constraints. In driving situations, the objective function can be modeled as avoiding obstacles while reaching the global path and maintaining target speed.
Constraints can be the control capabilities and maintaining a safe distance from obstacles.

The graph search path planning method builds a graph in the local area and then searches for a path. The Voronoi diagram [6], Visibility graph [7], and Probabilistic roadmap (PRM) [8] algorithms can be used to build the graph. These algorithms discretize the configuration space into obstacles and free space, which are represented in the form of a graph. Then, the graph is searched for the minimum path length with the Dijkstra [9] or A* [10] graph search algorithm. The searched path is interpolated through spline algorithms to satisfy vehicle constraints and obtain a smooth path.

The incremental search path planning method uses tree exploration algorithms. These algorithms iteratively expand a tree into free space until the end of the tree reaches a goal. The rapidly-exploring random trees’ (RRT*) algorithm [11] extends the tree with samples randomly selected in the configuration space. The hybrid-A* [12] and anytime-D* [13] algorithms expand the tree in grid units. Then, the path with the minimum length is searched for to reach the goal pose while satisfying the non-holonomic constraints of the vehicle.

However, these methods have three problems [14]. First, if the local area is large or complex, a long computational time is needed to generate the path, and the solution may not be found within a control loop (i.e., not real-time). Second, selecting a goal in the global path to search for the local path is heuristic. Third, when an algorithm is implemented, accurately recognizing whether an obstacle is near the global path and tracking the path without collision are difficult because of inaccurate localization data. In unstructured environments, various types of obstacles are complexly placed in the drivable area. Thus, obtaining accurate localization data at every point in unstructured environments is difficult.

Instead of searching and tracking a path, alternative methods can be used, including candidate path selection and the artificial field generation [14]. These methods find a solution near a vehicle that can be calculated in real-time. They select a candidate path or waypoint and calculate the control commands. The vehicle can then drive toward the global path while reactively avoiding obstacles.

The candidate path selection method generates candidate paths and selects one path that satisfies multiple objectives. These paths are smooth and are designed to account for the non-holonomic constraints of the vehicle. To select one path, the objective function is modeled to reach the global path, avoid obstacles, and keep a ride comfort. Typically, three algorithms have been used: the Dynamic window approach (DWA) [15], Curvature velocity method (CVM) [16], and tentacle [17] algorithms. The DWA algorithm designs a window according to the current state of the vehicle, and candidate paths are generated within the window. The CVM algorithm is similar to DWA and additionally considers accelerations of the vehicle. The tentacle algorithm mimics the antennas of a beetle as candidate paths to drive on narrow and variable-curvature roads more smoothly than DWA and CVM.

The artificial field generation method uses a repulsive field against obstacles and attractive field toward the global path. These fields are combined with different weights, and a vehicle is guided by the combined field’s vector. Three algorithms are available that differ on how to model the fields: the virtual force field (VFF) [18], artificial potential field (APF) [19], and velocity vector field (VVF) [20] algorithms. The VFF algorithm calculates the repulsive force as a vector from the obstacle to the vehicle and the attractive force as a vector from the vehicle to the target point. The APF algorithm creates a repulsive field with high potential energy for obstacles and an attractive field with high energy at the vehicle point and low energy at the goal point. Then, the field is generated from the gradient of the potential energy. The VVF algorithm considers the desired velocity and velocity of obstacles in addition to the fields of the APF algorithm.

However, the candidate path selection and artificial field generation methods have several problems that make them
difficult to be used in unstructured environments [21, 22]. First, the parameters (weights) in the objective function or field model may differ to cope with the various complex situations of unstructured environments. It is not easy to find specific parameters that can handle all of these situations. Second, inaccurate localization data make it difficult in practice to know where exactly the global path is located in a local area [23]. Third, if the local obstacle information is difficult to recognize accurately especially at road boundaries or shadowed areas (i.e., noisy state), the vehicle may not drive smoothly [24]. In addition, a vehicle may drive out of the drivable space or toward an obstacle.

1.1. Overview of Our Approach

To address these problems, this study proposes a method of selecting a waypoint (look-ahead point) to drive toward the drivable space while avoiding obstacles in real-time without the use of global information\(^1\) such as the global map, global path, and localization data. The proposed method only segments the drivable space and non-drivable space around a vehicle; it does not recognize whether obstacles exist near the global path. The segmented space is represented as an occupancy grid map, which is obtained by deep learning to segment the image acquired from the camera. The motion-planning algorithm in the proposed method is based on deep learning [25], which is an alternative to general motion-planning algorithms [21, 26]. It can handle the various complicated situations that occur in unstructured environments without requiring the model parameters to be tuned, and it is robust against noisy obstacle information.

In the motion-planning, training data are collected of state action pairs without the use of global information, and the data are used to train a deep neural network. In the data, states contain various situations that are difficult to handle with general motion-planning algorithms, such as large changes of the curvature and width of the drivable space, and noisy state. Using data, the network can learn patterns for all the information of the occupancy grid map, not just information around the candidate paths or artificial fields. Depending on how a dataset is collected, there are reinforcement learning and imitation learning.

For reinforcement learning, data are collected to maximize a reward. However, heuristics are required to model the reward function to achieve an objective [27]. In addition, the agent gathers data through trial and error with random actions, so the training takes a long time to complete. Therefore, training in a real environment is difficult, and most studies on reinforcement learning requires the use of simulations.

In this study, imitation learning was used to collect data effectively. Imitation learning collects successful driving data obtained by experts [28]. These data can be used to train the deep neural network in an approach similar to supervised learning. The trained network can imitate the driving patterns of expert driving. Therefore, a heuristic model is not needed to evaluate an action. Because the data are collected without trial and error, less training time is required than for reinforcement learning, and data of real environments are easily obtained.

In the proposed method, imitation learning is used to select a look-ahead point with the occupancy grid map as the input. The pure pursuit algorithm is used to calculate the steering angle to reach the look-ahead point, and velocity according to the longitudinal distance between the look-ahead point and vehicle. The dataset aggregation (DAgger) algorithm [29] is used to address unsafe and near-collision situations occurred by the trained network policy. By using the look-ahead point, DAgger can be applied to the real autonomous vehicle. In addition, we propose a method to train these problem situations faster and more accurately than EnsembleDAgger.

Our contributions are summarized as follows:

- A method is proposed to drive with only vision data in unstructured environments using imitation learning, which does not use high-cost HD-map and inaccurate localization data in a complex environment.
- Real-world experiments show limitations of the model-based motion-planning algorithms and effectiveness of the proposed method which is robust to sensor noise and does not need to tune model parameters to handle various and complex environments.
- As features and innovations, the vision information is converted into the occupancy grid map in order to apply imitation learning to this information. Besides, the look-ahead point of the pure pursuit algorithm, which has been widely used in autonomous driving, was used to train, which has a clear state action pattern relationship, and a safe driving policy is obtained.

Moreover, DAgger algorithm is introduced to further improve the performance of imitation learning, and (DAgger) can be applied to the real autonomous vehicle by using the look-ahead point.

The rest of this paper is organized as follows. Section 2 explains the vision-based occupancy grid map and imitation learning-based driving policy. The experimental setup and results are presented in Section 3 and 4. In the experiment with an autonomous vehicle, the tentacle algorithm, VVF algorithm, and proposed method were tested in real parking lots with the same input (i.e., occupancy grid map). The experimental results demonstrated that the vehicle was successfully driven with the proposed imitation learning method in situations where the tentacle and VVF algorithms encountered problems. Section 5 concludes this study and mentions future work.

\(^{1}\)The proposed method does not use global information, assuming it only handles environments without intersections.
2. METHODS

This section presents the methods for obtaining the occupancy grid map from vision data and the driving policy in an unstructured environment through imitation learning. The input for imitation learning is the occupancy grid map, and the output is the look-ahead point used to control the vehicle. In this study, the road was assumed to have only static obstacles and no intersections.

2.1. Vision-based Occupancy Grid Map

The occupancy grid map is a 2D map that divides an area into a grid. It is shown in the red box in the upper right of Fig. 2. Each grid in the map contains information on whether it is occupied (non-drivable) or unoccupied (drivable). It serves as the input for imitation learning, and it can be used for the tentacle and VVF algorithms as well.

Driving policies using the occupancy grid map have two advantages. First, the segmented image can ignore irrelevant information for driving, such as differences in the types of obstacles and pavement in the drivable space. Therefore, driving policies can achieve similar performance in untrained environments, which can enhance the generality of driving performance. Second, close and far distance information can be clearly distinguished because the occupancy grid map is a 2D map (i.e., bird’s-eye-view). Thus, depending on the situation, the vehicle can avoid nearby obstacles preferentially or consider distant obstacles in advance.

A camera is used to recognize the drivable and non-drivable spaces. The obstacle detection performance of ultrasonic and 2D-LiDAR sensors depends on the height that they are attached to the vehicle. With a 3D-LiDAR sensor, the point clouds provide a wide range of height information. However, this sensor is expensive and requires high computational cost and memory capacity. In contrast, a camera is compact and inexpensive, and it has lower computational and memory costs. In addition, more training data are available for deep learning with vision than with 3D-LiDAR; more data generally help increase the recognition performance.

The upper side of Fig. 2 illustrates the method for obtaining the occupancy grid map from a camera. The distortion of the front view camera image is corrected by using intrinsic and extrinsic parameters, but slight distortion remains in the side of the image. Nevertheless, a trapezoid area of this image has little distortion and is transformed into a bird’s-eye-view image through the warp perspective function of the OpenCV library. The transformed image is segmented into the drivable space and non-drivable space with a deep neural network through semantic segmentation, which refers to the process of linking each pixel to a class label. The following paragraph describes this network in detail. The road, crosswalk, and road marks are labeled as the drivable space. The outside area excluding the drivable space is considered non-drivable space. The road boundary lines, sidewalks, parking spaces (including parking lines), pedestrians, and vehicles are also labeled as the non-drivable space. The 200×200 segmented image is divided into 82 pixels per one grid to obtain a 25×25 grid map. If all pixel values inside each grid are non-occupied, the grid is regarded as non-occupied.

The perception network is similar to the segmentation task of MultiNet [30], which is based on the U-Net structure. It consists of an encoder and decoder based on a convolutional neural network (CNN). The encoder is the same as that of the VGG network [31] except for the last layer. It consists of five pairs of convolutional and max-pooling layers, which is used to extract several abstract features from the input image. Then, one 1×1 fully-connected layer is connected at the end. The structure of the decoder
follows that of the mainstream fully convolutional network. The output of the encoder is passed through a $3 \times 3$ convolutional layer and up-sampled with three transposed convolution layers. At this time, each convolutional layer of the encoder is combined with the decoder through the skip connections to extract high-resolution features from the encoded low-resolution features.

2.2. Imitation Learning for Autonomous Driving in Unstructured Environment

Imitation learning involves imitating the behavior of an expert for a certain state. State action pairs of data are collected while an expert is driving. The policy $\pi_{net}$ (i.e., deep neural network) is trained with the data in a process called behavior cloning, which is a basic training step of imitation learning [28]. To address the limitations of behavior cloning, DAgger [32] is used to collect additional data by executing the trained behavior cloning policy and retraining $\pi_{net}$. This process is repeated until the best policy is obtained. The following subsections describe the composition and collection of the dataset, behavior cloning, and the DAgger algorithm.

2.2.1 Dataset

The dataset consists of state and action pairs $D = \{(s_t, a_t)\}_t$, where $t$ is an index of the data. The state $s_t$ is the occupancy grid map ($25 \times 25$ grid $\in \{0$ (black): drivable(unoccupied), 1 (white): non-drivable(occupied)$\}$). It is used for the input of the policy $\pi_{net}$.

The action $a_t$ is a command of an expert and the mean value of the output of $\pi_{net}$. In this study, the look-ahead point $a_t$ was used as the action $a_t \in \{a_0, a_1, \ldots, a_i\}$, which is the target waypoint for a vehicle to reach. Most autonomous driving studies based on imitation learning use the steering-accel/brake as the action, but the look-ahead point is more useful for executing the proposed DAgger algorithm. This is explained in detail in Section 2.2.5. The output of the policy $\pi_{net}$ for a state is expressed as:

$$a_{net,t} = \pi_{net}(s_t), \tag{1}$$

which consists of $a_{net,t} \in \{a_{net,0}, a_{net,1}, \ldots, a_{net,i} \}$, where $\sigma_{a_{net,0}}, \ldots, \sigma_{a_{net,i}}$ are the mean and variance of the look-ahead point. The variance of the look-ahead point is calculated through a Gaussian process (GP) to quantify the uncertainty or confidence of $\pi_{net}$ [33].

To collect training data, the expert selects the look-ahead point $a_{exp,t} \in \{a_{exp,0}, a_{exp,1}, \ldots, a_{exp,i} \}$, and the vehicle is controlled to reach the selected look-ahead point in real-time. The steering angle command is calculated with the pure pursuit algorithm [34]. The velocity command is proportional to the distance between the look-ahead point and the vehicle. As the vehicle is driving, the dataset $D = \{(s_t, a_{exp,t})\}_t$ is stored for every period $t$, and numerous data can be collected easily. This process is repeated continuously until the driving is completed.

As illustrated in Fig. 3, the expert selects the look-ahead point $a_{exp,t}$ by using a mouse pointer in the combined image $x_t$ instead of the state $s_t$ (i.e., occupancy grid map):

$$a_{exp,t} = \pi_{exp}(x_t), \tag{2}$$

where $\pi_{exp}$ indicates the behavior of the expert. The combined image $x_t$ is an image of transparently combining the information of the drivable space to the RGB image: $x_t \in \{RGB$ with green: drivable, $RGB$ only: non-drivable$\}$. This is because, if $s_t$ is inaccurate (i.e., noisy), the expert may incorrectly select the look-ahead point [35]. This situation is shown in Figs. 3(b), 4(b), and 11.

The look-ahead point has a geometric relationship with the combined image $x_t$, and the expert $\pi_{exp}$ selects the look-ahead point $a_{exp,t}$ by referring to three criteria (rules):

(i) The look-ahead point must be within the drivable space.

(ii) When an obstacle is in front of the vehicle, the expert selects a look-ahead point for which obstacle avoidance is possible.

To easily check that selected the look-ahead point follows this criterion, the expert can refer to a future trajectory that the vehicle will drive along during a certain time. The look-ahead point is selected with the fewest obstacles around the trajectory. This trajectory is obtained by using the kinematic model of the vehicle, indicated in Figs. 3 and 4(b).

(iii) If there is no obstacle in front of the vehicle, the look-ahead point is selected as far as possible from the vehicle within the drivable space.
Based on these rules, the vehicle can avoid obstacles and drive toward the drivable space as fast as possible. For example, if an obstacle exists on the front and left side of a vehicle, the look-ahead point is selected to be on the right and near the front side of the vehicle in the drivable space (see Fig. 3(a)). With this point, a large steering angle and low-velocity command are calculated, and the vehicle can safely avoid obstacles. Conversely, if there are no obstacles, the look-ahead point is chosen as far as possible from the vehicle in the drivable space (see Fig. 3(b)). With this point, the vehicle can drive at high speed with small steering angle changes.

2.2.2 Behavior Cloning

The collected data can be used to train the policy \( \pi_{\text{net}} \) in a process similar to that of supervised learning. \( \pi_{\text{net}} \) is expressed as \( \pi_{\text{net}}(s_t; \theta) \) parameterized by \( \theta \) for the state \( s_t \). The process of optimizing \( \theta \) to minimize the loss function \( L \) is the process of training \( \pi_{\text{net}}(s_t; \theta) \). The loss \( L \) is the difference in the state \( s_t \) between the output of \( \pi_{\text{net}}(s_t; \theta) \) and the action in the dataset \( a_{\text{exp},t} \). This is expressed as \( L(\pi_{\text{net}}(s_t; \theta), a_{\text{exp},t}) \) and its detailed expression is given in (4). A large number \( T \) of datasets \( D = \{(s_t, a_{\text{exp},t})\}_{t=1}^{T} \) is used to optimize \( \theta \). This training process is called behavior cloning and is expressed as follows:

\[
\min_{\theta} \sum_{t=1}^{T} L(\pi_{\text{net}}(s_t; \theta), a_{\text{exp},t}). \tag{3}
\]

The trained policy \( \pi_{\text{net}} \) can minimize the loss \( L \) with behavior cloning policy denoted by \( \pi_{\text{BC}} \). When a vehicle drives with \( \pi_{\text{BC}} \) in an environment similar to the trained environment, \( \pi_{\text{BC}} \) can calculate an action similar to that of the expert.

On the other hand, if \( \pi_{\text{BC}} \) encounters states that are not similar to the dataset \( D \) or are noisy, \( \pi_{\text{BC}} \) may produce unsafe or unsafe actions. The noisy state is when the boundary of the drivable space or shadow area is not accurately recognized. This result is shown in Fig. 4(b) and is the result of executing \( \pi_{\text{BC}} \) on different days in a place used to collect dataset for behavior cloning. However, the location or type of obstacle differs from when the dataset was collected for \( \pi_{\text{BC}} \). In this case, the vehicle cannot sufficiently avoid obstacles; this is called the data mismatch problem. When a vehicle enters a narrow corner, if the non-drivable space (obstacle) is erroneously recognized as the drivable space, the look-ahead point can be placed on the non-drivable space, and the vehicle will collide with the obstacle. This occurs when the data for these situations are included in the training dataset \( D \) less often than situations of driving in a relatively large drivable space or with no misrecognition problems. Thus, the policy \( \pi_{\text{BC}} \) cannot reflect these situations in \( \pi_{\text{BC}} \) well; this is called the data imbalance problem. Moreover, when these problems occur in a driving situation, the error may magnify afterward because \( \pi_{\text{BC}} \) has not learned recovery behavior; that is called the compounding error problem.

2.2.3 DAgger Algorithm

The DAgger algorithm can be used in imitation learning to address the problems of behavior cloning [32]. DAgger aggregates an additional dataset \( D_i \) with the previously collected dataset \( D \) and trains the policy \( \pi_{\text{net},i} \) again. This process is repeated until the desired policy is obtained. DAgger is explained in detail in Algorithm 1.

First, DAgger initializes the policy \( \pi_{\text{net},i=1} \) and dataset \( D \) as those obtained by behavior cloning. The DAgger iteration \( i \) and \( \tilde{\pi}_i \) representing the performance of the trained policy \( \pi_{\text{net},i} \) are initialized. When the iteration is started (\( i = 1 \), line 6), the additional dataset \( D_i \) is collected by the data-sampling function as described in the next subsection. The data-sampling function checks whether an unsafe or near-collision situations occurs. When it occurs,
the expert action is used to control the vehicle, and the state and action are collected in the additional dataset $D_i$ (see Fig. 4(b)). Otherwise, the action of $\pi_{net,i}$ is used to control, and the additional dataset is not gathered.

After the driving takes place, the collected additional dataset $D_i$ is aggregated to the existing dataset $D$ (line 7). The aggregated dataset $D$ is used to retrain the policy $\pi_{net}$ with (3) (line 8). After training, a policy $\pi_{net,i+1}$ that can cause fewer unsafe or near-collision situations than $\pi_{net,i}$ can be obtained. As more data of these problem situations are aggregated, $\pi_{net,i}$ becomes more capable of dealing with these situations, which is proven in [32]. DAgger repeats this process until these problem situations rarely happen (line 5). This can be judged by $\hat{\eta}$ (line 19 of Algorithm 2) which is the ratio of executed network actions among the total executed actions. If $\hat{\eta}$ is greater than the threshold $\eta$, the iterations of DAgger are terminated. Finally, a policy $\pi_{net,i}$ that does not cause unsafe or near-collision situations is obtained (line 11).

2.2.4 Data-sampling Function in DAgger

To collect the additional dataset $D_i$ and judge the performance of the trained policy, the data-sampling function determines whether to use the trained policy or expert behavior depending on the driving situation. To imitate the additional dataset more precisely and quickly, the error dependence on the driving situation. To imitate the training dataset more

\textbf{Algorithm 1:} Pseudo-code of DAgger Algorithm

```plaintext
1 function DAgger($\pi_{BC}, D_{BC}$)
2 Initialize $\pi_{net,1} \leftarrow \pi_{BC}$
3 Initialize $D \leftarrow D_{BC}$
4 Initialize $i \leftarrow 1, \hat{\eta} \leftarrow 0.0$
5 while $\hat{\eta} \leq \eta$ do
6   $D_i, \hat{\eta} \leftarrow \text{Sample unsafe or near-collision}$
7   Aggregate datasets $D \leftarrow D \cup D_i$
8   Train policy $\pi_{net,i+1}$ on $D$ using Data-sampling Function($\pi_{net,i}$)
9   $i \leftarrow i + 1$
10 end
11 return $\pi_{net,i}$
```

variance of $\pi_{net,i}$ is obtained: $\hat{\chi}$ (line 8). As shown in Fig. 4(a), by checking whether $\hat{\chi}$ or $\hat{\chi}$ is larger than threshold values $\tau$ or $\chi$, unsafe or near-collision situations can be identified.

As shown in Fig. 4(b), in all three situations, $\hat{\tau}$ is greater than $\tau$ (blue circle). In the rightmost case, $\hat{\chi}$ (red lines) is also greater than $\chi$ (blue lines). In these cases, if the vehicle follows the action of the network (red circle), the distance between the vehicle and obstacle decreases, and the possibility of collision increases. To avoid unsafe situations in these cases, the action of the expert behavior (yellow circle) is used to control the vehicle (line 14). At the same time, only the state $s_i$ of this situation and the expert action $a_{exp,i}$ are collected to the additional dataset $D_i$ (line 15). This is to train the network intensively to overcome unsafe and near-collision situations.

By using the criteria for $\hat{\tau}$ and $\hat{\chi}$ (line 9), the states with unsafe or near-collision situations can be collected as much as possible within a range where the vehicle does not collide with obstacles. If the expert judges these situations heuristically without using these criteria, these problem states cannot be sufficiently collected. This is because experts prefer to avoid these situations immediately, so they are difficult to experience them. In the next iteration $i + 1$, these situations can be handled better with a larger dataset containing these problem situations, in contrast to when the criteria are not used.

The discrepancy between actions $\hat{\tau}$ is then added to the additional dataset $D_i$ to imitate the training dataset more

\textbf{Algorithm 2:} Pseudo-code of Data-sampling Function in DAgger Algorithm

```plaintext
1 function Data-sampling Function($\pi_{net,i}$)
2 Initialize $D_t \leftarrow \emptyset$
3 Initialize $n_{tot} \leftarrow 0, n_{net} \leftarrow 0$
4 for $t = 0$ to End of Execution do
5   $\hat{\tau}_{net,i}, (\sigma_{net,i}^2, \tau_{net,i}) \leftarrow \pi_{net,i}(s_t)$
6   $a_{exp,i} \leftarrow \pi_{exp}(x_t)$
7   $\hat{\tau}_t \leftarrow ||\hat{a}_{net,i} - a_{exp,i}||_2$
8   $\hat{\chi}_{i,t} \leftarrow \sigma_{net,i}^2$
9   if $\hat{\tau}_t < \tau$ or $\hat{\tau}_t < \chi$ or $\hat{\chi}_{i,t} < \chi$ then
10      Control the vehicle with $a_{net,i}$
11      $n_{net} \leftarrow 1$
12   end
13 else
14      Control the vehicle with $a_{exp,i}$
15      $D_t \leftarrow \text{append}(\{s_t, a_{exp,i}, \hat{\tau}_t\})$
16 end
17 $n_{tot} \leftarrow 1$
18 end
19 $\hat{\eta}_t \leftarrow \frac{n_{net}}{n_{tot}}$
20 return $D_t, \hat{\eta}_t$
```
precisely (line 15), $\tau_t$ is reflected in the training process as a weight (gain) for the loss function $L$, as expressed in (6). This updates the parameters of the network as much as the network generates the error ($\hat{x}_t$), which can reduce the possibility of the same mistake being repeated. Therefore, at the next iteration $i + 1$, DAgger can reduce $\tau_t$ more with the weight than without it. Thus, the final policy $\pi_{net,i}$ can be obtained with fewer DAgger iterations.

### 2.2.5 Reasons to Use Look-ahead Point As Action

If the action is the steering-accel/brake, the expert suffers two problems in executing the DAgger algorithm, and these can be addressed by using the look-ahead point.

First, as shown in lines 5 and 6 of Algorithm 2, the network action and expert behavior should be obtained simultaneously. However, if the steering-accel/brake is used as the action, an expert action cannot be obtained at the same time when the vehicle is being controlled by a network action. On the other hand, because the proposed method uses the look-ahead point as the action, the expert can select the look-ahead point with only a mouse pointer on the combined image $x_t$ regardless of the network action.

Second, even if the action is set as the steering-accel/brake and the expert action can be obtained simultaneously with the network action, the expert cannot clearly find a steering-accel/brake value that the vehicle can drive as safe and fast as possible when performing DAgger. This is because, when the vehicle is controlled by the network and expert intervention is needed, the expert cannot calculate an action value considering the current network action used for vehicle control. When humans drive, they do not directly calculate an absolute steering-accel/brake value, but calculate how much more or less rotate the steering angle and press the accel/brake pedals from the current steering-accel/brake (i.e., amount of change).

In this study, the expert selects the look-ahead point that the vehicle will reach on the combined image $x_t$ by referring to the three criteria mentioned in the previous subsection (2.2.4). These criteria specify where the look-ahead point is chosen for $x_t$ by its geometric relationship. Thus, the expert can clearly find one look-ahead point that the vehicle can drive as safe and fast as possible without the current steering-accel/brake feedback of the vehicle controlled by the network. This enables a state-action relationship (pattern) to be clearly identified, so a neural network can learn the driving pattern more clearly.

### 2.2.6 Driving Policy Network

The deep neural network is used as the policy $\pi(s_t; \theta)$, which is illustrated in the driving policy network of Fig. 2. It consists of the encoder with the CNN and fully connected layers. The encoder is composed of two pairs of convolutional and max-pooling layers, and the flattened layer nodes are connected. Then, the two fully connected layers with 1000 and four nodes are linked at the end. The last layer with four nodes has the mean and variance for $x$ and $y$ of the look-ahead point.

The loss function $L(\pi_{net}(s_t; \theta), a_{exp,j})$ in (3) is the multivariate Gaussian log-likelihood loss function (see (4)). This allows the network to infer the mean and variance of the Gaussian distribution for the output [33].

$$L = \frac{1}{n} \sum_{j=x,y} 1 \frac{1}{2} r_j^T K_{r_j}^{-1} r_j + \frac{1}{2} \log |K_{r_j}|,$$  

(4)  

where the first term penalizes wrong predictions. The second term predicts model complexity and penalizes it. $n$ is the dimension of the action, and the look-ahead point has two dimensions: the $x$ and $y$ axes. The variable $diff_j$ is the difference between the expert action $a_{exp,j}$ in the dataset and the predicted action by the network $\pi(s_j; \theta)$ which is the network output being trained for $s_j$:

$$diff_{j \in \{x,y\}} = |a_{exp,j} - \pi(s_j; \theta)|.$$  

(5)  

In addition, the weight (gain) $\tau_t$ is multiplied by $diff_j$ for more effective learning [36], which is described in the previous subsection (2.2.4):

$$r_j = (1 + \alpha \tau) \text{diff}_j,$$  

(6)  

where $\alpha$ is the gain of $\tau$. During the training process of behavior cloning, because there is no $\tau_t$ in the dataset, $\alpha \tau$ is not reflected in (6). The effectiveness of applying the weight $\tau_t$ in $diff_j$ is analyzed in the fourth row of Table 1. $K_{r_j}$ in (4) is $\sigma^2_{diff_j}$, which is used to estimate the variance of the output of the network $\pi(s; \theta)$.

### 3. EXPERIMENTAL SETUP

#### 3.1 Vehicle and Hardware

The vehicle used in the experiments was a Hyundai HG 300, as shown in Fig. 5. The operating system of the laptop computer was Ubuntu 16.04, and the robot operating system (ROS) was used as a meta-OS platform. The GPU was Nvidia GTX 1080-ti (8 GB), and the CPU was 3.9 GHz Intel i9-8950HK. The steering wheel, accelerator, and brake were controlled by a micro controller unit using a proportional-integral-derivative (PID) controller.

![Fig. 5. Hardware System.](image)
Fig. 6. Parking lots used in the experiment. At intersections, traffic cones are used to guide vehicles to drive in one direction. The yellow line is the center of the drivable space. The red boxes represent obstacle vehicles that were present in the fifth experiment. (a) Yellow line is about 230 m long; this parking lot was used to collect the training dataset for imitation learning. (b) Yellow line is 139 m long. (c) Yellow line is 149 m long.

A front camera was attached 1.55 m above the ground and 0.25 m forward from the vehicle center. It was rotated about 20° in the pitch direction to minimize the shaded area of the bird’s-eye-view image. This camera comprised two lenses to capture a wide view of the environment (see Fig. 5). The field of view (FoV) of each lens was 120°, and the distortion of images was corrected.

The pure pursuit algorithm [34] was used to calculate the steering angle command \( \delta \) to reach the look-ahead point:

\[
\delta = \tan^{-1}\left( \frac{2L \sin \theta_l}{L_f} \right),
\]

where \( L \) is the wheelbase, and \( L_f \) is the distance between the positions of the vehicle and look-ahead point. \( \theta_l \) is the look-ahead heading, which is the difference between the heading of the vehicle and heading of the vector from the vehicle to the look-ahead point. The range of \( \delta \) was -540° to 540°.

The velocity command \( v \) (m/s) to reach the look-ahead point was proportional to \( \alpha_y \) which is the longitudinal distance between this point and the vehicle [34]. Thus,

\[
v = \frac{\alpha_y}{2},
\]

where the final \( v \) was set to half of \( \alpha_y \) for safety reasons. The range of \( v \) was 0.5 - 2.2 (desired velocity) m/s. The accelerator and brake commands for controlling the velocity were calculated with the PI controller.

3.2. Perception Network Training

Softmax cross-entropy was used as the loss function to train the perception network. The drivable and non-drivable probability values were inferred for each pixel, and the average loss of each pixel was calculated. The Otsu algorithm was used to determine the threshold value about the drivable probability. Before training, weights were assigned to initialize the network for efficient training. The encoder was initialized with weights trained on ImageNet data. The transposed convolution layers in the decoder were initialized using the scheme in [30] to segment two classes. The Adam optimizer with a learning rate of \( 10^{-5} \) was used to train the network. A weight decay of \( 5 \times 10^{-4} \) was applied to the network. Epochs were 10k, and the batch size was set to 128.

The training dataset (i.e., RGB-segmented images) was collected for three parking lots as shown in Fig. 6. As the vehicle was driven, one image per second was collected for 989 RGB images in total. The pixel annotation tool [37] was used to segment the collected images into a drivable class and non-drivable class. The RGB and segmented images were transformed into the bird’s-eye-view image and used to train the perception network. Eighty percent of the dataset was used for training, and the rest was used for validation.

3.3. Driving Policy Network Training

The loss function of the driving policy network is explained in Section 2.2.6. Parameter \( \alpha \) in (6) was set to 1.5. The Adam optimizer with a learning rate of \( 10^{-5} \) was used to train the network. The network was not initialized with pretrained weights. Epochs were 100k, and the batch size was set to 512.

The training dataset \( D \) was collected for only the parking lot shown in Fig. 6(a). The vehicle was driven from the start point to the finish point. To collect more data, the vehicle was turned around from the finish point and driven to the start point (totaling 460 m). The dataset was collected according to the method explained in Section 2.2.1, and the process was recorded as a video 2.

The threshold \( \tau \) of \( \hat{\tau} \) (discrepancy of two actions) in Algorithm 2 was set to 0.07. \( \tau \) was set to the maximum value at which the vehicle cannot collide with an obstacle by the action of the network when performing DAgger. This is because the higher this value, the more unsafe or near-collision data can be obtained. The threshold \( \chi \) of \( \hat{\chi}_{\text{var}} \) (variance of the action) was set to 0.1. The threshold \( \eta \) of \( \hat{\eta} \) (performance of the trained policy, line 19 in Algorithm 2) was set to 0.9.

2https://youtu.be/KOXFTEYL-xs
Table 1. DAgger Results

| Dataset (ca) | BC | i = 1 | 2 | 3 | 4 | 5 |
|-------------|----|------|---|---|---|---|
| 6425        | -  | 3258 | 2263 | 2088 | 1505 | 731 |
| Network/Total (\(\hat{\eta}_i\) ) | -  | 0.44 | 0.64 | 0.65 | 0.74 | 0.90 |
| Effect of (6) | -  | 0.62 | 0.65 | 0.85 | 0.89 | 0.91 |

*Note. BC represents for "behavior cloning".*

The "Network/Total" (third) row represents the percentage of network actions among the total actions used in each DAgger iteration \(i\). This is identical to \(100 \times \hat{\eta}_i\), where \(\hat{\eta}_i\) is expressed in line 19 of Algorithm 2.

The "Effect of (6)" (fourth) row compares the results of accumulating the error \(\xi_i\) per second for the state in the dataset between two trained policies that do and do not reflect the weight \(\alpha\xi_i\) in the loss function: \(\sum\xi_i\), which reflects \(\xi_i\) in (6). In this row, a lower value indicates a larger difference between the two policies and shows how much the error can be reduced by reflecting \(\xi_i\) in (6).

The final policy was obtained through 5 DAgger iterations \((i = 5)\). Increasing the number of DAgger iterations can improve performance, but may not significantly. In the case of our experiment, the difference between \(\hat{\eta}_{i=5}\) and \(\hat{\eta}_{i=6}\) was only 0.02. As the vehicle was being driven, data were collected at intervals of 0.05 s. Table 1 presents the number of collected data and percentage of executed network actions and the effect of applying \(\xi_i\) in (6).

3.4. General Model-based Motion-Planning Algorithms Used for Comparison

The tentacle [17] and VVF [20] algorithms were used to compare with the proposed method. These are representative and general algorithms of the candidate path selection and artificial field generation methods, respectively. The optimization, graph search, and incremental tree search methods could not be considered because they require global information and real-time calculation is difficult.

Among the candidate path selection algorithms, only tentacle’s the curvature of the candidate path gradually increases to account for the constraint that the steering angle cannot be changed by a large amount instantaneously. Thus, a vehicle can drive more smoothly and safely on a narrow or variable-curvature road with this algorithm than with the dynamic window approach (DWA) [15] and curvature velocity method (CVM) [16]. With regard to artificial field generation, the VVF algorithm is similar to the APF algorithm [19] and can be used without the goal point because it also considers the desired velocity.

The occupancy grid map was used as the input for the tentacle and VVF algorithms. The steering angle was calculated with each algorithm, and the velocity was set as inversely proportional to the calculated steering angle. For example, if the steering angle was zero, the velocity was set to the desired velocity (2.2 m/s); when the steering angle was maximum (540°), the velocity was set to the lowest velocity (0.5 m/s).

3.4.1 Tentacle Algorithm [17]

This algorithm has 16 candidate path sets depending on the velocity, and each candidate path set has 81 candidate paths. The cost for each candidate path is calculated with the objective function, and the candidate path with the smallest value is selected. The objective function has four terms: clearance, flatness, trajectory, and forwarding. The clearance term prefers to choose a candidate path with the fewest obstacles around it. The flatness term is similar to the clearance term and additionally considers the probability of a grid being occupied (non-drivable) for each cell in the occupancy grid map for smooth driving. The trajectory term selects a candidate path that can head to the global path. The forwarding term was further designed in this study to drive forward preferentially. This term selects the candidate path with the least curvature.

In experiments, a set of candidate paths for 2.2 m/s was used. The application ratio of the clearance, flatness, trajectory, and forwarding terms was: 1:0:0:0.3. The flatness term was not used because the occupancy grid map in this study did not have the occupied probability. In addition, the trajectory term could not be used because of the absence of global information. When the forwarding term was set to greater than 0.3, the oscillation problem was reduced, but the risk of collision was increased for large curvature changes. The clearance term included a detection range parameter to calculate the proportion of obstacles around the candidate path. This range was set to 0.35 m, which is the width of the vehicle (0.2 m) plus the safety distance (0.15 m). When this was increased further, the vehicle could avoid obstacles more safely, but more oscillation occurred in narrow drivable space.

3.4.2 VVF Algorithm [20]

Like the APF algorithm, the VVF algorithm has a repulsive field for obstacles and an attractive field for the goal point. Additionally, to follow the desired velocity and direction, the velocity field is reflected to the APF field. To drive along the combined field, the look-ahead point is searched by descending along the gradient of the field’s direction from the front of the vehicle.

In experiments, the repulsive, attractive, and velocity fields were set to a ratio of 1:0:0.5. The attractive field could not be used because global information (global path, localization data) was not used in this study. The direction of the velocity field was set so that the vehicle could drive forward. When the fields were combined, only the repulsive field was applied around obstacles with a range of 2.3 m. If the range was set greater than 2.3 m, the vehicle could avoid obstacles more safely, but more oscillations occurred when it passed through narrow drivable space.
4. EXPERIMENTAL RESULTS

The experimental results for the perception network and driving policy are presented here to demonstrate the effectiveness of the proposed method. The experiments were conducted at three parking lots without intersections, as shown in Fig. 6. There were no lanes in the drivable space, where the width and curvature changed rapidly. In addition, several unknown static obstacles were present.

In the perception network test, the accuracy and speed of the perception network were measured. In the driving policy test, the proposed method was compared with the tentacle and VVF algorithms. The driving results were recorded as videos\(^3\) and quantitatively evaluated according to a designed evaluation metric. The limitations of the tentacle and VVF algorithms were analyzed for each situation. In addition, a stability and time-delay of our method were analysed.

4.1. Perception Network Test Results

The performance of the perception network was tested with the validation dataset that were not used to train the network. The pixel accuracy was used as the evaluation metric:

\[
\text{Pixel Accuracy} = \frac{\text{correctly classified pixels}}{\text{total number of pixels}} \times 100\% , \tag{7}
\]

where the numerator is the number of pixels correctly predicted by the network. Table 2 presents the pixel accuracy results for the parking lots in Fig. 6.

Table 2. Perception Network Results

| Parking Lots     | Fig. 6(a) | Fig. 6(b) | Fig. 6(c) |
|------------------|-----------|-----------|-----------|
| Pixel Accuracy (7) [%] | 98.14     | 97.75     | 97.85     |

The drivable space is represented as the green areas in Figs. 8-11. On average, the execution speed of the perception network was 27.9 fps.

4.2. Quantitative Analysis of Driving Policy

4.2.1 Collision Rate

The collision rate was used as an evaluation metric to quantify the performance of each driving policy algorithm. This metric indicates the number of collisions per 100 m as the vehicle was driven in each parking lot:

\[
\text{collision rate} = \frac{\text{cnt}_{\text{col}}}{\text{len}_{\text{path}}} \times 100\% , \tag{8}
\]

where \(\text{cnt}_{\text{col}}\) represents the number of times a near-collision situation occurred. When the vehicle headed toward an obstacle and the distance was 0.5 m or less, the vehicle was stopped, and \(\text{cnt}_{\text{col}}\) was incremented. Then, the driving was resumed at a point along the reference path closest to the collision point, as indicated by the yellow line in Fig. 6. At this point, the vehicle could drive without a collision. The length of the reference path was \(\text{len}_{\text{path}}\). A lower collision rate indicated a safer driving policy. When the rate was 0, the vehicle could reach the finish point without any collision.

Table 3. Collision Rate

|                  | Parking Lots |
|------------------|--------------|
|                  | Fig. 6(a)    | Fig. 6(b)    | Fig. 6(c)    |
|                  | Trained      | Untrained    | Untrained    |
| **Imitation**    | **DAgger**   | **0 (0)**    | **0 (0)**    | **0 (0)**    |
| Learning (Proposed) | Model-based  | Tentacle     | VVF          |
|                  | Motion       | 1.12 (0.95)  | 1.87 (2.01)  | 1.47 (1.15)  |
| Planning         | VVF          | 1.38 (1.29)  | 2.01 (2.15)  | 1.07 (0.93)  |

*Note. The values represent the average collision rate according to (8) over five trials. The values in parentheses indicate additional test results where the vehicle turned from the finish point and drove to the start point.

Table 3 presents the test results for the collision rate at the three parking lots over five trials. In the experiment, each algorithm was used to travel a distance of 5180 m. The vehicle using D Agger method did not encounter any collisions. Even in the untrained parking lot with obstacles of different sizes and shapes, the vehicle drove without any collisions. This result demonstrates that the proposed method has generality. The tentacle and VVF algorithms resulted in averages of 1.428 and 1.471 collisions per 100 m, respectively. Several unsafe or near-collision (near-collision) situations occurred with the tentacle and VVF algorithms as described in the next subsections.

4.2.2 Safe Distance Range Ratio

Fig. 7. Results of Safe Distance Range Ratio; The blue area is the safe distance range, and its ratio is a measure of how much drivable space (green) exist within the blue area.

Additionally, in order to evaluate a collision safety with obstacles, a ratio of the drivable space within 1.0 m range
from the end of ego vehicle’s bumper was measured:

\[ \text{safe ratio} = \frac{N_{dri}}{N_{ran}}, \]  

(9)

where \( N_{dri} \) is the number of pixels for the drivable area among \( N_{ran} \). \( N_{ran} \) is the number of pixels around 1.0 m range from the end of ego vehicle’s bumper, which is indicated in blue range in Fig. 7. By measuring this ratio, it can be possible to measure how safely the vehicle can maintain a safe distance from obstacles on average. This range and ratio is shown in Fig. 7 and indicated in Table 4. The proposed algorithm DAgger has the highest safe distance range ratio.

Table 4. Safe Distance Range Ratio

|                  | Parking Lots |        | Parking Lots |        |
|------------------|--------------|--------|--------------|--------|
|                  | Fig. 6(a)    | Trained Environment | Fig. 6(b)    | Untrained Environment | Fig. 6(c)    | Untrained Environment |
| Imitation Learning | DAgger       | 0.83   | 0.72          | 0.91   |
|                  | (Proposed)   |        |               |        |
| Model-based Motion Planning | Tentacle | 0.69   | 0.63          | 0.81   |
|                  | VVF          | 0.71   | 0.64          | 0.85   |

*Note. The values represent the average safe distance range ratio over five trials.

4.3. Limitations of Tentacle Algorithm

In the tentacle algorithm test, the vehicle drove near the boundary between the drivable and the non-drivable spaces rather than the center of the drivable space after avoiding obstacles or escaping the corner, which is shown in the leftmost image of Fig. 8. This is because the tentacle algorithm selected the most forward-facing candidate path with no obstacle among the candidate paths. Then, the vehicle drove at the minimum distance from side obstacles, which increased the possibility of collision. In the same situation, DAgger tried to direct the vehicle toward the center of the drivable space. This is because, when the training dataset was collected, experts kept the distance between the vehicle and obstacles as large as possible by considering the overall pattern of the occupancy grid map.

The second to fourth images in Fig. 8 show that, when the vehicle was driving on the side of the drivable space and there was an obstacle in front, the vehicle was unable to avoid the obstacle because of the lack of sufficient space to avoid it. In other situations, even when a vehicle drove along the center of the drivable space and avoided obstacles, it did not avoid the obstacle with sufficient clearance. These are because the tentacle algorithm chose the candidate path with the least spacing to avoid obstacles. In contrast, DAgger tried to avoid obstacles with sufficient safe distance in advance.

Fig. 9. Problems with the VVF algorithm; The white arrows represent the field direction. (a) Oscillation in a narrow drivable space. (b) The vehicle could not enter the side of the drivable space in advance at a right-angled corner. DAgger did not encounter any problems in situations (a) and (b).
4.4. Limitations of VVF Algorithm

In the VVF test, an oscillation problem occurred in narrow drivable spaces where the vehicle frequently turned left and right as shown in Fig. 9(a). In such spaces, because only a repulsive field was applied, the magnitudes of the fields from two obstacles were almost the same, but the directions were opposite. Thus, the position of the look-ahead point changed frequently in the opposite directions. This problem may be reduced by decreasing the gain and the range of the repulsive force. However, the probability of collision would be increased in other situations, especially where the curvature changed significantly. With DAgger, the vehicle drove stably without oscillation by imitating the expert who drove toward the middle of the drivable space even in narrow spaces.

As shown in Fig. 9(b), with VVF, the vehicle could not enter the drivable space when the curvature changed rapidly, such as right-angled corners. This problem may be addressed with the global information, where the goal point would be used as an attractive field. In contrast, this problem did not occur with DAgger. This is because, when the training set for DAgger was collected in this situation, the expert selected a look-ahead point for which the vehicle could drive the furthest without causing a collision.

4.5. Limitations of Both Tentacle and VVF Algorithms

Figure 10 shows the problems of the VVF and tentacle algorithms when the curvature and width of the drivable space changed more than the space where the vehicle was currently driving. The vehicle headed into the drivable space on the side of the adjacent obstacle before sufficiently avoiding it. For the tentacle algorithm, this is because it selected the path with the fewest obstacles among the candidate paths. The candidate path set according to the desired velocity (2.2 m/s) was limited in its ability to handle these situations. For the VVF algorithm, the generated field could not sufficiently consider the nearest obstacles. To address this problem, the range of the repulsive field should be increased. Meanwhile, DAgger tried to dodge the nearest obstacle until DAgger successfully avoided it because it learned the pattern of preferentially avoiding the nearest obstacles from experts.

4.6. Driving Results on Noisy Occupancy Grid Map

The occupancy grid map was not recognized accurately in complex and shadowy environments (i.e., noisy state) because the learning data for such situations were insufficient to train the perception network. Data with the noisy state were contained in training data, so the trained network could learn some patterns for the noise and deal with the noisy state. As can be seen from the experiment in Fig. 11, a vehicle could drive without collision, even though there were noise in one trained environment (see Fig. 11(a), 11(b), and 11(c)) and two untrained environments (see Fig. 11(d), 11(e), 11(f), and 11(g)). However, the Tentacle and VVF algorithms encountered several problems.

As shown in Fig. 11(a), the boundary between the speed bump and road was erroneously recognized as non-drivable space (i.e., noise). In this situation, DAgger was not affected by the noise because it trained the driving pattern from the overall shape of the state. However, the vehicle drove unstably with the Tentacle and VVF algorithms to avoid the misrecognized non-drivable space.

Figures 11(b) and 11(c) show situations where noise was caused by shadows. With DAgger, the vehicle drove towards the drivable space with fewer oscillations than Tentacle and VVF. This is because the DAgger training dataset contained similar situations, where the expert selected an action without being affected by the noise. With the tentacle and VVF algorithms, however, the vehicle in the Fig. 11(b) situation avoided the shadows and then drove toward the largest drivable space blocked by obstacles, so it became unable to drive any further. These algorithms also had larger oscillation problems than DAgger especially in the Fig. 11(c) situation.

Figures 11(d), 11(e), and 11(f) present situations in which a non-drivable space was recognized as drivable space. In detail, not only the non-drivable space at the curb
DAgger did not encounter any problems in this situation. However, the vehicle could not drive smoothly or headed towards obstacles with Tentacle and VVF. (a) Noise from misrecognition; (b), (c), and (g) Noise by shadow; (d), (e), and (f) Noise at the road boundary (misrecognition).

(i.e., boundary of the drivable space) but also the space behind the curb was recognized as drivable space. With DAgger, the vehicle tried to drive toward the largest drivable space, except for behind the curb. However, tentacle was influenced by the noise at the curb, which it detected to be drivable space. So, the vehicle was headed to the curb. VVF was less affected than the tentacle algorithm, but the vehicle was unable to drive toward the largest drivable space (see Figs. 11(d) and 11(e)).

As shown in Fig. 11(g), the vehicle with the VVF algorithm took actions to avoid the noise caused by a shadow next to the obstacle when passing through a narrow space. For the same situation, DAgger and the tentacle algorithm did not respond sensitively, and no problem occurred.

4.7. Analyses of Stability and Time-delay

Although the stability cannot be theoretically proven in imitation learning, it has been experimentally confirmed that there is no problem in the parking lot environment through sufficient training with DAgger. For a dynamic obstacle, if the speed is lower than about 10 km/h, it can be treated similarly to a static obstacle, and there was no problem in the actual experiment.

The time-delay problem did not occur because all parts were calculated within a defined control period of the proposed algorithm, 50 ms. The perception and driving networks calculated each outputs within 20 ms and 10 ms. Besides, obtaining control commands to reach the look-ahead point were calculated in 1 ms, and the vehicle responded to these commands within 30 ms.

5. CONCLUSION

In this study, an autonomous driving method using vision-based occupancy grid map and imitation learning is proposed to deal with unstructured environments such as parking lots. With the proposed method, the vehicle can drive toward the drivable space while avoiding obstacles reactively in real-time without using a global map and localization. Besides, it does not need to model the driving policy and tune model-parameters of the policy. The occupancy grid map obtained by the U-net-based deep neural network is used as an input for imitation learning, where the driving patterns of experts in various and complex environments are learned. In experiments, a real autonomous vehicle was trained with DAgger in one parking lot and tested in three parking lots (1036 m) without intersections five times each (totaling 5180 m).

With DAgger, the vehicle could drive more smoothly and safely than with the tentacle and VVF algorithms in environments where the width and curvature of the drivable space varied significantly. Especially, DAgger was more robust when the occupancy grid map was not accurately perceived or was noisy due to a shadow. With regard to the collision rate, DAgger did not cause any collision, but the tentacle and VVF algorithms caused 1.42 and 1.47 collisions per 100 m, respectively. This is because the tentacle and VVF algorithms require different parameters to accommodate different complex situations. In contrast, DAgger trains the deep neural network with
numerous weight parameters using expert driving data for these situations. Future work will focus on developing the proposed method to environments with intersections and dynamic obstacles.

REFERENCES

[1] H. Banzhaf, D. Nienhüser, S. Knoop, and J. M. Zöllner, “The future of parking: A survey on automated valet parking with an outlook on high density parking,” in 2017 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2017, pp. 1827–1834.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” IEEE Transactions on Intelligent Vehicles, vol. 1, no. 1, pp. 33–55, 2016.

[3] D. González, J. Pérez, V. Milanés, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135–1145, 2015.

[4] J. Shin, D. Kwak, and K. Kwak, “Model predictive path planning for an autonomous ground vehicle in rough terrain,” International Journal of Control, Automation and Systems, vol. 19, no. 6, pp. 2224–2237, 2021.

[5] M. A. Mousavi, B. Moshiri, and Z. Heshmati, “A new predictive motion control of a planar vehicle under uncertainty via convex optimization,” International Journal of Control, Automation and Systems, vol. 15, no. 1, pp. 129–137, 2017.

[6] M. R. H. Al-Dahhan and K. W. Schmidt, “Voronoi boundary visibility for efficient path planning,” IEEE Access, vol. 8, pp. 134 764–134 781, 2020.

[7] H. Niu, A. Savvaris, A. Tsourdos, and Z. Ji, “Voronoi-visibility roadmap-based path planning algorithm for unmanned surface vehicles,” The Journal of Navigation, vol. 72, no. 4, pp. 850–874, 2019.

[8] J. C. Mohanta and A. Keshari, “A knowledge based fuzzy-probabilistic roadmap method for mobile robot navigation,” Applied Soft Computing, vol. 79, pp. 391–409, 2019.

[9] E. W. Dijkstra et al., “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

[11] S. Shin, J. Ahn, and J. Park, “Desired orientation rrt (dorr) for autonomous vehicle in narrow cluttered spaces,” in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016, pp. 4736–4741.

[12] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path planning for autonomous driving in unknown environments,” in Experimental Robotics. Springer, 2009, pp. 55–64.

[13] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, “Anytime search in dynamic graphs,” Artificial Intelligence, vol. 172, no. 14, pp. 1613–1643, 2008.

[14] M. Hoy, A. S. Matveev, and A. V. Savkin, “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey,” Robotics, vol. 33, no. 3, pp. 463–497, 2015.

[15] M. Missura and M. Bennewitz, “Predictive collision avoidance for the dynamic window approach,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 8620–8626.

[16] J. López, C. Otero, R. Sanz, E. Paz, E. Molinos, and R. Barea, “A new approach to local navigation for autonomous driving vehicles based on the curvature velocity method,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 1751–1757.

[17] H. Mouhagir, R. Talj, V. Cherafaoui, F. Aioun, and F. Guillemand, “Evidential-based approach for trajectory planning with tentacles, for autonomous vehicles,” IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 8, pp. 3485–3496, 2019.

[18] V. Olunloyo and M. Ayomoh, “Autonomous mobile robot navigation using hybrid virtual force field concept,” European Journal of Scientific Research, vol. 31, no. 2, pp. 204–228, 2009.

[19] S. S. Ge and Y. J. Cui, “New potential functions for mobile robot path planning,” IEEE Transactions on Robotics and Automation, vol. 16, no. 5, pp. 615–620, 2000.

[20] J. Wang, J. Wu, and Y. Li, “The driving safety field based on driver–vehicle–road interactions,” IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 4, pp. 2203–2214, 2015.

[21] J. Chen, B. Yuan, and M. Tomizuka, “Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019, pp. 2884–2890.

[22] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Mickelthwaite, N. Griffiths, A. Shah et al., “Urban driving with conditional imitation learning,” in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 251–257.

[23] W. Youn, H. Ko, H. Choi, I. Choi, J.-H. Baek, and H. Myung, “Collision-free autonomous navigation of a small uav using low-cost sensors in gps-denied environments,” International Journal of Control, Automation and Systems, vol. 19, no. 2, pp. 953–968, 2021.

[24] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker, and C. Jawahar, “Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments,” in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2019, pp. 1743–1751.

[25] D.-H. Lee, K.-L. Chen, K.-H. Liou, C.-L. Liu, and J.-L. Liu, “Deep learning and control algorithms of direct perception for autonomous driving,” Applied Intelligence, vol. 51, no. 1, pp. 237–247, 2021.

[26] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Mickelthwaite, N. Griffiths, A. Shah et al., “Urban driving with conditional imitation learning,” arXiv preprint arXiv:1912.00177, 2019.
[27] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, “Inverse reward design,” in Advances in neural information processing systems, 2017, pp. 6765–6774.

[28] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.

[29] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, “Ensembledagger: A bayesian approach to safe imitation learning,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 5041–5048.

[30] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, “Multinet: Real-time joint semantic reasoning for autonomous driving,” in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 1013–1020.

[31] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[32] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 627–635.

[33] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?” in Advances in neural information processing systems, 2017, pp. 5574–5584.

[34] J. Ahn, S. Shin, M. Kim, and J. Park, “Accurate path tracking by adjusting look-ahead point in pure pursuit method,” International Journal of Automotive Technology, vol. 22, no. 1, pp. 119–129.

[35] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, “Hg-dagger: Interactive imitation learning with human experts,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 8077–8083.

[36] J. Byrd and Z. Lipton, “What is the effect of importance weighting in deep learning?” in International Conference on Machine Learning. PMLR, 2019, pp. 872–881.

[37] A. Bréhéret, “Pixel Annotation Tool.” https://github.com/abreheret/PixelAnnotationTool, 2017.